\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*Chapter 07 뷰 영역 구현\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

: 앞서 6장에서는 컨트롤러의 구현 방법에 대해서 살펴보았다.

컨트롤러는 최종적으로 결과를 출력 할 뷰와 뷰에 전달할 객체를 담고 있는 ModelAndView 객체를 리턴한다.

DispatcherServlet은 ViewResolver를 사용하여 결과를 출력할 View 객체를 구하고, 구한 View 객체를 이용하여 내용을 생성한다.

[[:HandlerMapping]] [[:Controller]]

|| ||

|| ||

|| 2: 요청URL과 Mapping되는 4:처리요청 || 5: ModelAndView 리턴

|| Controller검색 ||

|| ||

|| ||

|| ||

|| ||

|| ||

[[클라이언트]] ============= [[: DispacherServlet]]

1 : 처리 요청 || ||

|| ||

|| ||

|| ||

|| ||

|| ||

|| ||

|| 6: 응답 출력 요청 || 6: Controller의 실행 결과를

|| || 보여줄 View 검색

[[:View]] [[:ViewResolver]]

\*\*\*SPRING MVC의 처리 흐름

본 장에서는 SPRING이 제공하는 ViewResolver에 대해서 살펴보고, JSP, Tiles, Velocity와 같은 뷰 구현 기술을 SPRING에 적용시키는

방법을 살펴볼 것이다.

[index.jsp]

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>스프링 예제</title>

</head>

<body>

<ul>

<li>

<a href="<c:url value='/jsp/login/login.do'/>">/jsp/login/login.do</a>

(viewjsp/loginForm.jsp, &lt;form:\* /&gt; 태그 및 &lt;spring:message&gt; 태그 예제)

</li>

<li>

<a href="<c:url value='/jsp/member/regist.do'/>">/jsp/member/regist.do</a>

(viewjsp/registMemberForm.jsp, &lt;form:label/&gt; 태그 예제)

</li>

<li>

<a href="<c:url value='/jsp/articleForm.do'/>">/jsp/articleForm.do</a>

(viewjsp/article/\*.jsp, &lt;form:form&gt; 태그의 RESTful 지원 예제)

</li>

<li>

<a href="<c:url value='/jsp/changeLanguage'/>?lang=ko">/jsp/changeLanguage?lang=ko</a>

<a href="<c:url value='/jsp/changeLanguage'/>?lang=en">/jsp/changeLanguage?lang=en</a>

(LocaleChangeController, LocaleResolver 직접 이용 Locale 변환 예제)

<br/>

링크 클릭 후, <a href="<c:url value='/jsp/login/login.do'/>">/jsp/login/login.do</a>에서 라벨 확인

</li>

<li>

<a href="<c:url value='/jsp/login/login.do'/>?language=ko">/jsp/login/login.do?language=ko</a>

<a href="<c:url value='/jsp/login/login.do'/>?language=en">/jsp/login/login.do?language=en</a>

(LocaleChangeInterceptor를 이용한 Locale 변환 예제)

</li>

<li>

<a href="<c:url value='/tiles2/login/login.do'/>">/tiles2/login/login.do</a>

(tiles2def/\* 및 viewtiles2/loginForm.jsp, Tiles2 예제)

</li>

<li>

<a href="<c:url value='/tiles2/member/regist.do'/>">/tiles2/member/regist.do</a>

(tiles2def/\* 및 viewtiles2/registMemberForm.jsp, Tiles2 예제)

</li>

<li>

<a href="<c:url value='/vm/login/login.do'/>">/vm/login/login.do</a>

(viewvm/loginForm.vm, Velocity 예제, #springBind, #springShowErrors, #springMessage)

</li>

<li>

<a href="<c:url value='/vm/member/regist.do'/>">/vm/member/regist.do</a>

(viewvm/registMemberForm.vm, Velocity 예제, #springFormInput 등 폼 관련 매크로)

</li>

<li>

<a href="<c:url value='/vmLayout/login/login.do'/>">/vmLayout/login/login.do</a>

(viewvmlayout/template/\*, viewvmlayout/loginForm.vm, VelocityLayoutViewResolver 예제)

</li>

<li>

<a href="<c:url value='/download/file'/>">/download/file</a>

(DownloadController 및 DownloadView 예제)

</li>

<li>

<a href="<c:url value='/download/pageRanks'/>">/download/pageRanks</a>

(PageRanksController 및 PageRanksView 엑셀 예제)

</li>

<li>

<a href="<c:url value='/download/pageReport'/>">/download/pageReport</a>

(PageReportController 및 PageReportView PDF 예제)

</li>

<li>

<a href="<c:url value='/download/pageXmlReport'/>">/download/pageXmlReport</a>

(PageReportController 및 MarshallingView XML 응답 예제)

</li>

<li>

<a href="<c:url value='/download/pageJsonReport'/>">/download/pageJsonReport</a>

(PageReportController 및 MappingJacksonJsonView JSON 응답 예제)

</li>

</ul>

</body>

</html>

===================================================================================================

1. ViewResolver 설정

: SPRING 컨트롤러는 뷰에 의존적이지 않다. 컨트롤러는 아래 Code와 같이 결과를 생성할 뷰의 이름만 지정할 뿐이다.

@Controller

public class HelloController{

@RequestMapping("/hello.do")

public String hello(){

//처리 결과를 뷰 이름 지정

return "hello";

}

}

컨트롤러가 지정한 뷰 이름으로부터 응답 결과 화면을 생성하는 View 객체는 ViewResolver가 구한다. SPRING은 몇 가지 ViewResolver

구현 Class를 제공하고 있는데, 이중 주요 ViewResolver 구현 Class는 아래와 같다.

[[ViewResolver 구현 Class]] [[설 명]]

================================================================================

InternalResourceViewResolver 뷰 이름으로부터 JSP나 Tiles 연동을 위한 View 객체를 리턴한다.

----------------------------------------------------------------------------------------------------------------------------------------------------------------

VelocityViewResolver 뷰 이름으로부터 Velocity 연동을 위한 View 객체를 리턴한다.

----------------------------------------------------------------------------------------------------------------------------------------------------------------

VelocityLayoutViewResolver VelocityViewResolver와 동일한 기능을 제공하며, 추가로 Velocity의 레이아웃 기능을

제공한다.

----------------------------------------------------------------------------------------------------------------------------------------------------------------

BeanNameViewResolver 뷰 이름과 동일한 이름을 갖는 빈 객체를 View 객체로 사용한다.

----------------------------------------------------------------------------------------------------------------------------------------------------------------

ResourceBundleViewResolver 뷰 이름과 View 객체간의 Mapping정보를 저장하기 위해 Resource 파일을 사용한다.

----------------------------------------------------------------------------------------------------------------------------------------------------------------

XmlViewResolver 뷰 이름과 View 객체간의 Mapping정보를 저장하기 위해 XML 파일을 사용한다.

----------------------------------------------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1.1 ViewResolver Interface

: ViewResolver Interface는 다음과 같이 정의되어 있다.

package org.springframework.web.servlet;

import java.util.Locale;

public interface ViewResolver{

View resolveViewName(String viewName, Locale locale) throws Exception;

}

ViewResolver는 뷰 이름과 지역화를 위한 Locale을 파라미터로 전달받으며, Mapping되는 View 객체를 리턴한다. 만약, Mapping되는 View 객체가

존재하지 않으면 null을 리턴한다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1.2 View 객체

: ViewResolver는 응답 결과를 생성할 뷰 객체를 리턴한다. 모든 뷰 Class는 View Interface를 구현하고 있으며, View Interface는

다음과 같이 정의되어 있다.

package org.springframework.web.servlet;

import java.util.Map;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public interface View{

String RESPONSE\_STATUS\_ATTRIBUTE = View.clss.getName() + ".responseStatus";

String getContentType();

void render(Map<String,?> model, HttpServletRequest request) throws Exception;

}

getContentType() 메서드는 "text/html"과 같은 응답 결과의 컨텐츠 타입을 리턴한다. render() 메서드는 실제로 응답 결과를 생성한다.

render() 메서드의 첫 번째 파라미터인 model에는 컨트롤러가 리턴한 ModelAndView 객체의 모델 데이터가 전달된다. 각각의 View 객체는

이 모델 데이터로부터 응답 결과를 생성하는 데 필요한 정보를 구한다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1.3 InternalResourceViewResolver 설정

: InternalResourceViewResolver Class는 JSP나 HTML 파일과 같이 WEB Application의 내부 Resource을 이용하여 뷰를 생성하는 AbstractUrlBasedView

타입의 뷰 객체를 리턴한다. 기본적으로 사용하는 View Class는 InternalResourceView Class이다.

InternalResourceViewResolver Class는 다음과 같이 설정한다.

<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"

p:prefix="/WEB-INF/viewjsp" p:suffix=".jsp">

</bean>

InternalResourceViewResolver는 컨트롤러가 지정한 뷰 이름으로부터 실제로 사용될 뷰를 선택하는데, 이 때 컨트롤러가 지정한 뷰 이름 앞뒤로

prefix 프로퍼티와 suffix 프로퍼티를 추가한 값이 실제로 사용될 Resource의 경로가 된다.

예를 들어, 컨트롤러에서 다음과 같이 뷰 이름을 지정했다고 하자.

ModelAndView mav = new ModelAndView("hello");

return mav;

이 경우 InternalResourceViewResolver가 사용하는 Resource의 경로는 다음과 같이 결정된다.

/WEB-INF/viewjsp/hello.jsp

--------------------- |------

prefix | suff

|

컨트롤러가 지정한 논리적 뷰 이름

기본적으로 사용되는 InternalResourceView Class는 단순히 지정한 Resource 경로로(일반적으로 JSP) 요청을 전달한다. 만약, SPRING의 국제화 관련

정보를 JSTL에서 사용하고 싶다면, 다음과 같이 JstlView Class를 viewClass 프로퍼티로 지정하면 된다.

<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"

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

p:prefix="/WEB-INF/viewjsp" p:suffix=".jsp"/>

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1.4 BeanNameViewResolver 설정

: BeanNameViewResolver Class는 뷰 이름과 동일한 이름을 갖는 빈을 뷰 객체로 사용한다. BeanNameViewResolver는 주로

커스텀 View Class를 뷰로 사용해야 하는 경우에 사용된다. 예를 들어, 파일 Download를 위한 정보를 읽어와 뷰에 전달하는

컨트롤러는 Download 관련 정보를 뷰에 전달할 것이다.

@Controller

public class DownloadController{

@RequestMapping("/download.do")

public ModelAndView download(HttpServletRequest request, HttpServletResponse response){

File downloadFile = getFile(request);

return new ModelAndView("download", "downloadFile", downloadFile);

}

...

}

위 결과를 보여 줄 View Class가 DownloadView Class라고 하자. 이 경우 DownloadView Class를 "download" 이름으로

빈에 등록하고 ViewResolver로 BeanNameViewResolver Class를 사용하면 된다.

<bean id="viewResolver" class="org.springframework.web.servlet.view.BeanNameViewResolver"/>

<bean id="download" class="madvirus.spring.chap07.view.DownloadView"/>

위 Code와 같이 설정을 하면 DownloadController Class의 처리 결과를 DownloadView가 생성하게 된다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1.5 XmlViewResolver 설정

: XmlViewResolver는 BeanNameViewResolver와 마찬가지로 뷰 이름과 동일한 이름을 갖는 빈을 뷰 객체로 사용한다.

차이점이 있다면, XmlViewResolver는 별도의 Xml 설정 파일로부터 빈 객체를 검색한다는 것이다. XmlViewResolver는

다음과 같이 location 프로퍼티를 이용하여 외부 설정 파일의 경로를 지정한다.

<bean id="viewResolver" class="org.springframework.web.servlet.view.XmlViewResolver" p:location="/WEB-INF/nonHtml-view.xml"/>

location 프로퍼티의 값을 지정하지 않을 경우 기본 값은 "/WEB-INF/views.xml" 이다. location 프로퍼티에 지정하는 파일은 SPRING

XML 설정 파일이다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1.6 ResourceBundleViewResolver 설정

: ResourceBundleViewResolver는 리소스 번들(프로퍼티 파일)로 부터 뷰 이름과 Mapping되는 View Class를 구한다. ResourceBundleViewResolver는

ResourceBundleMessageSource와 마찬가지로 basename 프로퍼티나 basenames 프로퍼티를 이용해서 뷰 Class 정보를 저장한 리Source

경로를 명시한다. 아래는 설정 예이다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1.7 다수의 ViewResolver 설정하기

: 하나의 DispatcherServlet은 한 개 이상의 ViewResolver를 설정할 수 있도록 하고 있다. 다수의 ViewResolver를 설정한 경우 "order" 프로퍼티를

이용하여 뷰 이름을 검사할 ViewResolver의 순서를 결정할 수 있다. "order" 프로퍼티의 값이 작은 ViewResolver가 높은 우선 순위를 갖는다.

만약, 우선순위를 명시하지 않으면 Integer.MAX\_VALUE를 "order" 프로퍼티의 값으로 갖는다. (즉, 가장 낮은 우선순위를 갖게 된다.)

ViewResolver 구현 Class가 org.springframework.core.Ordered Interface를 구현하지 않은 경우, 해당 ViewResolver 구현 Class 가장 낮은

우선 순위를 갖게 된다. 하지만, SPRING이 제공하는 ViewResolver 구현 Class는 모두 Ordered Interface를 구현하고 있다.

DispatcherServlet은 "order" 프로퍼티의 값이 작은, 즉, 우선 순위가 높은 ViewResolver에게 뷰 객체를 요청한다. 만약, 우선순위가 높은 ViewResolver가

null을 리턴하면, 그 다음 우선순위를 갖는 ViewResolver에 뷰를 요청한다. 뷰 객체를 구하면, 해당 뷰 객체를 이용하여 응답 결과를 생성한다.

아래 Code는 "order" 프퍼티의 설정 예다.

<bean class="org.springframework.web.servlet.view.XmlViewResolver"

p:location="/WEB-INF/nonHtml-view.xml"

p:order="0"/>

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"

p:prefix="/WEB-INF/viewjsp/"

p:suffix=".jsp"

p:order="1"/>

우선 순위를 결정할 때 주의할 점은 InternalResourceViewResolver는 마지막 우선 순위를 갖도록 지정해야 한다는 점이다. 그 이유는 InternalResourceViewResolver는

항상 뷰 이름에 Mapping되는 뷰 객체를 리턴하기 때문이다. InternalResourceViewResolver는 null을 리턴하지 않기 때문에, InternalResourceViewResolver의

우선 순위가 높을 경우 우선순위가 낮은 ViewResolver는 사용되지 않게 된다.

VelocityViewResolver와 (하위 Class인) VelocityLayoutViewResolver의 경우에도 우선순위를 지정할 때 주의해 주어야 한다. 이 두 ViewResolver는 뷰 이름에

Mapping되는 Velocity 템플릿 파일이 존재하지 않을 경우 null을 리턴하지 않고 예외를 발생시킨다. 따라서, VelocityViewResolver 역시 가장 낮은 우선순위를 갖도록 해야 한다.

===================================================================================================

2. HTML 특수 문자 처리 방식 설정

: SPRING은 각 뷰 기술과 관련하여 Message나 커맨드 객체의 값을 출력할 수 있는 기능을 제공하고 있다. JSP를 뷰 기술로 사용할 경우 다음의 커스텀 태그를 이용해서

Message를 출력할 수 있다.

<title><spring:message code="login.form.title"/></title>

만약 위 커스텀 태그가 출력하는 값이 '<입력폼>'이라고 하자. 이때, '<' 와 '>' 는 HTML에서 특수 문자이기 때문에 '&lt;' 나 '&gt;'와 같은 엔티티 Reference로 변환해

주어야 원하는 값이 출력된다.

SPRING은 이런 특수 문자를 어떻게 처리할 지의 여부를 defaultHtmlEscape 컨텍스트 파라미터를 통해서 지정할 수 있다. 아래 Code는 설정 예이다.

<?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\_2\_5.xsd"

id="WebApp\_ID" version="2.5">

<context-param>

<param-name>defaultHtmlEscape</param-name>

<param-value>false</param-value>

</context-param>

...

</web-app>

defaultHtmlEscape 컨텍스트 파라미터의 값을 true로 지정하면 SPRING이 제공하는 커스텀 태그나 Velocity 매크로는 HTML의 특수 문자를 엔티티 Reference로 치환한다.

반면에 defaultHtmlEscape 컨텍스트 파라미터의 값이 false이면, 특수 문자를 그대로 출력한다.

참고로, defaultHtmlEscape 컨텍스트 파라미터의 기본 값은 true이다.

===================================================================================================

3. JSP를 이용한 뷰 구현

: JSP를 뷰로 사용하려면 앞서 살펴봤듯이 InternalResourceViewResolver를 사용하면 된다. 아래 Code와 같이 suffix 프로퍼티의 값을 .jsp로 지정함으로써 논리적 뷰 이름을

특정 JSP로 Mapping할 수 있다.

<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"

p:prefix="/WEB-INF/viewjsp/"

p:suffix=".jsp"/>

</bean>

위 설정을 보면 /WEB-INF 디렉터리의 하위 디렉터리에 JSP 파일을 위치시키고 있다. 이렇게 /WEB-INF 디렉터리에 뷰를 위한 JSP를 위치시키는 이유는 클라이언트가

뷰를 위한 JSP에 직접 접근하는 것을 막기 위함이다. /WEB-INF 디렉터리는 특수한 디렉터리로서 Servlet/JSP Container는 클라이언트가 /WEB-INF 경로로 직접 접근하는

것을 제한하고 있다.

SPRING은 JSP에서 사용할 수 있는 커스텀 태그를 제공하고 있다. 본 절에서는 이 중에서 Message 국제화를 위한 <spring:message> 커스텀 태그, 커맨드 객체와 HTML <form>

태그 사이의 연동을 위한 커스텀 태그, 그리고 에러 Message 출력을 위한 커스텀 태그에 대해서 살펴보도록 하겠다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

3.1 Message 출력을 위한 <spring:message> 커스텀 태그

: SPRING은 MessageSource로 부터 Message를 가져와 출력해 주는 <spring:message> 커스텀 태그를 제공하고 있다. <spring:message> 커스텀 태그는 다음과 같이

code 속성을 이용하여 읽어 올 Message의 Code를 지정한다.

<%@ page contentType="text/html; charset=EUC-KR" %>

<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title><spring:message code="login.form.title"/></title>

</head>

<body>

<form:form commandName="login">

<form:errors />

<p>

<label for="loginType"><spring:message code="login.form.type" /></label>

<form:select path="loginType" items="${loginTypes}" />

</p>

...

<p>

<input type="submit" value="<spring:message code="login.form.submit" />">

</p>

</form:form>

</body>

</html>

위 Code에서 사용되는 Message를 포함하고 있는 Message 리Source 파일은 다음과 같이 각 Code 값에 해당하는 Message를 설정하고 있을 것이다.

login.form.title=Login Form

login.form.type=Login Type

login.form.id=ID

login.form.password=Password

login.form.submit=Login

Message 리Source 파일은 {n} 형식을 이용하여 변하는 부분을 명시할 수 있다.

greeting=\uc804 {0} \uc785\ub2c8\ub2e4. "{1}"

<spring:message> 커스텀 태그는 arguments 속성을 이용하여 {n}에 들어갈 값을 설정할 수 있다. 이 때, 각 값은 콤마를 이용하여 구분한다.

아래 Code는 arguments 태그의 사용 예를 보여 주고 있다.

<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>

...

<spring:message code="greeting" arguments="${me}, ${greeting}"/>

<spring:message> 태그는 주어진 태그에 해당하는 Message가 존재하지 않는다면 예외를 발생시킨다. Code에 해당하는 Message가 없을 때 예외를

발생시키는 대신 지정한 Message를 출력하고 싶다면 text 속성에 기본 Message를 입력하면 된다.

<spring:message code="no\_code" text="Code가 없습니다."/>

출력되는 Message에 '<' 나 '&' 와 같이 HTML에서 특수하게 처리되는 문자가 포함되어 있을 경우, 알맞게 처리해 주어야 할 것이다.

그럼, '<', '&' 와 같은 특수한 문자가 '&lt;','&amp;' 와 같이 변경되어 출력된다.

htmlEscape 속성 값을 지정하지 않으면, defaultHtmlEscape 컨텍스트 파라미터에서 지정한 값을 사용한다.

Java스크립트에서 <spring:message> 태그가 생성한 문자열을 변수 값으로 사용하고 싶은 경우에는 javaScriptEscape 속성의 값을 true로 지정하면 된다.

javaScriptEscape 속성의 값이 true인 경우, 작은 따옴표나 큰 따옴표와 같은 문자를 \' 나 \" 와 같은 Java스크립트 특수 문자로 치환한다.

<script type="text/javascript">

var value = '<spring:message code="title" javaScriptEscape="true"/>'

</script>

...

<input type="submit" value='<spring:message code="login.form.submit" htmlEscape="false"/>'/>

<spring:message> 태그가 생성한 Message를 출력하지 않고 request나 session과 같은 기본 객체의 속성에 저장할 수도 있다. 아래는 사용 예이다.

<spring:message code="login.form.password" var="label" scope="request"/>

${label} : <input type=.../>

var 속성은 <spring:message> 태그가 생성한 Message를 저장할 변수 이름을 지정한다. 이 변수 이름은 JSP의 page, request, session 기본 객체등에

저장할때 사용되는 속성 이름으로 사용된다. scope 속성은 Message를 저장할 범위를 지정한다. 지정 가능한 범위는 page, request, session, application 이며,

기본 값은 page 이다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

3.2 SPRING이 제공하는 폼 관련 커스텀 태그

: SPRING의 장점 중 하나는 입력 폼 값을 커맨드 객체에 저장하는 기능을 제공한다는 것이다. SPRING은 또한 반대로 커맨드 객체의 값을 입력 폼에 출력해

주는 JSP 커스텀 태그를 제공하고 있어, 좀 더 쉽게 폼 관련 태그를 생성할 수 있도록 도와 준다. 이들 태그를 사용하려면 먼저 다음과 같이 커스텀 태그를

설정해 주어야 한다.

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

SPRING은 <form> 태그부터 <textarea> 태그에 이르기까지 커맨드 객체를 손쉽게 연동할 수 있도록 도와주는 커스텀 태그를 제공하고 있는데, 이들 태그에

대해서 차례대로 살펴보도록 하자.

(1) <form> 태그를 위한 커스텀 태그 : <form:form>

: <form:form> 커스텀 태그는 <form> 태그를 생성할 때 사용된다. <form:form> 커스텀 태그를 사용하는 가장 간단한 방법은 다음과 같다.

<form:form commandName="login">

<form:errors />

<p>

<label for="loginType"><spring:message code="login.form.type" /></label>

<form:select path="loginType" items="${loginTypes}" />

</p>

...

<p>

<input type="submit" value="<spring:message code="login.form.submit" />">

</p>

</form:form>

<form:form> 커스텀 태그는 method 속성과 action 속성을 표시하지 않으면 method 속성의 값은 "post"로 설정되고 action 속성의 값은

현재 요청 URL의 값이 설정된다.

<form id="command" action="/chap07/login/login.do" method="post">

...

</form>

생성된 <form> 태그의 id 속성은 입력 폼의 값을 지정하는 커맨드 객체의 이름이 할당된다.

만약, 커맨드 객체의 이름이 기본 값인 "command"가 아니라면 다음과 같이 commandName 속성에 커맨드 객체의 이름을 명시해 주어야 한다.

<form:form commandName="login">

...

</form:form>

<form:form> 커스텀 태그는 <form> 태그와 관련하여 다음의 속성들을 추가적으로 제공하고 있다.

: action : 폼데이터를 전송할 URL을 입력 (HTML <form> 태그 속성)

: enctype : 전송될 데이터의 인코딩 타입. HTML <form> 태그 속성과 동일

: method : 전송 방식. HTML <form> 태그 속성과 동일

<form:form> 태그의 몸체에는 <input> 태그나 <select> 태그와 같이 입력 폼을 출력하는데 필요한 HTML 태그를 입력할 수 있다. 이때, 입력한 값이

잘못되어 다시 값을 입력해야 하는 경우에는 다음과 같이 커맨드 객체의 값을 사용해서 이전에 입력한 값을 출력한다.

<form:form commandName="login">

...

<input type="text" name="id" value="${login.id}"/>

...

</form:form>

이렇게 커맨드 객체의 값을 다시 입력 폼에 출력해 주어야 하는 경우에는 <form:input>이나 <form:checkbox>와 같이 SPRING이 제공하는 커스텀 태그를

사용하면 좀 더 쉽게 커맨드 객체의 값을 입력 폼에 설정할 수 있다. SPRING은 입력 폼과 관련하여 다양한 커스텀 태그를 제공하고 있는데, 이들 커스텀 태그에

대해서 차례대로 살펴보도록 하자.

(2) <input> 태그를 위한 커스텀 태그 : <form:input>, <form:password>, <form:hidden>

: SPRING은 <input> 태그와 관련하여 아래와 같은 커스텀 태그를 제공하고 있다.

커스텀 태그 설 명

==================================================

<form:input> text 타입의 <input> 태그

==================================================

<form:password> password 타입의 <input> 태그

==================================================

<form:hidden> hidden 타입의 <input> 태그

==================================================

<form:input> 커스텀 태그는 다음과 같이 path 속성을 사용해서 바인딩 될 커맨드 객체의 프로퍼티를 지정한다.

<form:form commandName="memberInfo">

<p>

<form:label path="userId">회원 ID</form:label>

<form:input path="userId"/>

<form:errors path="userId"/>

</p>

위 Code가 생성하는 HTML <input> 태그는 아래와 같다. 이때 id 속성과 name 속성의 값은 프로퍼티의 이름을 이용하여 생성하며, value 속성에는

<form:input> 커스텀 태그의 path 속성에서 지정한 커맨드 객체의 프로퍼티 값이 출력된다.

<form id="memberInfo" action="/chap07/member/regist.di" method="post">

<p>

<label for="userId"> 회원 ID </label>

<input id="userId" name="userId" type="text" value=""/>

</p>

<form:password> 커스텀 태그는 password 타입의 <input> 태그를 생성하고, <form:hidden> 커스텀 태그는 hidden 타입의 <input> 태그를 생성한다.

두 태그 모두 path 속성을 사용하여 바인딩 할 커맨드 객체의 프로퍼티를 지정한다.

<form:form commandName="login">

<form:hidden path="defaultSecurityLevel"/>

...

<form:password path="password"/>

</form:form>

(3) <select> 태그를 위한 커스텀 태그 : <form:select>, <form:options>, <form:option>

: <select> 태그와 관련된 커스텀 태그는 아래와 같이 세 가지가 존재한다.

커스텀 태그 설 명

====================================================================

<form:select> <select> 태그를 생성한다. <option> 태그를 생성하는 데 필요한 컬렉션을 전달받을

수도 있다.

====================================================================

<form:options> 지정한 컬렉션 객체를 이용하여 <option> 태그를 생성한다.

====================================================================

<form:option> 한 개의 <option> 태그를 생성한다.

====================================================================

<select> 태그는 선택 옵션을 제공할 때 주로 사용된다. 예를 들어, <select> 태그를 이용해서 직업 선택을 위한 옵션을 제공한다고 하자.

이런 옵션 정보는 컨트롤러에서 생성해서 뷰에 전달하는 경우가 많다. 보통 @ModelAttribute Annotation을 이용해서 <select> 태그에서

사용될 옵션 목록을 전달한다.

@ModelAttribute("loginTypes")

protected List<String> referenceData() throws Exception{

List<String> loginTypes = new ArrayList<String>();

loginTypes.add("일반회원");

loginTypes.add("기업회원");

loginTypes.add("헤드헌터회원");

return loginTypes;

}

이 경우 <form:select> 커스텀 태그를 사용하면 손쉽게 뷰에 전달한 객체를 이용하여 <select>와 <option> 태그를 생성할 수 있다.

아래 Code는 <form:select> 커스텀 태그를 이용하여 <select> 태그를 생성하는 예를 보여주고 있다. 이때 path 속성은 바인딩 될 커맨드 객체의

이름을 입력하며, items 속성에는 <option> 태그를 생성할 때 사용될 컬렉션 객체를 지정한다.

<form:form commandName="login">

<form:errors />

<p>

<label for="loginType"><spring:message code="login.form.type" /></label>

<form:select path="loginType" items="${loginTypes}" />

</p>

...

</form:form>

위의 <form:select> 커스텀 태그는 다음과 같은 HTML 태그를 생성한다. (실제로는 한 줄로 생성되는데, 가독성을 위해 포맷 일부를 변경했다.)

<select id="loginType" name="loginType">

<option value="일반회원">일반회원</option>

<option value="기업회원">기업회원</option>

<option value="헤드헌터회원">헤드헌터회원</option>

</select>

생성된 Code를 보면 컬렉션 객체의 값을 이용하여 <option> 태그의 value 속성과 텍스트를 설정한 것을 알 수 있다.

<form:options> 태그를 사용해도 동일한 작업을 수행할 수 있다. <form:options> 커스텀 태그를 사용할 경우, 다음과 같이 <form:select> 커스텀 태그에

<form:options> 커스텀 태그를 중첩하며, items 속성을 <form:options> 커스텀 태그에서 설정한다.

<form:select path="loginType">

<option value="">-----선택하세요 -----</option>

<form:options items="${loginTypes}"/>

</form:select>

<form:options> 커스텀 태그는 주로 위 Code와 같이 컬렉션에 포함되어 있지 않은 값을 갖는 <option> 태그를 추가할 때 사용된다.

<form:option> 커스텀 태그는 <option> 태그를 직접 지정할 때 사용된다. 아래 Code는 <form:option> 커스텀 태그의 사용 예이다.

<form:select path="loginType">

<form:option value="일반회원"/>

<form:option value="기업회원"/>기업</form:option>

<form:option value="헤드헌터회원" label="헤드헌터"/>

<form:select>

<form:option> 커스텀 태그의 value 속성을 사용하여 <option> 태그의 value 속성 값을 지정한다. <form:option> 커스텀 태그의 몸체 내용을 입력하지 않으면

value 속성에 지정한 값이 텍스트로 사용되고, 몸체 내용을 입력하면 몸체 내용이 텍스트로 사용된다. label 속성을 사용할 경우에는 label 속성에 명시한 값이

텍스트로 사용된다. 위 Code가 생성한 HTML Code는 다음과 같다.

<select id="loginType" name="loginType">

<option value="일반회원">일반회원</option>

<option value="기업회원">기업</option>

<option value="헤드헌터회원">헤드헌터</option>

</select>

<option> 태그를 생성하는 데 사용되는 컬렉션 객체가 String이 아닐 수도 있다. 예를 들어, <option>을 생성하는 데 사용되는 컬렉션에 다음과 같은

Code Class의 객체가 저장된다고 하자.

public class Code{

private String code;

private String label;

.. //get & set 메서드

}

이 경우, Code 객체의 code 프로퍼티와 label 프로퍼티를 각각 <option> 태그의 value 속성과 텍스트로 사용하고 싶을 것이다. 이렇게 컬렉션에 저장된 객체 자체가

아닌 객체의 특정 프로퍼티를 사용해야 하는 경우에는 다음과 같이 itemValue 속성과 itemLabel 속성을 사용해서 컬렉션에 저장된 객체에서 <option> 태그를 생성하는데

사용될 프로퍼티를 지정할 수 있다.

<form:select path="jobCode">

<option value="">-----선택하세요-----</option>

<form:options items="${jobCodes}" itemLabel="label" itemValues="code"/>

</form:select>

위 Code는 jobCodes 컬렉션에 저장된 객체를 이용하여 <option> 태그를 생성한다. 이때 객체의 code 프로퍼티 값을 <option> 태그의 value 속성 값으로 사용하고,

객체의 label 프로퍼티의 값을 <option> 태그의 텍스트로 사용한다. <form:select> 커스텀 태그도 <form:options> 커스텀 태그와 마찬가지로 itemLabel 속성과

itemValue 속성을 사용할 수 있다.

SPRING이 제공하는 <form:select>, <form:options>, <form:option> 커스텀 태그의 장점은 커맨드 객체의 프로퍼티 값과 일치하는 값을 갖는 <option>을

자동으로 선택해 준다는 점이다. 예를 들어, 커맨드 객체의 loginType 프로퍼티의 값이 "기업회원"인 경우 다음과 같이 해당하는 <option> 태그에 selected

속성이 추가된다.

<select id="loginType" name="loginType">

<option value="일반회원">일반회원</toption>

<option value="기업회원" selected="selected">기업회원</toption>

<option value="헤드헌터회원">헤드헌터회원</toption>

</select>

(4) checkbox 타입 <input> 태그를 위한 커스텀 태그 : <form:checkboxes>, <form:checkbox>

: 한 개 이상의 값을 커맨드 객체의 특정 프로퍼티에 저장하고 싶은 경우, 배열이나 List와 같은 컬렉션 타입을 사용해서 값을 저장한다.

public class MemberInfo{

...

private String[] favorites;

public String[] getFavorites(){

return favorites;

}

public void setFavorites(String[] favorites){

this.favorites = favorites;

}

...

}

HTML 입력 폼에서는 다음과 같이 checkbox 타입의 <input> 태그를 이용해서 한 개 이상의 값을 선택할 수 있도록 하게 된다.

<input type="checkbox" name="favorite" value="윈도우2000">윈도우2000</input>

<input type="checkbox" name="favorite" value="윈도우XP">윈도우XP</input>

SPRING은 checkbox 타입의 <input> 태그와 관련하여 다음과 같은 커스텀 태그를 제공하고 있다.

커스텀 태그 설 명

=======================================================================================

<form:checkboxes> 커맨드 객체의 특정 프로퍼티와 관련된 checkbox 타입의 <input> 태그 목록을 생성한다.

=======================================================================================

<form:checkbox> 커맨드 객체의 특정 프로퍼티와 관련된 한 개의 checkbox 타입 <input> 태그를 생성한다.

=======================================================================================

<form:checkboxes> 커스텀 태그는 items 속성을 이용하여 값으로 사용할 컬렉션을 전달받고, path 속성을 이용하여 값을 바인딩 할 커맨드 객체의

프로퍼티를 지정한다. 아래 Code는 <form:checkboxes> 커스텀 태그의 사용 예이다.

<p>

<form:label path="favorites">선호 OS</form:label>

<form:checkboxes items="${favoritesOsNames}" path="favorites"/>

<form:errors path="favorites" />

</p>

위 Code에서 <form:checkboxes> 커스텀 태그는 다음과 같은 HTML Code를 생성한다. (아래 Code는 실제로 모두 한 줄로 출력된다.)

<span><input id="favorites1" name="favorites" type="checkbox" value="윈도우XP"/><label for="favorites1">윈도우XP</label></span>

<span><input id="favorites3" name="favorites" type="checkbox" value="윈도우7"/><label for="favorites3">윈도우7</label></span>

...

<input type="hidden" name="\_favorites" value="on"/>

각 <input> 태그의 value 속성에 사용된 값이 체크박스를 위한 텍스트로 사용되고 있다. 앞서 <option> 태그와 마찬가지로 컬렉션에 저장된 객체가

String이 아니라면, 다음과 같이 itemValue 속성과 itemLabel 속성을 이용해서 값과 텍스트를 설정 할 때 사용될 객체의 프로퍼티를 설정할 수 있다.

<p>

<form:label path="favorites">선호 OS</form:label>

<form:checkboxes items="${favoritesOsNames}" path="favorites" itemValue="code" itemLabel="label" />

<form:errors path="favorites" />

</p>

<form:checkbox> 커스텀 태그는 앞서 <form:option> 커스텀 태그와 같이 한 개의 checkbox 타입의 <input> 태그를 생성할 때 사용된다.

<form:checkbox> 커스텀 태그는 value 속성과 label 속성을 사용해서 값과 텍스트를 설정한다.

<form:checkbox path="favorites" value="WIN2000" label="윈도우즈2000"/>

<form:checkbox path="favorites" value="WINXP" label="윈도우즈XP"/>

<form:checkbox> 커스텀 태그는 바인딩 되는 값의 타입에 따라서 처리방식이 달라진다. 첫 번째로 바인딩 되는 값의 타입이 boolean 기본 데이터 타입이나

Boolean 래퍼 타입이라고 하자.

public class MemberInfo{

...

private boolean contractAgreement;

public boolean isContractAgreement(){

return contractAgreement;

}

public void setContractAgreement(boolean contractAgreement){

this.contractAgreement = contractAgreement;

}

...

}

이 경우 <form:checkbox>는 바인딩 되는 값이 true인 경우 "checked" 속성을 설정하며, false인 경우 "checked" 속성을 설정하지 않는다. 또한, 생성되는

<input> 태그의 value 속성의 값은 "true"가 된다. 아래 Code는 사용 예이다.

<form:checkbox path="contractAgreement" label="약관에 동의합니다."/>

위 Code가 생성하는 HTML Code는 다음과 같다. (실제로 모두 한 줄로 생성된다.)

<input id="contractAgreement1" name="contractAgreement" type="checkbox" value="true"/>

<label for="contractAgreement1"> 약관에 동의합니다. </label>

<input type="hidden" name="\_contractAgreement" value="on"/>

<form:checkbox> 태그의 두 번째 처리 방식은 바인딩 되는 값의 타입이 배열이나 Collection인 경우이다. 값의 타입이 배열이나 Collection인 경우,

해당 컬렉션에 값이 포함되어 있는 경우에 "checked" 속성을 설정한다. 예를 들어, 아래와 같은 배열 타입의 프로퍼티가 있다고 해 보자.

public class MemberInfo{

private String[] favorites;

...

public String[] getFavorites(){

return favorites;

}

public void setFavorites(String[] favorites){

this.favorites=favorites;

}

...

}

<form:checkbox> 커스텀 태그를 사용하면 다음과 같이 favorites 프로퍼티에 대한 폼을 처리할 수 있다.

<form:checkbox path="favorites" value="WINXP" label="윈도우즈XP"/>

<form:checkbox path="favorites" value="비스타" label="비스타"/>

<form:checkbox path="favorites" value="윈도우7" label="윈도우7"/>

<form:checkbox path="favorites" value="우분투" label="우분투"/>

<form:checkbox path="favorites" value="맥" label="맥"/>

<form:checkbox> 태그의 세 번째 처리 방식은 임의 타입의 프로퍼티와 바인딩 되는 경우이다. 이 경우, <form:checkbox> 태그는 value 속성의 값과

프로퍼티의 값이 일치하는 경우, "checked" 속성의 값을 설정한다.

(5) radio 타입 <input> 태그를 위한 커스텀 태그 : <form:radiobuttons>, <form:radiobutton>

: 여러 가지 옵션 중에서 한 가지를 선택해야 하는 경우, radio 타입의 <input> 태그를 사용한다. SPRING은 radio 타입의 <input> 태그와 관련하여

아래와 같은 커스텀 태그를 제공하고 있다.

커스텀 태그 설 명

=============================================================================

<form:radiobuttons> 커맨드 객체의 특정 프로퍼티와 관련된 radio 타입의 <input> 태그 목록을 생성한다.

=============================================================================

<form:radiobutton> 커맨드 객체의 특정 프로퍼티와 관련된 한 개의 radio 타입 <input> 태그를 제공하고 있다.

=============================================================================

<form:radiobuttons> 커스텀 태그는 다음과 같이 items 속성을 이용하여 값으로 사용할 컬렉션을 전달받고, path 속성을 이용하여 값을 바인딩 할

커맨드 객체의 프로퍼티를 지정한다.

<p>

<form:label path="tool">주로 사용하는 개발 툴</form:label>

<form:radiobuttons items="${tools}" path="tool"/>

</p>

<form:radiobuttons> 커스텀 태그는 다음과 같은 HTML 태그를 생성한다.

<span><input id="tool1" name="tool" type="radio" value="Eclipse"/><label for="tool1">Eclipse</label></span>

<span><input id="tool2" name="tool" type="radio" value="IntelliJ"/><label for="tool1">IntelliJ</label></span>

<span><input id="tool3" name="tool" type="radio" value="NetBeans"/><label for="tool1">NetBeans</label></span>

<form:radiobuttons> 커스텀 태그는 1개의 radio 타입 <input> 태그를 생성할 때 사용되며, value 속성과 label 속성을 이용하여 값과 텍스트를 설정한다.

(6) <textarea> 태그를 위한 커스텀 태그 : <form:textarea>

: 게시글 내용과 같이 여러 줄을 입력 받아야 하는 경우 <textarea> 태그를 사용한다. SPRING은 <form:textarea> 커스텀 태그를 제공하고 있으며, 이 태그를

이용하면 커맨드 객체와 관련된 <textarea> 태그를 생성할 수 있다. 아래 Code는 사용 예이다.

<p>

<form:label path="etc">기타</form:label>

<form:textarea path="etc" cols="20" rows="3"/>

</p>

<form:textarea> 커스텀 태그가 생성하는 HTML 태그는 다음과 같다.

<p>

<label for="etc">기타</label>

<textarea id="etc" name="etc" rows="3" cols="20"></textarea>

</p>

(7) CSS 및 HTML 태그와 관련된 공통 속성

: <form:input>, <form:select> 등 SPRING이 입력 폼과 관련해서 제공하는 커스텀 태그는 HTML의 CSS 및 이벤트 관련 속성을 제공하고 있다. 먼저 CSS와 관련된

속성은 다음과 같다.

: cssClass : HTML의 class 속성 값

: cssErrorClass : 폼 검증 에러가 발생했을 때 사용할 HTML의 class 속성 값.

: cssStyle : HTML의 style 속성 값.

HTML 태그가 사용하는 속성 중 다음의 속성들도 사용 가능하다.

: id, titlem dir

: disable, tabindex

: onfocus, onblur, onchange

: onclick, ondbclick

: onkeydown, onkeypress, onkeyup

: onmousedown, onmousemove, onmouseup

: onmouseout, onmouseover

또한, 각 커그텀 태그는 htmlEscape 속성을 사용해서 커맨드 객체의 값에 포함된 HTML 특수 문자를 엔티티 Reference로 변환할 지의 여부를 결정할 수 있다.

htmlEscape 속성을 지정하지 않을 경우 defaultHtmlEscape 컨텍스트 파라미터에서 설정한 값을 사용한다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

3.3 SPRING이 제공하는 에러 관련 커스텀 태그

: 6장에서 Validator를 이용해서 커맨드 객체의 값을 검사하는 방법을 살펴보았다. Validator는 아래 Code와 같이 Errors를 이용하여 에러 정보를 저장하였다.

public class MemberInfoValidator implements Validator{

@Override

public void validate(Object target, Errors errors){

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "userId", "required");

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "required");

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "address.address1", "required");

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "address.address2", "required");

...

}

...

}

Errors 나 BindingResult를 이용해서 에러 정보를 추가한 경우, <form:errors> 커스텀 태그를 이용해서 에러 Message를 출력할 수 있다. <form:errors> 커스텀 태그는

path 속성을 이용해서 커맨드 객체의 특정 프로퍼티와 관련된 에러Message를 출력할 수 있다.

<form:form commandName="memberInfo">

<p>

<form:label path="userId">회원 ID</form:label>

<form:input path="userId"/>

<form:errors path="userId"/>

</p>

</form:form>

위 Code의 경우 "userId" 프로퍼티와 관련된 모든 에러 Message를 출력한다. 에러 Message는 앞서 6장에서 살펴봤듯이 에러 Code와 필드 이름, 그리고 커맨드 Class 이름을

통해서 결정된다. 에러 Code와 Message Code 사이의 Mapping에 대한 정보는 6장의 "DefaultMessageCodesResolver와 에러 Message"절을 참고하지 바란다.

<form:errors> 커스텀 태그는 지정한 프로퍼티와 관련된 한 개 이상의 에러 Message를 출력하게 된다. 각 에러 Message를 생성할 때 다음과 같은 두 개의 속성이 사용된다.

: element : 각 에러Message를 출력할 때 사용될 HTML 태그. 기본 값은 span이다.

: delimeter : 각 에러 Message를 구분할 때 사용될 HTML 태그. 기본 값은 <br/>이다.

아래 Code는 위 두 속성의 사용 예를 보여주고 있다.

<form:errors path="userId" element="div" delimeter=""/>

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

3.4 <spring:htmlEscape> 커스텀 태그와 htmlEscape 속성

: defaulthtmlEscape 컨텍스트 파라미터를 사용해서 WEB Application 전반에 걸쳐서 HTML의 특수 문자를 엔티티 Reference로 치환할 지의 여부를 결정하는데, 만약,

각 JSP 페이지 별로 특수 문자 치환 여부를 설정해 주고 싶다면 다음과 같이 <spring:htmlEscape> 커스텀 태그를 사용하면 된다.

< %-- JSP 페이지의 앞 부분에서 설정 --%>

<spring:htmlEscape defaultHtmlEscape="true"/>

...

<springmessage.../>

<form:input.../>

<spring:htmlEscape> 커스텀 태그를 설정하면, 이후로 실행되는 <spring:message> 커스텀 태그나 <form:input> 커스텀 태그와 같이 SPRING이 제공하는 커스텀 태그는

<spring:htmlEscape> 커스텀 태그의 defaultHtmlEscape 속성에서 지정한 값을 기본 값으로 사용한다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

3.5 <form:form>의 RESTful 지원

: SPRING MVC는 HTTP의 GET, POST, PUT, DELETE 방식을 지원하고 있으며, 다음과 같이 컨트롤러 메서드에서 어떤 HTTP 방식을 지원할 지 선택할 수 있다.

@Controller

public class ArticleController {

@RequestMapping(value="/article/{id}", method=RequestMethod.GET)

public String read(@PathVariable("id") Integer id, Model model) {

...

return "article/read";

}

@RequestMapping(value="/article/{id}", method=RequestMethod.DELETE)

public String delete(@PathVariable("id") Integer id, Model model) {

...

return "article/delete";

}

@RequestMapping(value="/article/{id}", method=RequestMethod.PUT)

public String modify(@PathVariable("id") Integer id, Model model) {

model.addAttribute("article", new Article(id));

return "article/modify";

}

@RequestMapping(value="/article", method=RequestMethod.POST)

public String write(Model model) {

...

return "article/write";

}

@RequestMapping("/articleForm.do")

public String testForm(Model model) {

...

return "article/testForm";

}

}

그런데, 대부분의 WEB 브라우저는 GET 방식과 POST 방식만을 지원하고 있어서 DELETE방식이나 PUT방식의 요청을 전송할 수 없기 때문에,

WEB 브라우저를 이용하는 경우 GET방식과 POST방식으로만 처리 해야 했다.

SPRING3 버전부터는 PUT과 DELETE 방식을 이용해서 컨트롤러를 구현하면서 WEB 브라우저에서도 그대로 해당 컨트롤러를 사용할 수 있도록 지원해주고 있다.

이 기능을 이용하려면 다음의 두 가지 작업만 해 주면 된다.

: web.xml 파일에 HiddenMethodFilter 적용

: <form:form> 태그의 method 속성에 put 또는 delete 이용

먼저, web.xml 파일에 HiddenHttpMethodFilter를 설정한다. HiddenHttpMethodFilter의 필터 Mapping 대상으로는 아래 Code와 같이 DispatcherServlet을 지정한다.

<web-app>

...

<filter>

<filter-name>httpMethodFilter</filter-name>

<filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>httpMethodFilter</filter-name>

<servlet-name>dispatcherInternal</servlet-name>

<servlet-name>dispatcherTiles2</servlet-name>

</filter-mapping>

...

</web-app>

이제 <form:form> 태그의 method 속성 값으로 delete나 put을 지정한다.

<form:form method="delete">

...

</form:form>

<form:form> 태그의 method 값이 delete나 put인 경우 <form:form> 태그는 다음과 같이 hidden 타입의 <input> 태그를 추가로 생성한다.

<form id="article" action"/chap07/jsp/article/1" method="post">

<input type="hidden" name="\_method" value="delete"/>

...

<input type="submit" value="삭제"/>

</form>

hiddenHttpMethodFilter는 요청 파라미터에 \_method 파라미터가 존재할 경우, \_method 파라미터에 명시된 요청 방식을 이용하도록 SPRING MVC의 관련 정보를

설정한다. DispatcherServlet은 이 정보를 이용해서 컨트롤러의 알맞은 메서드를 찾기때문에, WEB 브라우저를 이용하더라도 RESTful 방식으로 구현된 컨트롤러를

사용할 수 있게 된다.

<Example001> /jsp/login/login.do

[web.xml]

...

<context-param>

<param-name>defaultHtmlEscape</param-name>

<param-value>false</param-value>

</context-param>

<servlet>

<servlet-name>dispatcherInternal</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcherInternal</servlet-name>

<url-pattern>/jsp/\*</url-pattern>

</servlet-mapping>

<filter>

<filter-name>encodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>EUC-KR</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

...

[LoginController.java]

package madvirus.spring.chap07.controller;

import java.util.ArrayList;

import java.util.List;

import org.springframework.stereotype.Controller;

import org.springframework.validation.BindingResult;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

@Controller

@RequestMapping("/login/login.do")

public class LoginController {

private Authenticator authenticator;

@ModelAttribute("login")

public LoginCommand formBacking() {

return new LoginCommand();

}

@RequestMapping(method = RequestMethod.GET)

public String form() {

return "loginForm";

}

@RequestMapping(method = RequestMethod.POST)

public String submit(@ModelAttribute("login") LoginCommand loginCommand,

BindingResult result) {

new LoginCommandValidator().validate(loginCommand, result);

if (result.hasErrors()) {

return "loginForm";

}

try {

authenticator.authenticate(loginCommand.getId(), loginCommand

.getPassword());

return "loginSuccess";

} catch (AuthenticationException ex) {

result.reject("invalidIdOrPassword", new Object[] { loginCommand

.getId() }, null);

return "loginForm";

}

}

@ModelAttribute("loginTypes")

protected List<String> referenceData() throws Exception {

List<String> loginTypes = new ArrayList<String>();

loginTypes.add("일반회원");

loginTypes.add("기업회원");

loginTypes.add("헤드헌터회원");

return loginTypes;

}

public void setAuthenticator(Authenticator authenticator) {

this.authenticator = authenticator;

}

}

1[Authenticator.java]

package madvirus.spring.chap07.controller;

public interface Authenticator {

void authenticate(String id, String password);

}

3[MockAuthenticator.java]

package madvirus.spring.chap07.controller;

public class MockAuthenticator implements Authenticator {

@Override

public void authenticate(String id, String password) {

if (!id.equals("madvirus")) {

throw new AuthenticationException("invalid id "+id);

}

}

}

2[AuthenticationException.java]

package madvirus.spring.chap07.controller;

public class AuthenticationException extends RuntimeException {

private static final long serialVersionUID = 1L;

public AuthenticationException(String message) {

super(message);

}

}

4[LoginCommand.java]

package madvirus.spring.chap07.controller;

public class LoginCommand {

private String id;

private String password;

private String loginType;

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getLoginType() {

return loginType;

}

public void setLoginType(String loginType) {

this.loginType = loginType;

}

}

5[LoginCommandValidator.java]

package madvirus.spring.chap07.controller;

import org.springframework.validation.Errors;

import org.springframework.validation.ValidationUtils;

import org.springframework.validation.Validator;

public class LoginCommandValidator implements Validator {

@Override

public void validate(Object target, Errors errors) {

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "id", "required");

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "password",

"required");

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "loginType",

"required");

}

@Override

public boolean supports(Class<?> clazz) {

return LoginCommand.class.isAssignableFrom(clazz);

}

}

6[/src/messages/validation\_en.properties]

required=required

required.login.id=login id is required

required.login.password=login password is requreid

required.login.loginType=You have to select login type

invalidIdOrPassword.login=Login id and password do not match. (You provided {0})

duplicate.homepageUrl=Aleady registered URL.

invalidValue=invalid value

not\_supported=not supported type

must\_select=You have to choose at least one.

duplicate=Aleady used value.

7[/src/messages/validation.properties]

required=필수 항목입니다.

required.login.id=로그인 ID는 필수 항목입니다.

required.login.password=로그인 암호는 필수 항목입니다.

required.login.loginType=로그인 타입은 반드시 선택하셔야 합니다.

invalidIdOrPassword.login=잘못된 ID나 암호를 입력하셨습니다. 입력한 ID는 {0}입니다.

duplicate.homepageUrl=이미 등록된 주소입니다.

invalidValue=올바른 값을 입력하세요.

not\_supported=지원하지 않는 타입니다.

must\_select=1개 이상 선택하셔야 합니다.

duplicate=중복된 값입니다.

8[/src/messages/label\_en.properties]

label.currentTime=Current Time

title.currentTime=Time

login.form.title=Login Form

login.form.type=Login Type

login.form.id=ID

login.form.password=Password

login.form.submit=Login

greeting=I'm {0}. {1}

9[/src/messages/label.properties]

label.currentTime=현재 시간

title.currentTime=시간

login.form.title=로그인 입력 폼

login.form.type=로그인 타입

login.form.id=로그인 ID

login.form.password=로그인 암호

login.form.submit=로그인

greeting=전 {0} 입니다. "{1}"

[/WEB-INF/viewjsp/loginForm.jsp]

<%@ page contentType="text/html; charset=EUC-KR" %>

<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title><spring:message code="login.form.title"/></title>

</head>

<body>

<form:form commandName="login">

<form:errors />

<p>

<label for="loginType"><spring:message code="login.form.type" /></label>

<form:select path="loginType" items="${loginTypes}" />

</p>

<p>

<label for="id"><spring:message code="login.form.id" /></label>

<form:input id="id" path="id"/>

<form:errors path="id" />

</p>

<p>

<label for="password"><spring:message code="login.form.password" /></label>

<form:password id="password" path="password"/>

<form:errors path="password" />

</p>

<p>

<input type="submit" value="<spring:message code="login.form.submit" />">

</p>

</form:form>

</body>

</html>

[/WEB-INF/viewjsp/loginSuccess.jsp]

<%@ page language="java" contentType="text/html; charset=EUC-KR" %>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>로그인 성공</title>

</head>

<body>

로그인에 성공했습니다.

</body>

</html>

[/WEB-INF/dispatcherInternal-servlet.xml]

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context" xmlns:p="http://www.springframework.org/schema/p"

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-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<bean class="madvirus.spring.chap07.controller.LoginController">

<property name="authenticator">

<bean class="madvirus.spring.chap07.controller.MockAuthenticator" />

</property>

</bean>

<bean id="viewResolver"

class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/viewjsp/" />

<property name="suffix" value=".jsp" />

</bean>

<bean id="messageSource"

class="org.springframework.context.support.ResourceBundleMessageSource">

<property name="basenames">

<list>

<value>messages.validation</value>

<value>messages.label</value>

</list>

</property>

</bean>

</beans>

<Example002> /jsp/member/regist.do

[RegistMemberController.java]

package madvirus.spring.chap07.controller;

import java.util.ArrayList;

import java.util.List;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.validation.BindingResult;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

@Controller

@RequestMapping("/member/regist.do")

public class RegistMemberController {

private String formViewName = "registMemberForm";

@RequestMapping(method = RequestMethod.GET)

public String form(Model model) {

referenceData(model);

return formViewName;

}

private void referenceData(Model model) {

List<Code> jobCodes = new ArrayList<Code>();

jobCodes.add(new Code("1", "개발자"));

jobCodes.add(new Code("2", "UI 개발자"));

jobCodes.add(new Code("3", "웹 디자이너"));

jobCodes.add(new Code("3", "기획자"));

String[] favoritesOsNames = { "윈도우XP", "비스타", "윈도우7", "우분투", "맥" };

String[] tools = { "Eclipse", "IntelliJ", "NetBeans" };

model.addAttribute("jobCodes", jobCodes);

model.addAttribute("favoritesOsNames", favoritesOsNames);

model.addAttribute("tools", tools);

}

@ModelAttribute

protected Object formBackingObject() throws Exception {

return new MemberInfo();

// MemberInfo member = new MemberInfo();

// member.setAddress(new Address());

// return member;

}

@RequestMapping(method = RequestMethod.POST)

public String submit(@ModelAttribute MemberInfo memberInfo,

BindingResult result, Model model) {

new MemberInfoValidator().validate(memberInfo, result);

checkDuplicateId(memberInfo.getUserId(), result);

if (result.hasErrors()) {

referenceData(model);

return formViewName;

}

return "registMember";

}

private void checkDuplicateId(String userId, BindingResult errors) {

if (userId.equals("madvirus")) {

errors.rejectValue("userId", "duplicate");

}

}

}

[Code.java]

package madvirus.spring.chap07.controller;

public class Code {

private String code;

private String label;

public Code() {

}

public Code(String code, String value) {

this.code = code;

this.label = value;

}

public String getCode() {

return code;

}

public void setCode(String code) {

this.code = code;

}

public String getLabel() {

return label;

}

public void setLabel(String value) {

this.label = value;

}

}

[MemberInfo.java]

package madvirus.spring.chap07.controller;

public class MemberInfo {

private String userId;

private String name;

private Address address;

private String[] favorites;

private String jobCode;

private String tool;

private String etc;

private boolean contractAgreement;

public String getUserId() {

return userId;

}

public void setUserId(String userId) {

this.userId = userId;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Address getAddress() {

return address;

}

public void setAddress(Address address) {

this.address = address;

}

public String[] getFavorites() {

return favorites;

}

public void setFavorites(String[] favorates) {

this.favorites = favorates;

}

public String getJobCode() {

return jobCode;

}

public void setJobCode(String jobCode) {

this.jobCode = jobCode;

}

public String getTool() {

return tool;

}

public void setTool(String tool) {

this.tool = tool;

}

public String getEtc() {

return etc;

}

public void setEtc(String etc) {

this.etc = etc;

}

public boolean isContractAgreement() {

return contractAgreement;

}

public void setContractAgreement(boolean contractAgreement) {

this.contractAgreement = contractAgreement;

}

}

[Address.java]

package madvirus.spring.chap07.controller;

public class Address {

private String address1;

private String address2;

public String getAddress1() {

return address1;

}

public void setAddress1(String address1) {

this.address1 = address1;

}

public String getAddress2() {

return address2;

}

public void setAddress2(String address2) {

this.address2 = address2;

}

}

[MemberInfoValidator.java]

package madvirus.spring.chap07.controller;

import org.springframework.validation.Errors;

import org.springframework.validation.ValidationUtils;

import org.springframework.validation.Validator;

public class MemberInfoValidator implements Validator {

@Override

public void validate(Object target, Errors errors) {

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "userId", "required");

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "required");

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "address.address1",

"required");

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "address.address2",

"required");

ValidationUtils

.rejectIfEmptyOrWhitespace(errors, "jobCode", "required");

MemberInfo memberInfo = (MemberInfo) target;

if (memberInfo.getFavorites() == null

|| memberInfo.getFavorites().length == 0) {

errors.rejectValue("favorites", "must\_select");

}

}

@Override

public boolean supports(Class<?> clazz) {

return MemberInfo.class.isAssignableFrom(clazz);

}

}

[registMemberForm.jsp]

<%@ page contentType="text/html; charset=EUC-KR" %>

<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>회원 등록 폼</title>

</head>

<body>

<form:form commandName="memberInfo">

<p>

<form:label path="userId">회원 ID</form:label>

<form:input path="userId" />

<form:errors path="userId" />

</p>

<p>

<form:label path="name">이름</form:label>

<form:input path="name" />

<form:errors path="name" />

</p>

<p>

<form:label path="address.address1">주소1</form:label>

<form:input path="address.address1" />

<form:errors path="address.address1" />

</p>

<p>

<form:label path="address.address2">주소2</form:label>

<form:input path="address.address2" />

<form:errors path="address.address2" />

</p>

<p>

<form:label path="jobCode">직업</form:label>

<form:select path="jobCode" >

<option value="">--- 선택하세요 ---</option>

<form:options items="${jobCodes}" itemLabel="label" itemValue="code" />

</form:select>

<form:errors path="jobCode" />

</p>

<p>

<form:label path="favorites">선호 OS</form:label>

<form:checkboxes items="${favoritesOsNames}" path="favorites" />

<%--

<form:checkbox path="favorites" value="윈도우XP" label="윈도우XP" />

<form:checkbox path="favorites" value="비스타" label="비스타" />

<form:checkbox path="favorites" value="윈도우7" label="윈도우7" />

<form:checkbox path="favorites" value="우분투" label="우분투" />

<form:checkbox path="favorites" value="맥" label="맥" />

--%>

<form:errors path="favorites" />

</p>

<p>

<form:label path="tool">주로 사용하는 개발툴</form:label>

<form:radiobuttons items="${tools}" path="tool" />

</p>

<p>

<form:label path="etc">기타</form:label>

<form:textarea path="etc" cols="20" rows="3"/>

</p>

<p>

<form:checkbox path="contractAgreement" label="약관에 동의합니다."/>

</p>

<p>

<input type="submit" value="회원 등록">

</p>

</form:form>

</body>

</html>

[registMember.jsp]

<%@ page contentType="text/html; charset=EUC-KR" %>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>회원 등록</title>

</head>

<body>

회원 등록: ${memberInfo.name}, ${memberInfo.address.address1}, ${memberInfo.address.address2}

</body>

</html>

[dispatcherInternal-servlet.xml]

...

<bean class="madvirus.spring.chap07.controller.RegistMemberController" />

...

<Example003> /jsp/articleForm.do

[ArticleController.java]

package madvirus.spring.chap07.controller;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

@Controller

public class ArticleController {

@RequestMapping(value="/article/{id}", method=RequestMethod.GET)

public String read(@PathVariable("id") Integer id, Model model) {

model.addAttribute("article", new Article(id));

return "article/read";

}

@RequestMapping(value="/article/{id}", method=RequestMethod.DELETE)

public String delete(@PathVariable("id") Integer id, Model model) {

model.addAttribute("article", new Article(id));

return "article/delete";

}

@RequestMapping(value="/article/{id}", method=RequestMethod.PUT)

public String modify(@PathVariable("id") Integer id, Model model) {

model.addAttribute("article", new Article(id));

return "article/modify";

}

@RequestMapping(value="/article", method=RequestMethod.POST)

public String write(Model model) {

model.addAttribute("article", new Article(2));

return "article/write";

}

@RequestMapping("/articleForm.do")

public String testForm(Model model) {

model.addAttribute("article", new Article(1));

return "article/testForm";

}

}

[Article.java]

package madvirus.spring.chap07.controller;

public class Article {

private Integer id;

public Article(Integer id) {

this.id = id;

}

public Article() {

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

}

[/WEB-INF/viewjsp/article/testForm.jsp]

<%@ page contentType="text/html; charset=EUC-KR" %>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<%

String servletPath = request.getRequestURI();

%>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>테스트 코드</title>

</head>

<body>

<form:form action="/SpringView/jsp/article/${article.id}" commandName="article" method="delete">

<input type="submit" value="삭제">

</form:form>

<form:form action="/SpringView/jsp/article/${article.id}" commandName="article" method="put">

<input type="submit" value="수정">

</form:form>

<form:form action="/SpringView/jsp/article" commandName="article" method="post">

<input type="submit" value="생성">

</form:form>

</body>

</html>

[/WEB-INF/viewjsp/article/write.jsp]

<%@ page language="java" contentType="text/html; charset=EUC-KR" %>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>게시글 쓰기</title>

</head>

<body>

게시글 쓰기: ${article.id}

</body>

</html>

[/WEB-INF/viewjsp/article/read.jsp]

<%@ page language="java" contentType="text/html; charset=EUC-KR" %>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>게시글 읽기</title>

</head>

<body>

게시글 읽기: ${article.id}

</body>

</html>

[/WEB-INF/viewjsp/article/modify.jsp]

<%@ page language="java" contentType="text/html; charset=EUC-KR" %>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>게시글 수정</title>

</head>

<body>

게시글 수정: ${article.id}

</body>

</html>

[/WEB-INF/viewjsp/article/delete.jsp]

<%@ page language="java" contentType="text/html; charset=EUC-KR" %>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>게시글 삭제</title>

</head>

<body>

게시글 삭제: ${article.id}

</body>

</html>

[dispatcherInternal-servlet.xml]

...

<bean class="madvirus.spring.chap07.controller.ArticleController" />

...

<Example004> /jsp/changeLanguage?lang=ko, /jsp/changeLanguage?lang=en

[LocaleChangeController.java]

package madvirus.spring.chap07.controller;

import java.util.Locale;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.servlet.LocaleResolver;

@Controller

public class LocaleChangeController {

private LocaleResolver localeResolver;

@RequestMapping("/changeLanguage")

public String change(@RequestParam("lang") String language,

HttpServletRequest request, HttpServletResponse response) {

Locale locale = new Locale(language);

localeResolver.setLocale(request, response, locale);

return "redirect:/index.jsp";

}

public void setLocaleResolver(LocaleResolver localeResolver) {

this.localeResolver = localeResolver;

}

}

[dispatcherInternal-servlet.xml]

...

<bean class="madvirus.spring.chap07.controller.LocaleChangeController">

<property name="localeResolver" ref="localeResolver" />

</bean>

<bean id="localeResolver"

class="org.springframework.web.servlet.i18n.SessionLocaleResolver" />

...

<Example005> /jsp/login/login.do?language=ko, /jsp/login/login.do?language=en

[LocaleChangeController2.java]

package madvirus.spring.chap07.controller;

import java.util.Locale;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.servlet.LocaleResolver;

import org.springframework.web.servlet.support.RequestContextUtils;

@Controller

public class LocaleChangeController2 {

@RequestMapping("/changeLanguage2")

public String change(@RequestParam("lang") String language,

HttpServletRequest request, HttpServletResponse response) {

Locale locale = new Locale(language);

LocaleResolver localeResolver = RequestContextUtils

.getLocaleResolver(request);

localeResolver.setLocale(request, response, locale);

return "redirect:/index.jsp";

}

}

[dispatcherInternal-servlet.xml]

...

<bean class="madvirus.spring.chap07.controller.LocaleChangeController2" />

...

<bean id="localeChangeInterceptor"

class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"

p:paramName="language" />

...

<bean

class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">

<property name="interceptors">

<list>

<ref bean="localeChangeInterceptor" />

</list>

</property>

</bean>

...

===================================================================================================

4. Tiles 2 연동을 이용한 레이아웃 템플릿 처리

: 다수의 WEB 페이지는 동일한 페이지 구성을 갖고 있다. 아래와 같이 각각의 페이지가 상단과 하단은 동일한 내용을 출력하고 내용 부분만 각자 알맞은 내용을

출력하는 경우가 많다.

===================== =====================

헤더 헤더

===================== =====================

로그인 폼 게시글 내용

ID : 번호 | 작성자 | 제목

Password : 1 | 윤대명 | 글 제목

===================== =====================

푸터 푸터

===================== =====================

레이아웃이 동일하고 공통된 내용이 들어가는 영역이 많은 경우에는 Tiles와 같은 템플릿 Library를 사용해서 레이아웃을 처리하면 뷰 관련 Code에서 레이아웃을

처리하기 위한 Code의 중복을 제거할 수 있다는 장점이 있다. SPRING은 널리 사용되고 있는 템플릿 Library인 Tiles2 버전을 지원하고 있는데, 본 절에서는

SPRING MVC와 Tiles2를 연동하는 방법을 살펴보도록 하겠다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

4.1 TilesConfigurer를 이용한 Tiles 2 연동

: Tiles 2 연동을 위해서는 다음의 jar 파일을 Class 패스에 추가해 주어야 한다.(commons-logging.jar의 경우 spring core Module에서도 의존하는 Module이다.)

- tiles-api-2.1.x.jar, tiles-core-2.1.x.jar, tiles-jsp-2.1.x.jar

- commons-beanutils.jar, commons-digester.jar, commons-logging.jar

- SPRING webmvc Module

SPRING3에서 Tiles2를 연동하기 위해서는 SPRING 설정 파일에 다음의 두 가지 내용을 설정해야 한다.

- TilesConfigurer를 이용하여 Tiles 2 레이아웃 설정 파일 명시

- UrlBasedViewResolver의 viewClass 프로퍼티를 TilesView로 지정

아래 Code는 설정 예이다. TilesConfigurer Class와 TilesView Class는 모두 tiles2 패키지에 포함된 Class임에 유의하자.

<bean id="tilesConfigurer" class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">

<property name="definitions">

<list>

<value>/WEB-INF/tiles2def/tilesdef.xml</value>

</list>

</property>

</bean>

<bean id="viewResolver" class="org.springframework.web.servlet.view.UrlBasedViewResolver">

<property name="viewClass" value="org.springframework.web.servlet.view.tiles2.TilesView"/>

</bean>

TilesConfigurer 빈 객체는 definitions 속성을 이용해서 Tiles 설정 파일 목록을 전달받는다. Tiles2 설정 파일은 다음과 같이 작성된다.

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE tiles-definitions PUBLIC

"-//Apache Software Foundation//DTD Tiles Configuration 2.1//EN"

"http://tiles.apache.org/dtds/tiles-config\_2\_1.dtd">

<tiles-definitions>

<definition name="base\_layout"

template="/WEB-INF/viewtiles2/template/layout.jsp">

<put-attribute name="header"

value="/WEB-INF/viewtiles2/template/header.jsp" />

<put-attribute name="footer"

value="/WEB-INF/viewtiles2/template/footer.jsp" />

</definition>

<definition name="loginForm" extends="base\_layout">

<put-attribute name="title" value="로그인폼" />

<put-attribute name="body"

value="/WEB-INF/viewtiles2/loginForm.jsp" />

</definition>

<definition name="loginSuccess" extends="base\_layout">

<put-attribute name="title" value="로그인 성공" />

<put-attribute name="body"

value="/WEB-INF/viewtiles2/loginSuccess.jsp" />

</definition>

...

</tiles-definitions>

TilesView Class는 컨트러가 지정한 뷰 이름과 동일한 이름을 갖는 <definition> 태그를 사용하여 뷰를 생성한다. 예를 들어, 컨트롤러가 다음과 같이

뷰 이름을 "loginForm"으로 지정했다고 하자.

@Controller

@RequestMapping("/login/login.do")

public class LoginController{

@RequestMapping(method=RquestMethod.GET)

public String form(){

return "loginForm"

}

...

}

이 경우 name 속성의 값이 "loginForm"인 <definition> 태그의 정보를 사용한다. 즉, 앞서 Tiles 설정 내용에서 봤듯이 layout.jsp를 레이아웃 템플릿으로

사용하고, 헤더와 푸터에 각각 header.jsp와 footer.jsp를 삽입하고, 그리고 내용에는 loginForm.jsp를 사용하게 된다.

레이아웃 템플릿 파일은 다음과 같이 Tiles2가 제공하는 커스텀 태그를 이용하여 레이아웃 정보를 설정한다.

<%@ page contentType="text/html; charset=EUC-KR"%>

<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles"%>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title><tiles:getAsString name="title" /></title>

</head>

<body>

<tiles:insertAttribute name="header" />

<hr />

<tiles:insertAttribute name="body" />

<hr/>

<tiles:insertAttribute name="footer" />

</body>

</html>

<Example005> /tiles2/login/login.do

[/WEB-INF/tiles2def/tilesdef.xml]

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE tiles-definitions PUBLIC

"-//Apache Software Foundation//DTD Tiles Configuration 2.1//EN"

"http://tiles.apache.org/dtds/tiles-config\_2\_1.dtd">

<tiles-definitions>

<definition name="base\_layout"

template="/WEB-INF/viewtiles2/template/layout.jsp"

preparer="menuPreparer">

<put-attribute name="header"

value="/WEB-INF/viewtiles2/template/header.jsp" />

<put-attribute name="footer"

value="/WEB-INF/viewtiles2/template/footer.jsp" />

</definition>

<definition name="loginForm" extends="base\_layout">

<put-attribute name="title" value="로그인폼" />

<put-attribute name="body"

value="/WEB-INF/viewtiles2/loginForm.jsp" />

</definition>

<definition name="loginSuccess" extends="base\_layout">

<put-attribute name="title" value="로그인 성공" />

<put-attribute name="body"

value="/WEB-INF/viewtiles2/loginSuccess.jsp" />

</definition>

</tiles-definitions>

[/WEB-INF/dispatcherTiles2-servlet.xml]

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context" xmlns:p="http://www.springframework.org/schema/p"

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-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<bean class="madvirus.spring.chap07.controller.LoginController">

<property name="authenticator">

<bean class="madvirus.spring.chap07.controller.MockAuthenticator" />

</property>

</bean>

<bean id="messageSource"

class="org.springframework.context.support.ResourceBundleMessageSource">

<property name="basenames">

<list>

<value>messages.validation</value>

<value>messages.label</value>

</list>

</property>

</bean>

<bean

class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">

<property name="alwaysUseFullPath" value="false" />

</bean>

<bean

class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">

<property name="alwaysUseFullPath" value="false" />

<property name="cacheSeconds" value="0" />

</bean>

<bean id="tilesConfigurer"

class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">

<property name="definitions">

<list>

<value>/WEB-INF/tiles2def/tilesdef.xml</value>

</list>

</property>

</bean>

<bean id="viewResolver"

class="org.springframework.web.servlet.view.UrlBasedViewResolver">

<property name="viewClass"

value="org.springframework.web.servlet.view.tiles2.TilesView" />

</bean>

</beans>

[web.xml]

...

<servlet>

<servlet-name>dispatcherTiles2</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcherTiles2</servlet-name>

<url-pattern>/tiles2/\*</url-pattern>

</servlet-mapping>

...

[/WEB-INF/viewtiles2/template/layout.jsp]

<%@ page contentType="text/html; charset=EUC-KR"%>

<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles"%>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title><tiles:getAsString name="title" /></title>

</head>

<body>

<tiles:insertAttribute name="header" />

<hr />

<tiles:insertAttribute name="body" />

<hr/>

<tiles:insertAttribute name="footer" />

</body>

</html>

[/WEB-INF/viewtiles2/template/header.jsp]

<%@ page contentType="text/html; charset=EUC-KR"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<c:forEach var="menu" items="${menuList}">${menu} </c:forEach> |

환영합니다!

[/WEB-INF/viewtiles2/template/footer.jsp]

<%@ page contentType="text/html; charset=EUC-KR"%>

스프링 예제.

[/WEB-INF/viewtiles2/loginForm.jsp]

<%@ page contentType="text/html; charset=EUC-KR" %>

<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<form:form commandName="login">

<form:errors />

<p>

<label for="loginType"><spring:message code="login.form.type" /></label>

<form:select path="loginType" items="${loginTypes}" />

</p>

<p>

<label for="id"><spring:message code="login.form.id" /></label>

<form:input id="id" path="id"/>

<form:errors path="id" />

</p>

<p>

<label for="password"><spring:message code="login.form.password" /></label>

<form:password id="password" path="password"/>

<form:errors path="password" />

</p>

<p>

<input type="submit" value="<spring:message code="login.form.submit" />">

</p>

</form:form>

[/WEB-INF/viewtiles2/loginSuccess.jsp]

<%@ page language="java" contentType="text/html; charset=EUC-KR" %>

로그인에 성공했습니다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

4.2 SPRING 빈을 ViewPreparer로 사용하기

: 메뉴 목록처럼 모든 뷰 Code에서 공통으로 사용되는 데이터가 존재할 경우 Tiles에서는 ViewPreparer를 이용한다. 보통 ViewPreparer는 아래 Code와 같이

Tiles 설정 파일에 완전한 Class 이름을 적는 것이 보통이다.

<tiles-definitions>

<definition name="base\_layout"

template="/WEB-INF/viewtiles2/template/layout.jsp"

preparer="madvirus.spring.chap07.tiles2.menuPreparer">

<put-attribute name="header"

value="/WEB-INF/viewtiles2/template/header.jsp" />

<put-attribute name="footer"

value="/WEB-INF/viewtiles2/template/footer.jsp" />

</definition>

...

</titles-definitions>

그런데 이 경우 SPRING이 제공하는 DI/AOP를 ViewPreparer에 적용할 수 없는 단점이 있다. SPRING MVC는 Tiles의 ViewPreparer로 SPRING 빈을 사용할 수

있는 기능을 제공하고 있다. SPRING 빈을 Tiles의 ViewPreparer로 사용하려면 다음의 세 가지만 설정하면 된다.

- TilesConfigurer 설정에 prepareFactoryClass 프로퍼티 설정 추가

- ViewPreparer로 사용할 SPRING 빈 등록

- Tiles 설정 파일의 preparer 속성에 SPRING 빈 이름 등록

먼저, 아래 Code와 같이 TilesConfigurer의 preparerFactoryClass 프로퍼티의 값으로 SpringBeanPreparerFactory Class의 완전한 이름을 설정하고,

ViewPreparer로 사용할 Class를 SPRING 빈으로 등록한다.

<bean id="tilesConfigurer" class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">

<property name="definitions">

<list>

<value>/WEB-INF/tiles2def/tilesdef.xml</value>

</list>

</property>

<property name="preparerFactoryClass" value="org.springframework.web.servlet.view.tiles2.SpringBeanPreparerFactory"/>

</bean>

<bean id="menuPreparer" class="madvirus.spring.chap07.tiles2.MenuPreparer"/>

Tiles 설정 파일에서는 다음과 같이 <definition> 태그의 preparer 속성의 값으로 앞서 설정한 SPRING 빈의 이름을 지정하면, ViewPreparer로

해당 빈을 사용하게 된다.

<tiles-definitions>

<definition name="base\_layout"

template="/WEB-INF/viewtiles2/template/layout.jsp"

preparer="menuPreparer">

<put-attribute name="header"

value="/WEB-INF/viewtiles2/template/header.jsp" />

<put-attribute name="footer"

value="/WEB-INF/viewtiles2/template/footer.jsp" />

</definition>

...

</tiles-definitions>

<Example006> /tiles2/member/regist.do

[/WEB-INF/tiles2def/tilesdef.xml]

...

<definition name="registMemberForm" extends="base\_layout">

<put-attribute name="title" value="회원 가입" />

<put-attribute name="body"

value="/WEB-INF/viewtiles2/registMemberForm.jsp" />

</definition>

<definition name="registMember" extends="base\_layout">

<put-attribute name="title" value="회원 가입 완료" />

<put-attribute name="body"

value="/WEB-INF/viewtiles2/registMember.jsp" />

</definition>

...

[/WEB-INF/dispatcherTiles2-servlet.xml]

...

<bean class="madvirus.spring.chap07.controller.RegistMemberController" />

...

<bean id="tilesConfigurer"

class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">

<property name="definitions">

<list>

<value>/WEB-INF/tiles2def/tilesdef.xml</value>

</list>

</property>

<property name="preparerFactoryClass"

value="org.springframework.web.servlet.view.tiles2.SpringBeanPreparerFactory" />

</bean>

...

<bean id="menuPreparer" class="madvirus.spring.chap07.tiles2.MenuPreparer" />

...

[MenuPreparer.java]

package madvirus.spring.chap07.tiles2;

import java.util.ArrayList;

import java.util.List;

import org.apache.tiles.AttributeContext;

import org.apache.tiles.context.TilesRequestContext;

import org.apache.tiles.preparer.ViewPreparer;

public class MenuPreparer implements ViewPreparer {

@Override

public void execute(TilesRequestContext tilesContext,

AttributeContext attributeContext) {

List<String> menuList = new ArrayList<String>();

menuList.add("홈");

menuList.add("로그인");

menuList.add("회원가입");

tilesContext.getRequestScope().put("menuList", menuList);

}

}

[/WEB-INF/viewtiles2/registMemberForm.jsp]

<%@ page contentType="text/html; charset=EUC-KR" %>

<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<form:form commandName="memberInfo">

<p>

<form:label path="userId">회원 ID</form:label>

<form:input path="userId" />

<form:errors path="userId" />

</p>

<p>

<form:label path="name">이름</form:label>

<form:input path="name" />

<form:errors path="name" />

</p>

<p>

<form:label path="address.address1">주소1</form:label>

<form:input path="address.address1" />

<form:errors path="address.address1" />

</p>

<p>

<form:label path="address.address2">주소2</form:label>

<form:input path="address.address2" />

<form:errors path="address.address2" />

</p>

<p>

<form:label path="jobCode">직업</form:label>

<form:select path="jobCode" >

<option value="">--- 선택하세요 ---</option>

<form:options items="${jobCodes}" itemLabel="label" itemValue="code" />

</form:select>

<form:errors path="jobCode" />

</p>

<p>

<form:label path="favorites">선호 OS</form:label>

<form:checkboxes items="${favoritesOsNames}" path="favorites" />

<form:errors path="favorites" />

</p>

<p>

<form:label path="tool">주로 사용하는 개발툴</form:label>

<form:radiobuttons items="${tools}" path="tool" />

</p>

<p>

<form:label path="etc">기타</form:label>

<form:textarea path="etc" cols="20" rows="3"/>

</p>

<p>

<form:checkbox path="contractAgreement" label="약관에 동의합니다."/>

</p>

<p>

<input type="submit" value="회원 등록">

</p>

</form:form>

[/WEB-INF/viewtiles2/registMember.jsp]

<%@ page contentType="text/html; charset=EUC-KR" %>

회원 등록: ${memberInfo.name}, ${memberInfo.address.address1}, ${memberInfo.address.address2}

===================================================================================================

5. Velocity를 이용한 뷰 구현

: JSP와 더불어 뷰를 생성하는 데 많이 사용되는 기술 중의 하나가 Velocity이다. Velocity는 템플릿 엔진으로서 템플릿 파일을 비교적 쉽게 작성할 수 있고,

Java 객체를 템플릿 파일에서 사용할 수 있기 때문에 뷰를 생성하는데 JSP만큼 널리 사용되고 있다.

앞서 Tiles나 JSP와 마찬가지로 SPRING은 Velocity와의 연동도 지원하고 있는데, 본절에서는 VelocityViewResolver를 이용한 기본적인 연동 방법과

VelocityLayoutViewResolver를 이용하여 Velocity 기반의 레이아웃 템플릿을 적용하는 방법을 살펴보도록 하겠다.

참고로, Velocity를 뷰로 사용하려면 다음의 Velocity 관련.jar 파일을 추가해 주어야 한다.

- Velocity 1.5

- Velocity Tools View 1.4

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

5.1 VelocityViewResolver와 VelocityConfigurer를 이용한 Velocity 연동

: SPRING에서 Velocity를 연동할 때는 VelocityViewResolver를 사용하면 된다. VelocityViewResolver는 Velocity 템플릿 파일을 이용해서 뷰를 생성하며

VelocityConfigurer를 사용해서 Velocity와 관련된 설정 정보를 입력한다. 아래 Code는 설정 예이다.

<bean id="velocityConfigurer" class="org.springframework.web.servlet.view.velocity.VelocityConfigurer">

<property name="resourceLoaderPath" value="/WEB-INF/viewvm/" />

<property name="velocityProperties">

<props>

<prop key="input.encoding">EUC-KR</prop>

<prop key="output.encoding">EUC-KR</prop>

</props>

</property>

</bean>

<bean id="viewResolver" class="org.springframework.web.servlet.view.velocity.VelocityViewResolver" p:suffix=".vm" p:contentType="text/html; charset=EUC-KR" />

VelocityConfigurer의 resourceLoaderPath 프로퍼티는 탬플릿 파일을 로딩할 경로를 입력하며, velocityProperties 프로퍼티는 Velocity 설정 정보를 지정한다.

VelocityViewResolver는 VelocityConfigurer에서 설정한 경로에서 템플릿 파일을 읽어 와 뷰를 생성한다. prefix 프로퍼티와 suffix 프로퍼티는 뷰 이름으로부터 템플릿

파일의 경로를 생성할 때 사용된다. 예를 들어, 위와 같이 설정한 경우, 컨트롤러가 리턴한 뷰 이름이 "loginForm"이라면 실제 사용되는 템플릿 파일은

"/WEB-INF/viewvm/loginForm.vm"이 된다.

템플릿 파일에서는 컨트롤러에서 생성한 모델 정보를 사용할 수 있다. 예를 들어, 아래 Code와 같이 컨트롤러에서 모델 정보를 생성했다고 하자.

@Controller

public class HelloController{

@RequestMapping("/hello.do")

public ModelAndView hello(){

ModelAndView mav = new ModelAndView();

mav.setViewName("hello");

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

return mav;

}

...

}

이 경우 템플릿 파일에서는 다음과 같이 모델의 이름을 사용해서 객체를 사용할 수 있다.

<html>

<head>

<meta http-equiv="Content-Type" content="text;html; charset=EUC-KR">

<title>인사</title>

</head>

<body>

인사말 : <strong>${greeting}</string>

</body>

</html>

(1) Velocity Tools의 NumberTool과 DateTool 설정

: Velocity는 템플릿 파일에서 유용하게 사용할 수 있는 다양한 툴을 구현한 Velocity Tools를 제공하고 있는데, 이 툴에는 숫자의 날짜를 형식에 맞게

출력해주는 NumberTool과 DateTool을 제공하고 있다. VelocityViewResolver는 이 두 가지 툴을 템플릿 파일에서 접근할 때 사용할 이름을 지정할 수 있는

프로퍼티를 제공하고 있다.

- date ToolAttribute : DateTool에 접근할 때 사용할 변수 명을 지정.

- number ToolAttribute : NumberTool에 접근할 때 사용할 변수 명을 지정.

아래 Code는 설정 예이다.

<bean id="viewResolver" class="org.springframework.web.servlet.view.velocity.VelocityViewResolver"

p:suffix=".vm"

p:contentType="text/html; charset=EUC-KR"

p:dateToolAttribute = "dateTool"

p:numberToolAttribute = "numberTool" />

위 Code와 같이 DateTool과 NumberTool 관련 설정을 했다면, 템플릿 파일에서 설정한 이름을 사용해서 툴을 사용할 수 있다.

예를 들면 다음과 같이 "dateTool"을 사용할 수 있다.

<html>

<head>

<meta http-equiv="Content-Type" content="text;html; charset=EUC-KR">

<title>#springMessage("title.currentTime")</title>

</head>

<body>

#springMessage("label.currentTime")

: $dateTool.format("yyyy-MM-dd HH:mm:ss", $time)

</body>

</html>

(2) request 및 session 속성 템플릿 파일에서 사용하기

: VelocityViewResolver는 모델에 저장된 모델 데이터만 뷰에 전달한다. 하지만, 경우에 따라서 request 객체나 session 객체에 저장된 속성(attribute)을

템플릿에서 접근하고 싶을 때가 있을 것이다. 예를 들어, session 객체에 현재 로그인 한 사용자 정보가 저장되어 있는 경우를 생각해 보자.

템플릿 파일에서 session 객체의 속성에 접근할 수 없다면, 다음과 같이 컨트롤러에서 일일이 필요한 값을 모델 객체에 추가해 주어야 할 것이다.

HttpSession session = request.getSession();

ModelAndView mav = new ModelAndView(successView);

mav.addObject("userInfo", session.getAttribute("userInfo");

...

이런 불편함을 없애기 위해, VelocityViewResolver는 다음의 두 프로퍼티의 값을 true로 지정함으로써 request나 session의 속성을 템플릿에 전달할 수

있도록 하고 있다.

- exposeRequestAttributes

- exposeSessionAttributes

아래 Code는 설정 예를 보여주고 있다.

<bean id="viewResolver" class="org.springframework.web.servlet.view.velocity.VelocityViewResolver"

p:suffix=".vm"

p:contentType="text/html; charset=EUC-KR"

p:dateToolAttribute = "dateTool"

p:numberToolAttribute = "numberTool"

p:exposeRequestAttributes="true"

p:exposeSessionAttributes="true"/>

두 프로퍼티의 기본 값은 false이다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

5.2 SPRING이 제공하는 Velocity 매크로

: SPRING이 JSP를 위한 다양한 커스텀 태그를 제공하고 있듯이, Velocity를 위한 매크로도 제공하고 있다. 커스텀 태그와 마찬가지로 입력 폼을 위한 HTML 태그와

에러 Message 등을 생성할 때 매크로를 유용하게 사용할 수 있다.

(1) Message 출력을 위한 #springMessage 매크로와 #springMessageText 매크로

: #springMessage 매크로와 #springMessageText 매크로는 SPRING의 MessageSource로부터 Message를 읽어와 출력할 때 사용된다. 두 매크로는 다음과 같이

정의되어 있다.

- #springMessage(code) : Code에 해당하는 Message를 출력한다. Message가 존재하지 않을 경우 예외가 발생된다.

- #springMessageText(code text) : Code에 해당하는 Message를 출력한다. Message가 존재하지 않을 경우 text를 출력한다.

아래 Code는 #springMessage 매크로의 사용 예를 보여 주고 있다.

<p>

<label for="id">#springMessage("login.form.id")</label>

#springFormInput("login.id" " ")

#springShowErrors("<br/>" " ")

#springMessageText("login.form.id.help", "도움말이 없습니다.")

</p>

위 Code에서 #springMessage 매크로는 login.form.id Message Code에 해당하는 Message를 출력한다. #springMessageText 매크로는 login.form.id.help Message Code에

해당하는 Message를 출력하는데, 만약 해당 Message가 존재하지 않는다면 "도움말이 없습니다"를 대신 출력한다.

(2) 커맨드 객체 연동을 위한 #springBind 매크로와 #springBindEscaped 매크로

: #springBind 매크로는 커맨드 객체와 관련된 BindStatus 정보를 설정합니다.

#springBind 매크로는 다음과 같이 커맨드 객체와 관련된 경로를 설정하며, 관련 BindStatus 정보는 status 변수를 통해서 접근할 수 있다.

<p>

<label for="id">#springMessage("login.form.id")</label>

#springBind("login.id")

<input type="text" name="${status.expression}" id="${status.expression}" value="${status.value}"/>

#springShowErrors("<br/>" " ")

</p>

위 Code는 login 커맨드 객체의 id 프로퍼티를 #springBind 매크로를 사용해서 바인딩 하였다. 이후 id 프로퍼티와 관련된 BindStatus 객체는 status 변수를 통해서

접근할 수 있게 된다.

status.expression은 BindStatus 객체와 연관된 커맨드 객체의 경로를 출력한다. 위 Code의 경우는 "id"를 출력한다. status.value는 커맨드 객체와 관련된 값을 출력하며,

위 Code의 경우는 login 커맨드 객체의 id 프로퍼티의 값을 출력한다. status.value는 defaultHtmlEscaped 컨텍스트 파라미터의 값에 따라 HTML 특수 문자 처리 여부를 결정한다.

만약, HTML 특수 문자 처리 여부를 직접 명시하고 싶다면, 다음과 같이 #springBindEscaped 매크로를 사용하면 된다.

<p>

<label for="id">#springMessage("login.form.id")</label>

#springBindEscaped("login.id" false)

<input type="text" name="${status.expression}" id="${status.expression}" value="${status.value}"/>

#springShowErrors("<br/>" " ")

</p>

#springBindEscaped 매크로의 두 번째 파라미터는 특수문자를 치환할 지의 여부를 설정한다. 위 Code의 경우 특수 문자를 변환하지 않도록 설정하고 있다.

따라서, status.value는 id 프로퍼티의 값에 &나 '와 같은 특수 문자가 포함되어 있더라도 변환하지 않는다.

(3) <input> 태그를 위한 매크로

: #springFormInput 매크로를 사용하면 커맨드 객체와 관련된 <input> 태그를 손쉽게 출력할 수 있다. #springFormInput 매크로는 다음과 같이 정의되어 있다.

- #springFormInput($path $attributes)

$path는 <input> 태그를 생성할 때 사용할 커맨드 객체의 경로이며, $attributes는 <input> 태그에 삽입할 HTML 속성을 설정한다. 아래 Code는 #springFormInput

매크로의 사용 예를 보여 주고 있다.

<label for="name">이름</label>

#springFormInput("memberInfo.name" "class='input'")

#springShowErrors("<br/>" " ")

위 Code에서 #springFormInput 매크로는 memberInfo 커맨드 객체의 name 프로퍼티와 관련된 <input> 태그를 생성한다. 실제로 생성되는 HTML 태그는 다음과 같다.

<input type="text" id="name" name="name" value="" class='input'>

hidden 타입의 <input> 태그나 password 타입의 <input> 태그를 생성하고 싶다면 다음의 두 매크로를 사용하면 된다.

- #springFormPasswordInput($path $attributes) : password 타입의 <input> 태그를 생성한다.

- #springFormHiddenInput($path $attributes) : hidden 타입의 <input> 태그를 생성한다.

위 두 매크로의 사용 방법은 #springFormInput 매크로와 동일하다.

#springFormInput 매크로는 내부적으로 #springBind 매크로를 사용해서 커맨드 객체와 관련된 BindStatus 정보에 접근한다. #springFormInput 매크로는

다음과 같이 정의되어 있다.

#macro(springFormInput $path $attribute)

#springBind($path)

<input type="text" id="${status.expression}" name="${status.expression}" value="$!status.value" ${attributes}#springCloseTag()

#end

#springFormInput 매크로뿐만 아니라 폼과 관련된 모든 매크로는 위 Code와 같이 #springBind 매크로를 이용해서 커맨드 객체와 관련된 BindStatus 정보를

설정한다.

(4) <select> 태그를 위한 매크로

: SPRING은 <select> 태그와 관련하여 다음과 같이 두 개의 매크로를 제공하고 있다.

- #springFormSingleSelect($path $options $attributes) : 한 개 옵션을 선택할 수 있는 <select> 태그를 생성

- #springFormMultiSelect($path $options $attributes) : 다중 선택이 가능한 <select> 태그를 생성

$attributes는 HTML 태그의 속성 설정을 입력한다.

$options는 <option> 태그를 생성할 때 사용할 Map 객체를 지정한다. 예를 들어, 컨트롤러 Class에서 다음과 같이 모델 객체에 Map 객체를 저장했다고 하자.

참고로 LinkedHashMap을 사용하는 이유는 put()한 순서대로 <option> 항목을 화면에 출력하기 위함이다.

@RequestMapping(method=RequestMethod.GET)

public String form(Model model){

referenceData(model);

return formViewName;

}

private void referenceData(Model model){

Map<String, String> jobCodes = new LinkedHashMap<String,String>();

jobCodes.put("1","개발자");

jobCodes.put("2","UI 개발자");

jobCodes.put("3","WEB 디자이너");

jobCodes.put("4","기획자");

...

model.assAttributes("jobCodes", jobCodes);

}

이 경우, 다음 Code와 같이 #springFormSingleSelect 매크로를 사용할 수 있다.

<label for="jobCodes">직업</label>

#springFormSingleSelect("memberInfo.jobCode" ${jobCodes} "")

#springShowErrors("<br/>" "")

#springFormSingleSelect 매크로는 $options에 전달받는 Map 객체의 키를 <option> 태그의 value 속성의 값으로 사용하고, Map 객체의 값을 <option> 태그의

텍스트로 사용한다. 예를 들어, 위 Code는 컨트롤러에서 전달받는 "jobCodes" 모델 객체로부터 다음과 같은 HTML 태그를 생성한다.

<select id="jobCodes" name="jobCode">

<option value="1">개발자</option>

<option value="2">UI 개발자</option>

<option value="3">WEB 디자이너</option>

<option value="4">기획자</option>

</select>

만약, List 객체를 이용해서 <option> 태그를 생성하고 싶다면 #springFormSingleSelect 매크로나 #springFormMultiSelect 매크로를 사용하지 말고,

다음과 같이 #springBind 매크로를 이용해서 직접 <select> 태그와 <option> 태그를 생성해 주어야 한다.

#springBind("login.loginType")

<select id="${status.expression}" name="${status.expression}">

#foreach($option in $loginType)

<option value="${option}"

#if("$!status.value" == "$option")

selected = "selected"

#end>

$option</option>

#end

</select>

(5) checkbox 타입 <input> 태그를 위한 매크로

: checkbox 타입을 위한 <input> 태그를 생성할 때에는 다음과 같은 매크로를 사용한다.

- #springFormCheckboxes($path $options $seperator $attributes)

- #springFormCheckbox($path $attributes)

$options는 checkbox 타입의 <input> 태그를 생성할 때 사용할 Map 객체를 전달받으며, $seperator는 각각의 선택 항목을 구분하기 위한 구분자를 입력한다.

예를 들어, $options에 전달할 Map 객체를 다음과 같이 생성했다고 하자.

Map<String, String> favoritesOsNames = new LinkedHashMap<String,String>();

favoritesOsNames.put("WIN2000","윈도우2000");

favoritesOsNames.put("WINXP","윈도우XP");

favoritesOsNames.put("VISTA","비스타");

favoritesOsNames.put("UBUNTU","우분투");

favoritesOsNames.put("MAC","맥");

...

model.assAttributes("favoritesOsNames", favoritesOsNames);

이 때, "favoritesOsNames" 모델 객체를 사용해서 checkbox 타입의 <input> 태그를 생성하고 싶다면 다음과 같이 #springFormCheckboxes 매크로를

사용하면 된다.

<label for="favoritesOsNames">선호 OS</label>

#springFormCheckboxes("memberInfo.favorites" ${favoritesOsNames} "&nbsp; | &nbsp;" "")

#springShowErrors("<br/>" "")

위 #springFormCheckboxes 매크로가 생성하는 HTML Code는 다음과 같다.

<input type="checkbox" name="favorites" value="WINXP">윈도우 XP</input>

<input type="checkbox" name="favorites" value="WIN2000">윈도우2000</input>

<input type="checkbox" name="favorites" value="MAC">맥</input>

<input type="checkbox" name="favorites" value="UBUNTU">우분투</input>

<input type="checkbox" name="favorites" value="VISTA">비스타</input>

<input type="hidden" name="\_favorites" value="on"/>

#springFormCheckbox 매크로는 바인딩 되는 프로퍼티의 값이 true인 경우 체크박스가 선택된 상태가 되고, true가 아닌 경우 선택되지 않은 상태가 된다.

아래 Code는 #springFormCheckbox 매크로의 사용 예를 보여주고 있다.

<p>

#springFormCheckbox("memberInfo.contractAgreement" "")

<label for="contractAgreement">약관에 동의합니다.</label>

</p>

(6) radio 타입 <input> 태그를 위한 매크로

: radio 타입의 <input> 태그를 생성하기 위한 SPRING 매크로는 다음과 같다.

- #springFormRadioButtons($path $options $seperator $attributes)

#springFormRadioButtons 매크로의 사용 방법은 #springFormCheckboxes 매크로와 동일하다. 아래 Code는 사용 예를 보여 주고 있다.

<p>

<label for="tool">주로 사용하는 개발툴</label>

#springFormRadioButtons("memberInfo.tool" $tools "" "")

</p>

#springFormRadioButtons 매크로가 생성하는 HTML 태그는 다음과 같다.

<input type="radio" name="tool" value="Eclipse">Eclipse

<input type="radio" name="tool" value="NetBeans">NetBeans

<input type="radio" name="tool" value="IntelliJ">IntelliJ

(7) <textarea> 태그를 위한 매크로

: <textarea> 태그를 생성하기 위한 매크로는 다음과 같다.

- #springFormTextarea($path $attributes)

아래 Code는 #springFormTextarea 매크로의 사용예를 보여 주고 있다.

<p>

<label for="etc">기타</label>

#springFormTextarea("memberInfo.etc" "cols='20' rows='3'")

</p>

(8) 에러 Message 출력을 위한 #springShowErrors 매크로

: #springShowErrors 매크로는 커맨드 객체와 관련된 에러 Message를 출력할 때 사용된다. #springShowErrors 매크로는 다음과 같이 정의되어 있다.

- #springShowErrors($seperator $class/style) : #springBind/#springBindEscaped 매크로로 바인딩 된 BindStatus와 관련된 에러 Message들을 출력한다.

$seperator는 각 에러 Message를 구분할 때 출력되는 값이다. $class/style은 각 에러 Message를 위한 CSS 관련 정보를 설정할 때 사용된다.

$class/style과 관련된 정보는 뒤에서 살펴보도록 하겠다.

#springShowErrors 매크로는 컨트롤러에서 생성한 에러 Message를 출력한다. 예를 들어, 아래 Code와 같이 커맨드 객체와 같이 컨맨드 객체와 관련된

에러 정보를 추가했다고 하자.

errors.reject("invalidIdOrPassword", new Object[]{loginCommand.getId()}, null);

이 경우, 다음과 같이 #springBind 매크로와 #springShowErrors 매크로를 사용해서 에러 Message를 출력할 수 있다.

#springBind("login")

#springShowErrors("<br/>" "">

특정 프로퍼티와 관련된 에러 Message를 출력할 때에는 #springBind 매크로를 사용하거나 폼 관련 매크로를 사용하면 된다.

<p>

<label for="id">#springMessage("login.form.id")</label>

#springBind("login.id")

<input type="text" name="${status.expression}" id="${status.expression}" value="$!status.value"/>

#springShowErrors("<br/>" "")

</p>

<p>

<label for="password">#springMessage("login.form.password")</label>

#springFormPasswordInput("login.password" "")

#springShowErrors("<br/>" "")

</p>

#springFormInput 매크로나 #springFormTextArea 매크로와 같은 폼 입력을 위한 매크로는 #springBind 매크로를 사용하기 때문에, 위 Code와 같이

폼 관련 매크로를 사용한 뒤에는 #springShowErrors 매크로를 사용해서 입력 폼과 관련된 에러 Message를 출력할 수 있다.

#springShowErrors 매크로는 두 번째 파라미터 값을 ""로 지정하면 각 에러 Message를 <b> 태그를 사용해서 강조한다. 예를 들면, 다음과 같은 에러 Message를

생성한다.

<b>잘못된 ID나 암호를 입력하셨습니다. 입력한 ID는 124입니다.</b>

<b> 태그가 아닌 CSS 관련 속성을 사용해서 에러 Message를 출력하고 싶다면, 두 번째 파라미터 값에 CSS 관련 정보를 입력하면 된다. 두 번째 파라미터의 값이

""이 아니면, <span> 태그를 이용해서 에러Message를 감싼다. 이때, 두 번째 파라미터의 값이 ":"로 시작하면 <span>의 style 속성의 값을 두 번째 파라미터의

값으로 설정한다. ":"로 시작하지 않으면 class 속성의 값을 두 번째 파라미터 값으로 설정한다.

예를 들어, 아래의 Code에서 첫 번째 springShowErrors 매크로는 ':'으로 시작하는 값을 두 번째 파라미터로 전달하고 있다.

#springFormInput("memberInfo.userId" "")

#springShowErrors("<br/>" ":font-size:15pt:")

#springFormInput("memberInfo.name" "class='input'")

#springShowErrors("<br/>" "error")

첫 번째 #springShowErrors 매크로가 생성하는 에러 Message는 다음과 같이 두 번째 파라미터로 전달받은 값을 style 속성의 값으로 사용한다.

<span style=":font-size=15pt:">필수 항목입니다.</span>

반면에, 두 번째 #springShowErrors 매크로가 생성하는 에러 Message는 다음과 같이 두 번째 파라미터로 전달받은 값을 class 속성의 값으로 사용한다.

<span class="error">필수 항목입니다.</span>

<Example007> /vm/login/login.do, /vm/member/regist.do

[dispatcherVm-servlet.xml]

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context" xmlns:p="http://www.springframework.org/schema/p"

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-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<bean class="madvirus.spring.chap07.controller.LoginController">

<property name="authenticator">

<bean class="madvirus.spring.chap07.controller.MockAuthenticator" />

</property>

</bean>

<bean class="madvirus.spring.chap07.controller.RegistMemberControllerForVelocity" />

<bean class="madvirus.spring.chap07.controller.HelloController" />

<bean class="madvirus.spring.chap07.controller.TimeController" />

<bean id="messageSource"

class="org.springframework.context.support.ResourceBundleMessageSource">

<property name="basenames">

<list>

<value>messages.validation</value>

<value>messages.label</value>

</list>

</property>

</bean>

<bean

class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">

<property name="alwaysUseFullPath" value="false" />

</bean>

<bean

class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">

<property name="alwaysUseFullPath" value="false" />

<property name="cacheSeconds" value="0" />

</bean>

<bean id="velocityConfigurer"

class="org.springframework.web.servlet.view.velocity.VelocityConfigurer">

<property name="resourceLoaderPath" value="/WEB-INF/viewvm/" />

<property name="velocityProperties">

<props>

<prop key="input.encoding">EUC-KR</prop>

<prop key="output.encoding">EUC-KR</prop>

</props>

</property>

</bean>

<bean id="viewResolver"

class="org.springframework.web.servlet.view.velocity.VelocityViewResolver"

p:suffix=".vm" p:contentType="text/html; charset=EUC-KR"

p:dateToolAttribute="dateTool" p:numberToolAttribute="numberTool" />

</beans>

[web.xml]

...

<servlet>

<servlet-name>dispatcherVm</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcherVm</servlet-name>

<url-pattern>/vm/\*</url-pattern>

</servlet-mapping>

...

[RegistMemberControllerForVelocity.java]

package madvirus.spring.chap07.controller;

import java.util.LinkedHashMap;

import java.util.Map;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.validation.BindingResult;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

@Controller

@RequestMapping("/member/regist.do")

public class RegistMemberControllerForVelocity {

private String formViewName = "registMemberForm";

@RequestMapping(method = RequestMethod.GET)

public String form(Model model) {

referenceData(model);

return formViewName;

}

private void referenceData(Model model) {

Map<String, String> jobCodes = new LinkedHashMap<String, String>();

jobCodes.put("1", "개발자");

jobCodes.put("2", "UI 개발자");

jobCodes.put("3", "웹 디자이너");

jobCodes.put("4", "기획자");

Map<String, String> favoritesOsNames = new LinkedHashMap<String, String>();

favoritesOsNames.put("윈도우XP", "윈도우XP");

favoritesOsNames.put("비스타", "비스타");

favoritesOsNames.put("윈도우7", "윈도우7");

favoritesOsNames.put("우분투", "우분투");

favoritesOsNames.put("맥", "맥");

Map<String, String> tools = new LinkedHashMap<String, String>();

tools.put("Eclipse", "Eclipse");

tools.put("IntelliJ", "IntelliJ");

tools.put("NetBeans", "NetBeans");

model.addAttribute("jobCodes", jobCodes);

model.addAttribute("favoritesOsNames", favoritesOsNames);

model.addAttribute("tools", tools);

}

@ModelAttribute

protected Object formBackingObject() throws Exception {

return new MemberInfo();

// MemberInfo member = new MemberInfo();

// member.setAddress(new Address());

// return member;

}

@RequestMapping(method = RequestMethod.POST)

public String submit(@ModelAttribute MemberInfo memberInfo,

BindingResult result, Model model) {

new MemberInfoValidator().validate(memberInfo, result);

checkDuplicateId(memberInfo.getUserId(), result);

if (result.hasErrors()) {

referenceData(model);

return formViewName;

}

return "registMember";

}

private void checkDuplicateId(String userId, BindingResult errors) {

if (userId.equals("madvirus")) {

errors.rejectValue("userId", "duplicate");

}

}

}

[HelloController.java]

package madvirus.spring.chap07.controller;

import java.util.Calendar;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.ModelAndView;

@Controller

public class HelloController {

@RequestMapping("/hello.do")

public ModelAndView hello() {

ModelAndView mav = new ModelAndView();

mav.setViewName("hello");

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 <= 15) {

return "점심 식사는 하셨나요?";

} else if (hour >= 18 && hour <= 22) {

return "좋은 밤 되세요";

}

return "안녕하세요";

}

}

[TimeController.java]

package madvirus.spring.chap07.controller;

import java.util.Date;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.RequestMapping;

@Controller

public class TimeController {

@RequestMapping("/time.do")

public String time(Model model) {

model.addAttribute("time", new Date());

return "time";

}

}

[/WEB-INF/viewvm/hello.vm]

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>인사</title>

</head>

<body>

인사말: <strong>${greeting}</strong>

</body>

</html>

[/WEB-INF/viewvm/loginForm.vm]

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>#springMessage("login.form.title")</title>

</head>

<body>

<form action="login.do" method="post">

#springBind("login")

#foreach($error in $status.errorMessages) <b>$error</b> <br> #end

<p>

<label for="loginType">#springMessage("login.form.type")</label>

#springBind("login.loginType")

<select id="${status.expression}" name="${status.expression}">

#foreach($option in $loginTypes)

<option value="${option}"

#if("$!status.value" == "$option")

selected="selected"

#end>

$option</option>

#end

</select>

</p>

<p>

<label for="id">#springMessage("login.form.id")</label>

#springBind("login.id")

<input type="text" name="${status.expression}" id="${status.expression}" value="$!status.value" />

#springShowErrors("<br/>" "")

</p>

<p>

<label for="password">#springMessage("login.form.password")</label>

#springFormPasswordInput("login.password" "")

#springShowErrors("<br/>" "")

</p>

<p>

<input type="submit" value="#springMessage("login.form.submit")">

</p>

</form>

</body>

</html>

[/WEB-INF/viewvm/loginSuccess.vm]

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>로그인 성공</title>

</head>

<body>

로그인에 성공했습니다.

</body>

</html>

[/WEB-INF/viewvm/registMemberForm.vm]

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>회원 등록 폼</title>

</head>

<body>

<form action="regist.do" method="post">

<p>

<label for="userId">회원 ID</label>

#springFormInput("memberInfo.userId" "")

#springShowErrors("<br/>" ":font-size:15pt; ")

</p>

<p>

<label for="name">이름</label>

#springFormInput("memberInfo.name" "class='input'")

#springShowErrors("<br/>" "error")

</p>

<p>

<label for="address.address1">주소1</label>

#springFormInput("memberInfo.address.address1" "")

#springShowErrors("<br/>" "")

</p>

<p>

<label for="address.address2">주소2</label>

#springFormInput("memberInfo.address.address2" "")

#springShowErrors("<br/>" "")

</p>

<p>

<label for="jobCode">직업</label>

#springFormSingleSelect("memberInfo.jobCode" $jobCodes "")

#springShowErrors("<br/>" "")

</p>

<p>

<label for="favorites">선호 OS</label>

#springFormCheckboxes("memberInfo.favorites" $favoritesOsNames "&nbsp;|&nbsp;" "")

#springShowErrors("<br/>" "")

</p>

<p>

<label for="tool">주로 사용하는 개발툴</label>

#springFormRadioButtons("memberInfo.tool" $tools "" "")

</p>

<p>

<label for="etc">기타</label>

#springFormTextarea("memberInfo.etc" "cols='20' rows='3'")

</p>

<p>

#springFormCheckbox("memberInfo.contractAgreement" "")

<label for="contractAgreement">약관에 동의합니다.</label>

</p>

<p>

<input type="submit" value="회원 등록">

</p>

</form>

</body>

</html>

[/WEB-INF/viewvm/registMember.vm]

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>회원 등록</title>

</head>

<body>

회원 등록: ${memberInfo.name}, ${memberInfo.address.address1}, ${memberInfo.address.address2}

</body>

</html>

[/WEB-INF/viewvm/time.vm]

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>#springMessage("title.currentTime")</title>

</head>

<body>

#springMessage("label.currentTime")

: $dateTool.format("yyyy-MM-dd HH:mm:ss", $time)

</body>

</html>

(9) VelocityLayoutViewResolver를 이용한 레이아웃 템플릿 적용

: Tiles와 비슷하게 Velocity도 동일한 레이아웃 템플릿을 적용해주는 기능을 제공하고 있다. 레이아웃 템플릿을 모든 Velocity 템플릿에 적용하려면

VelocityLayoutViewResolver를 ViewResolver로 사용하도록 설정해 주어야 한다.

<bean id="velocityConfigurer" class="org.springframework.web.servlet.view.velocity.VelocityConfigurer">

<property name="resourceLoaderPath" value="/WEB-INF/viewvmlayout/" />

<property name="velocityProperties">

...

</property>

</bean>

<bean id="viewResolver" class="org.springframework.web.servlet.view.velocity.VelocityLayoutViewResolver"

p:layoutUrl="/template/layout.vm"

p:suffix=".vm"

p:contentType="text/html;charset=EUC-KR"

...

/>

prefix나 suffix 등의 프로퍼티는 VelocityViewResolver와 동일하다. 차이점이 있다면 layoutUrl 프로퍼티를 사용하여 레이아웃으로 사용될 Velocity 템플릿

파일을 지정한다는 것이다.

VelocityLayoutViewResolver는 결과 화면을 생성해 주는 View Class로 VelocityLayoutView 객체를 생성하는데, VelocityViewLayout 객체는 다음과 같이

2단계 과정을 거쳐서 응답 화면을 생성한다.

1) 컨트롤러의 처리 결과를 보여 줄 템플릿 파일을 찾은 뒤, 템플릿 파일로 부터 결과 화면을 생성한다. 이때, 생성된 결과 화면을 screen\_content 변수에 저장한다.

2) layoutUrl 프로퍼티로 지정한 레이아웃 템플릿 파일을 파싱하여 결과화면을 생성한다. 이때, 레이아웃 템플릿 파일은 $screen\_content 변수를 사용하여

1단계에서 생성한 결과 화면을 내부에 포함시키게 된다.

예를 들어, /greeting/hello.htm 요청을 처리한 뒤 그 결과를 /hello.vm 템플릿 파일을 통해서 출력한다고 해 보자. 이 경우, /hello.vm을 통해서 생성된 결과 화면이

레이아웃으로 지정한 /template/layout.vm 템플릿 파일의 특정 영역에 삽입된다. 즉, layout.vm이 포함하고 있는 레이아웃이 hello.vm의 출력 결과에 적용되는 것이다.

(10) Velocity 레이아웃 파일 생성

: 레이아웃으로 사용될 Velocity 템플릿 파일은 일반적인 Velocity 템플릿 파일과 별다른 차이가 없다. 차이점이 있다면 앞서 설명했듯이 처리 결과로 생성된 결과를

포함하기 위한 $screen\_content 변수가 사용된다는 점이다.

<html>

<head>

<meta http-equiv="Content-Type" content="text/html;charset=EUC-KR">

<title>제목</샤싣>

</head>

<body>

#parse("/template/header.vm")

<hr/>

$screen\_content

<hr/>

#parse("/template/footer.vm")

</body>

</html>

다수의 요청에 대해서 동일한 레이아웃 템플릿 파일이 사용되기 때문에 모든 페이지가 동일한 레이아웃을 갖게 된다. 각 요청의 처리 결과를 생성하는 Velocity

템플릿의 출력 결과는 $screen\_content 변수에 위치하게 된다.

Velocity의 #parse나 #include를 사용하면 다수의 레이아웃 템플릿 파일에서 사용되는 부분을 재사용할 수 있다. 예를 들어, 왼쪽 메뉴가 추가된 레이아웃이

필요한 경우 다음과 같이 #parse를 사용하여 head.vm이나 top\_logo.vm등의 내용을 재사용할 수 있을 것이다.

<html>

<head>

<meta http-equiv="Content-Type" content="text/html;charset=EUC-KR">

<title>제목</샤싣>

</head>

<body>

#parse("/template/header.vm")

<hr/>

<div id="left">#parse("/common/left\_menu.vm")</div>

<div id="content" class="content\_side">

$screen\_content

<hr/>

#parse("/template/footer.vm")

</body>

</html>

(11) 레이아웃 커스터마이징

: /a.htm 요청과 /b.htm 요청을 처리 결과를 각각 a.vm과 b.vm을 통해 생성한다고해 보자. 그리고, 공통으로 사용되는 레이아웃으로 layout.vm을

사용했다고 해보자. 이때, layout.vm은 a.vm이나 b.vm에서 생성한 변수를 사용할 수 있다.

-a.vm의 내용-

#set($htmlTitleTail = '미르3')

...

#foreach($content on $contents)

...

$content.title

...

#end

...

-b.vm의 내용-

<html>

<title>위 메이드 - 우리들의 행복한 놀이터

#if($htmlTitleTail) : $htmlTitleTail #end </title>

...

<body>

$(screen\_content)

#parse("/common/bottom.vm")

</html>

위 Code에서 a.vm은 $htmlTitleTail 변수의 값을 지정하였다. 그리고 layout.vm은 $htmlTitleTail 변수가 존재할 경우 해당 값을 출력하는 Code를 갖고 있다.

a.vm이 먼저 파싱된 후, layout.vm이 파싱되고, 추가적으로 a.vm에서 정의한 변수를 layout.vm에서 사용할 수 있기 때문에, /a.htm의 요청 결과는 다음과 같은

HTML Code를 생성하게 된다.

<html>

<title>우리들의 행복한 놀이터 : 미르3 </title>

...

<body>

...

글제목1

글제목2

...

변수를 전달할 수 있는 기능을 사용하면 레이아웃의 일부를 변경할 수 있게 된다. 예를 들어, 레이아웃 템플릿 파일을 다음과 같이 작성했다고 해 보자.

#if($headVm)

#parse($headVm)

#else

#parse("/template/head.vm")

#end

...

위 템플릿 파일은 $headVm 변수가 존재할 경우 변수에 해당하는 vm 파일을 파싱해서 포함하고, $headVm 변수가 존재하지 않을 경우 "/template/head.vm"을

파싱해서 포함한다. /a.htm의 결과는 /template/head.vm이 아닌 /template/head.vm이 아닌 /template/head\_wise.vm을 헤드로 사용해야 한다고 해보자.

이 경우 /a.htm의 처리 결과를 생성하는 a.vm 파일에서는 다음과 같이 $headVm 변수 값으로 '/template/head\_wide.vm'을 지정함으로써 원하는 결과를 얻을 수 있다.

#set($headVm = '/template/head\_wide.vm')

...

부분을 변경하는 것이 아닌 레이아웃 템플릿 자체를 변경할 수도 있다. VelocityLayoutViewResolver를 설정할 때 사용한 기본 레이아웃 템플릿 파일이 아닌 특정한

레이아웃 템플릿 파일을 사용하고 �럽摸�, 다음과 같이 $layout 변수에 레이아웃으로 사용할 템플릿 파일을 지정해 주면 된다.

#set($layout = '/template/layout\_index.vm')

...

비록 VelocityLayoutViewResolver가 Tiles나 Sitemesh처럼 설정 파일을 사용하여 레이아웃을 관리할 수는 없지만, 레이아웃을 관리하는데 있어서 필요한 부분적인

레이아웃의 변경, 전체 레이아웃의 변경과 같은 기능을 제공하고 있다. 따라서, Site에서 사용되는 레이아웃 개수가 2~3개 이내이고 레이아웃을 구성하는

공통 요소의 개수가 적다면 간단한 설정만으로 동일한 레이아웃을 모든 템플릿에 적용할 수 있는 VelocityLayoutViewResolver를 유용하게 사용할 수 있을 것이다.

<Example008> /vmLayout/login/login.do

[dispatcherVmLayout-servlet.xml]

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context" xmlns:p="http://www.springframework.org/schema/p"

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-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<bean class="madvirus.spring.chap07.controller.LoginController">

<property name="authenticator">

<bean class="madvirus.spring.chap07.controller.MockAuthenticator" />

</property>

</bean>

<bean class="madvirus.spring.chap07.controller.RegistMemberControllerForVelocity" />

<bean class="madvirus.spring.chap07.controller.HelloController" />

<bean class="madvirus.spring.chap07.controller.TimeController" />

<bean id="messageSource"

class="org.springframework.context.support.ResourceBundleMessageSource">

<property name="basenames">

<list>

<value>messages.validation</value>

<value>messages.label</value>

</list>

</property>

</bean>

<bean

class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">

<property name="alwaysUseFullPath" value="false" />

</bean>

<bean

class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">

<property name="alwaysUseFullPath" value="false" />

<property name="cacheSeconds" value="0" />

</bean>

<bean id="velocityConfigurer"

class="org.springframework.web.servlet.view.velocity.VelocityConfigurer">

<property name="resourceLoaderPath" value="/WEB-INF/viewvmlayout/" />

<property name="velocityProperties">

<props>

<prop key="input.encoding">EUC-KR</prop>

<prop key="output.encoding">EUC-KR</prop>

</props>

</property>

</bean>

<bean id="viewResolver"

class="org.springframework.web.servlet.view.velocity.VelocityLayoutViewResolver"

p:layoutUrl="/template/layout.vm"

p:suffix=".vm" p:contentType="text/html; charset=EUC-KR"

p:dateToolAttribute="dateTool" p:numberToolAttribute="numberTool" />

</beans>

[web.xml]

...

<servlet>

<servlet-name>dispatcherVmLayout</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcherVmLayout</servlet-name>

<url-pattern>/vmLayout/\*</url-pattern>

</servlet-mapping>

...

[/WEB-INF/viewvmlayout/template/layout.vm]

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>제목</title>

</head>

<body>

#parse("/template/header.vm")

<hr />

$screen\_content

<hr/>

#parse("/template/footer.vm")

</body>

</html>

[/WEB-INF/viewvmlayout/template/header.vm]

환영합니다!

[/WEB-INF/viewvmlayout/template/footer.vm]

스프링 VelocityLayoutViewResolver 예제.

[/WEB-INF/viewvmlayout/hello.vm]

인사말: <strong>${greeting}</strong>

[/WEB-INF/viewvmlayout/loginForm.vm]

<form action="login.do" method="post">

#springBind("login")

#foreach($error in $status.errorMessages) <b>$error</b> <br> #end

<p>

<label for="loginType">#springMessage("login.form.type")</label>

#springBind("login.loginType")

<select id="${status.expression}" name="${status.expression}">

#foreach($option in $loginTypes)

<option value="${option}"

#if("$!status.value" == "$option")

selected="selected"

#end>

$option</option>

#end

</select>

</p>

<p>

<label for="id">#springMessage("login.form.id")</label>

#springBind("login.id")

<input type="text" name="${status.expression}" id="${status.expression}" value="$!status.value" />

#springShowErrors("<br/>" "")

</p>

<p>

<label for="password">#springMessage("login.form.password")</label>

#springFormPasswordInput("login.password" "")

#springShowErrors("<br/>" "")

</p>

<p>

<input type="submit" value="#springMessage("login.form.submit")">

</p>

</form>

[/WEB-INF/viewvmlayout/loginSuccess.vm]

로그인에 성공했습니다.

[/WEB-INF/viewvmlayout/registMember.vm]

회원 등록: ${memberInfo.name}, ${memberInfo.address.address1}, ${memberInfo.address.address2}

[/WEB-INF/viewvmlayout/registMemberForm.vm]

<form action="regist.do" method="post">

<p>

<label for="userId">회원 ID</label>

#springFormInput("memberInfo.userId" "")

#springShowErrors("<br/>" ":font-size:15pt; ")

</p>

<p>

<label for="name">이름</label>

#springFormInput("memberInfo.name" "class='input'")

#springShowErrors("<br/>" "error")

</p>

<p>

<label for="address.address1">주소1</label>

#springFormInput("memberInfo.address.address1" "")

#springShowErrors("<br/>" "")

</p>

<p>

<label for="address.address2">주소2</label>

#springFormInput("memberInfo.address.address2" "")

#springShowErrors("<br/>" "")

</p>

<p>

<label for="jobCode">직업</label>

#springFormSingleSelect("memberInfo.jobCode" $jobCodes "")

#springShowErrors("<br/>" "")

</p>

<p>

<label for="favorites">선호 OS</label>

#springFormCheckboxes("memberInfo.favorites" $favoritesOsNames "&nbsp;|&nbsp;" "")

#springShowErrors("<br/>" "")

</p>

<p>

<label for="tool">주로 사용하는 개발툴</label>

#springFormRadioButtons("memberInfo.tool" $tools "" "")

</p>

<p>

<label for="etc">기타</label>

#springFormTextarea("memberInfo.etc" "cols='20' rows='3'")

</p>

<p>

#springFormCheckbox("memberInfo.contractAgreement" "")

<label for="contractAgreement">약관에 동의합니다.</label>

</p>

<p>

<input type="submit" value="회원 등록">

</p>

</form>

[/WEB-INF/viewvmlayout/time.vm]

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>#springMessage("title.currentTime")</title>

</head>

<body>

#springMessage("label.currentTime")

: $dateTool.format("yyyy-MM-dd HH:mm:ss", $time)

</body>

</html>

===================================================================================================

6. HTML 이외의 뷰 구현

: 지금까지 JSP나 Velocity를 이용하여 응답 결과를 생성하는 방법을 살펴 보았다. 하지만, WEB Application은 파일 Download 기능을 제공하기도 하며, 동적으로 엑셀이나

PDF 파일을 생성해야 하는 경우도 있다. 본 절에서는 파일 Download를 구현하기 위해 커스텀 View Class를 구현하는 방법과 AbstractExcelView Class와 AbstractPdfView

Class를 이용해서 엑셀과 PDF를 동적으로 생성하는 방법에 대해 살펴보도록 하겠다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

6.1 파일 Download 구현을 위한 커스텀 View

: 파일 Download를 구현하는 경우, 컨트롤 Class는 Download 받을 파일과 관련된 정보를 생성해서 뷰에 전달할 것이다. 예를 들어, 아래 Code는 File 객체를 "download"

뷰에 전달하는 컨트롤러이다.

@Controller

public class DownloadController implements ApplicationContextAware {

private WebApplicationContext context = null;

@RequestMapping("/file")

public ModelAndView download() throws Exception {

File downloadFile = getFile();

return new ModelAndView("download", "downloadFile", downloadFile);

}

private File getFile() {

String path = context.getServletContext().getRealPath(

"/WEB-INF/설명.txt");

return new File(path);

}

@Override

public void setApplicationContext(ApplicationContext applicationContext)

throws BeansException {

this.context = (WebApplicationContext) applicationContext;

}

}

파일 Download를 위한 뷰는 JSP나 Velocity가 아니기 때문에, 파일 Download를 위한 커스텀 뷰 Class를 구현해 주어야 한다. 또한, BeanNameViewResolver나

XmlViewResolver를 이용해서 커스텀 뷰 Class를 사용할 수 있도록 알맞게 설정해 주어야 한다. 아래 Code는 설정 예이다.

<bean id="viewResolver" class="org.springframework.web.servlet.view.BeanNameViewResolver"/>

<bean id="download" class="madvirus.spring.chap07.view.DownloadView"/>

파일 Download를 구현하려면 컨텐츠 타입을 "application/octet-stream"과 같이 Download를 위한 타입으로 설정해 주어야 하며, Download 받는 파일 이름을 알맞게

설정하기 위해서는 Content-Disposition 헤더의 값을 알맞게 설정해 주어야 한다. 아래의 SourceCode는 파일 Download를 위한 뷰 Class의 구현 예이다.

[DownloadView.java]

package madvirus.spring.chap07.view;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.OutputStream;

import java.net.URLEncoder;

import java.util.Map;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.util.FileCopyUtils;

import org.springframework.web.servlet.view.AbstractView;

public class DownloadView extends AbstractView {

public DownloadView() {

setContentType("application/download; charset=utf-8");

}

@Override

protected void renderMergedOutputModel(Map<String, Object> model,

HttpServletRequest request, HttpServletResponse response)

throws Exception {

File file = (File) model.get("downloadFile");

response.setContentType(getContentType());

response.setContentLength((int) file.length());

String userAgent = request.getHeader("User-Agent");

boolean ie = userAgent.indexOf("MSIE") > -1;

String fileName = null;

if (ie) {

fileName = URLEncoder.encode(file.getName(), "utf-8");

} else {

fileName = new String(file.getName().getBytes("utf-8"),

"iso-8859-1");

}

response.setHeader("Content-Disposition", "attachment; filename=\""

+ fileName + "\";");

response.setHeader("Content-Transfer-Encoding", "binary");

OutputStream out = response.getOutputStream();

FileInputStream fis = null;

try {

fis = new FileInputStream(file);

FileCopyUtils.copy(fis, out);

} finally {

if (fis != null)

try {

fis.close();

} catch (IOException ex) {

}

}

out.flush();

}

}

라인 21에서는 파일 Download를 위해 컨텐츠 타입을 "application/download"로 설정해 주고 있다. 라인29에서는 Download 되는 파일의 크기를 설정해 주며,

라인40~41에서는 Content-Disposition 헤더를 이용해서 전송되는 파일의 이름을 명시하고 있다. 라인42에서는 전송되는 데이터의 인코딩이 바이너리 타입이라는

것을 명시하였다.

파일 Download와 관련된 헤더를 알맞게 설정한 뒤에는 response의 OutputStream에 파일을 전송하면 된다. 위 Code에서는 SPRING이 제공하는 유틸리티 Class인

FileCopyUtils Class를 이용해서 FileInputStream으로부터 데이터를 읽어와 response의 OutputStream을 통해서 출력하였다.

마지막으로 response의 OutputStream에 데이터를 전송하면 구현이 완료된다.

WEB 브라우저에서는 파일 Download 대화창을 생성해서, 파일을 Download 받을 수 있도록 한다.

<Example009> /download/file

[DownloadController.java]

package madvirus.spring.chap07.controller;

import java.io.File;

import org.springframework.beans.BeansException;

import org.springframework.context.ApplicationContext;

import org.springframework.context.ApplicationContextAware;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.context.WebApplicationContext;

import org.springframework.web.servlet.ModelAndView;

@Controller

public class DownloadController implements ApplicationContextAware {

private WebApplicationContext context = null;

@RequestMapping("/file")

public ModelAndView download() throws Exception {

File downloadFile = getFile();

return new ModelAndView("download", "downloadFile", downloadFile);

}

private File getFile() {

String path = context.getServletContext().getRealPath(

"/WEB-INF/설명.txt");

return new File(path);

}

@Override

public void setApplicationContext(ApplicationContext applicationContext)

throws BeansException {

this.context = (WebApplicationContext) applicationContext;

}

}

[DownloadView.java]

package madvirus.spring.chap07.view;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.OutputStream;

import java.net.URLEncoder;

import java.util.Map;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.util.FileCopyUtils;

import org.springframework.web.servlet.view.AbstractView;

public class DownloadView extends AbstractView {

public DownloadView() {

setContentType("application/download; charset=utf-8");

}

@Override

protected void renderMergedOutputModel(Map<String, Object> model,

HttpServletRequest request, HttpServletResponse response)

throws Exception {

File file = (File) model.get("downloadFile");

response.setContentType(getContentType());

response.setContentLength((int) file.length());

String userAgent = request.getHeader("User-Agent");

boolean ie = userAgent.indexOf("MSIE") > -1;

String fileName = null;

if (ie) {

fileName = URLEncoder.encode(file.getName(), "utf-8");

} else {

fileName = new String(file.getName().getBytes("utf-8"),

"iso-8859-1");

}

response.setHeader("Content-Disposition", "attachment; filename=\""

+ fileName + "\";");

response.setHeader("Content-Transfer-Encoding", "binary");

OutputStream out = response.getOutputStream();

FileInputStream fis = null;

try {

fis = new FileInputStream(file);

FileCopyUtils.copy(fis, out);

} finally {

if (fis != null)

try {

fis.close();

} catch (IOException ex) {

}

}

out.flush();

}

}

[web.xml]

...

<servlet>

<servlet-name>dispatcherNonHtml</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>dispatcherNonHtml</servlet-name>

<url-pattern>/download/\*</url-pattern>

</servlet-mapping>

...

[dispatcherNonHtml-servlet.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"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="downloadController" class="madvirus.spring.chap07.controller.DownloadController" />

<bean id="viewResolver"

class="org.springframework.web.servlet.view.BeanNameViewResolver" />

<bean id="download" class="madvirus.spring.chap07.view.DownloadView" />

</beans>

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

6.2 AbstractExcelView Class를 이용한 엑셀 Download 구현

: 월별로 페이지 뷰 통계 정보나 급여 명세서와 같은 정보를 엑셀로 제공해야 하는 경우가 있는데, SPRING은 엑셀 형식으로 뷰 데이터를 생성할 수 있도록 다음의

두 View Class를 제공하고 있다.

- AbstractExcelView - POI API를 이용하여 엑셀 응답을 생성한다.

- AbstractJExcelView - JExcel API를 이용하여 엑셀 응답을 생성한다.

본 절에서는 POI API를 이용하여 엑셀 응답을 생성하는 AbstractExcelView Class의 사용 방법을 살펴보도록 하겠다. AbstractExcelView Class는 다음과 같은

메서드를 정의 하고 있다.

protected abstract void buildExcelDocument(

Map<String, Object> model, HSSFWorkbook workbook, HttpServletRequest request, HttpServletResponse response) throws Exception;

HSSFWorkbook은 POI API가 제공하는 엑셀 관련 Class이다. 하위 Class는 이 Class를 이용해서 엑셀 문서를 생성하면 된다. 아래의 SourceCode는 AbstractExcelView

Class를 상속받아 엑셀 문서를 생성하는 Class의 작성 예를 보여 주고 있다.

[PageRankView.java]

package madvirus.spring.chap07.view;

import java.util.List;

import java.util.Map;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import madvirus.spring.chap07.controller.PageRank;

import org.apache.poi.hssf.usermodel.HSSFCell;

import org.apache.poi.hssf.usermodel.HSSFRow;

import org.apache.poi.hssf.usermodel.HSSFSheet;

import org.apache.poi.hssf.usermodel.HSSFWorkbook;

import org.springframework.web.servlet.view.document.AbstractExcelView;

public class PageRanksView extends AbstractExcelView {

@SuppressWarnings("unchecked")

@Override

protected void buildExcelDocument(Map<String, Object> model,

HSSFWorkbook workbook, HttpServletRequest request,

HttpServletResponse response) throws Exception {

HSSFSheet sheet = createFirstSheet(workbook);

createColumnLabel(sheet);

List<PageRank> pageRanks = (List<PageRank>) model.get("pageRanks");

int rowNum = 1;

for (PageRank rank : pageRanks) {

createPageRankRow(sheet, rank, rowNum++);

}

}

private HSSFSheet createFirstSheet(HSSFWorkbook workbook) {

HSSFSheet sheet = workbook.createSheet();

workbook.setSheetName(0, "페이지 순위");

sheet.setColumnWidth(1, 256 \* 20);

return sheet;

}

private void createColumnLabel(HSSFSheet sheet) {

HSSFRow firstRow = sheet.createRow(0);

HSSFCell cell = firstRow.createCell(0);

cell.setCellValue("순위");

cell = firstRow.createCell(1);

cell.setCellValue("페이지");

}

private void createPageRankRow(HSSFSheet sheet, PageRank rank, int rowNum) {

HSSFRow row = sheet.createRow(rowNum);

HSSFCell cell = row.createCell(0);

cell.setCellValue(rank.getRank());

cell = row.createCell(1);

cell.setCellValue(rank.getPage());

}

}

<Example010> /download/pageRanks

[PageRanksController.java]

package madvirus.spring.chap07.controller;

import java.util.ArrayList;

import java.util.List;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.ModelAndView;

@Controller

public class PageRanksController {

@RequestMapping("/pageRanks")

public ModelAndView handleRequestInternal() {

List<PageRank> pageRanks = new ArrayList<PageRank>();

pageRanks.add(new PageRank(1, "/bbs/mir2/list"));

pageRanks.add(new PageRank(2, "/bbs/mir3/list"));

pageRanks.add(new PageRank(3, "/bbs/changchun2/list"));

return new ModelAndView("pageRanks", "pageRanks", pageRanks);

}

}

[PageRank.java]

package madvirus.spring.chap07.controller;

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlType;

@XmlAccessorType(XmlAccessType.FIELD)

@XmlType(name = "", propOrder = { "rank", "page" })

public class PageRank {

private int rank;

private String page;

public PageRank() {

}

public PageRank(int rank, String page) {

this.rank = rank;

this.page = page;

}

public int getRank() {

return rank;

}

public String getPage() {

return page;

}

}

[PageRanksView.java]

package madvirus.spring.chap07.view;

import java.util.List;

import java.util.Map;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import madvirus.spring.chap07.controller.PageRank;

import org.apache.poi.hssf.usermodel.HSSFCell;

import org.apache.poi.hssf.usermodel.HSSFRow;

import org.apache.poi.hssf.usermodel.HSSFSheet;

import org.apache.poi.hssf.usermodel.HSSFWorkbook;

import org.springframework.web.servlet.view.document.AbstractExcelView;

public class PageRanksView extends AbstractExcelView {

@SuppressWarnings("unchecked")

@Override

protected void buildExcelDocument(Map<String, Object> model,

HSSFWorkbook workbook, HttpServletRequest request,

HttpServletResponse response) throws Exception {

HSSFSheet sheet = createFirstSheet(workbook);

createColumnLabel(sheet);

List<PageRank> pageRanks = (List<PageRank>) model.get("pageRanks");

int rowNum = 1;

for (PageRank rank : pageRanks) {

createPageRankRow(sheet, rank, rowNum++);

}

}

private HSSFSheet createFirstSheet(HSSFWorkbook workbook) {

HSSFSheet sheet = workbook.createSheet();

workbook.setSheetName(0, "페이지 순위");

sheet.setColumnWidth(1, 256 \* 20);

return sheet;

}

private void createColumnLabel(HSSFSheet sheet) {

HSSFRow firstRow = sheet.createRow(0);

HSSFCell cell = firstRow.createCell(0);

cell.setCellValue("순위");

cell = firstRow.createCell(1);

cell.setCellValue("페이지");

}

private void createPageRankRow(HSSFSheet sheet, PageRank rank, int rowNum) {

HSSFRow row = sheet.createRow(rowNum);

HSSFCell cell = row.createCell(0);

cell.setCellValue(rank.getRank());

cell = row.createCell(1);

cell.setCellValue(rank.getPage());

}

}

[dispatcherNonHtml-servlet.xml]

...

<bean id="pageRanksController" class="madvirus.spring.chap07.controller.PageRanksController" />

...

<bean id="pageRanks" class="madvirus.spring.chap07.view.PageRanksView" />

...

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

6.3 AbstractPdfView Class를 이용한 PDF Download 구현

: SPRING은 iText API를 이용해서 PDF를 생성할 수 있는 AbstractPdfView Class를 제공하고 있다. AbstractPdfView Class는 다음과 같은 메서드를 정의하고 있다.

protected abstract void buildPdfDocument(Map<String, Object> model, Document document, PdfWriter writer,

HttpServletRequest request, HttpServletResponse response) throws Exception;

com.lowagie.text.Document Class는 iText가 제공하는 Class로서, Document 객체에 PDF 문서를 생성하는데 필요한 객체를 추가함으로써 PDF 문서를 생성할 수 있다.

아래 Code는 AbstractPdfView Class를 상속 받아 PDF 문서를 생성하는 뷰 Class의 구현 예를 보여 주고 있다.

[PageReportView.java]

package madvirus.spring.chap07.view;

import java.util.List;

import java.util.Map;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import madvirus.spring.chap07.controller.PageRank;

import org.springframework.web.servlet.view.document.AbstractPdfView;

import com.lowagie.text.Cell;

import com.lowagie.text.Document;

import com.lowagie.text.Font;

import com.lowagie.text.Paragraph;

import com.lowagie.text.Table;

import com.lowagie.text.pdf.BaseFont;

import com.lowagie.text.pdf.PdfWriter;

public class PageReportView extends AbstractPdfView {

@SuppressWarnings("unchecked")

@Override

protected void buildPdfDocument(Map<String, Object> model,

Document document, PdfWriter writer, HttpServletRequest request,

HttpServletResponse response) throws Exception {

List<PageRank> pageRanks = (List<PageRank>) model.get("pageRanks");

Table table = new Table(2, pageRanks.size() + 1);

table.setPadding(5);

BaseFont bfKorean = BaseFont.createFont(

"c:\\windows\\fonts\\batang.ttc,0", BaseFont.IDENTITY\_H,

BaseFont.EMBEDDED);

Font font = new Font(bfKorean);

Cell cell = new Cell(new Paragraph("순위", font));

cell.setHeader(true);

table.addCell(cell);

cell = new Cell(new Paragraph("페이지", font));

table.addCell(cell);

table.endHeaders();

for (PageRank rank : pageRanks) {

table.addCell(Integer.toString(rank.getRank()));

table.addCell(rank.getPage());

}

document.add(table);

}

}

<Example011> /download/pageReport

[PageReportController.java]

package madvirus.spring.chap07.controller;

import java.util.ArrayList;

import java.util.List;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.ModelAndView;

@Controller

public class PageReportController {

@RequestMapping("/pageReport")

public ModelAndView pdfReport() {

List<PageRank> pageRanks = new ArrayList<PageRank>();

pageRanks.add(new PageRank(1, "/bbs/mir2/list"));

pageRanks.add(new PageRank(2, "/bbs/mir3/list"));

pageRanks.add(new PageRank(3, "/bbs/changchun2/list"));

return new ModelAndView("pageReport", "pageRanks", pageRanks);

}

}

[PageReportView.java]

package madvirus.spring.chap07.view;

import java.util.List;

import java.util.Map;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import madvirus.spring.chap07.controller.PageRank;

import org.springframework.web.servlet.view.document.AbstractPdfView;

import com.lowagie.text.Cell;

import com.lowagie.text.Document;

import com.lowagie.text.Font;

import com.lowagie.text.Paragraph;

import com.lowagie.text.Table;

import com.lowagie.text.pdf.BaseFont;

import com.lowagie.text.pdf.PdfWriter;

public class PageReportView extends AbstractPdfView {

@SuppressWarnings("unchecked")

@Override

protected void buildPdfDocument(Map<String, Object> model,

Document document, PdfWriter writer, HttpServletRequest request,

HttpServletResponse response) throws Exception {

List<PageRank> pageRanks = (List<PageRank>) model.get("pageRanks");

Table table = new Table(2, pageRanks.size() + 1);

table.setPadding(5);

BaseFont bfKorean = BaseFont.createFont(

"c:\\windows\\fonts\\batang.ttc,0", BaseFont.IDENTITY\_H,

BaseFont.EMBEDDED);

Font font = new Font(bfKorean);

Cell cell = new Cell(new Paragraph("순위", font));

cell.setHeader(true);

table.addCell(cell);

cell = new Cell(new Paragraph("페이지", font));

table.addCell(cell);

table.endHeaders();

for (PageRank rank : pageRanks) {

table.addCell(Integer.toString(rank.getRank()));

table.addCell(rank.getPage());

}

document.add(table);

}

}

[dispatcherNonHtml-servlet.xml]

...

<bean id="pageReportController" class="madvirus.spring.chap07.controller.PageReportController" />

...

<bean id="pageReport" class="madvirus.spring.chap07.view.PageReportView" />

...

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

6.4 MarshallingView를 이용한 XML 응답 생성

: SPRING3 버전에서는 Java 객체를 XML로 출력해주는 뷰 구현체인 MarshallingView Class가 추가되었다. MarshallingView Class는 Java 객체를 XML 문서로

변환시켜주는 Marshaller와 변환할 모델 키를 각각 marshaller 프로퍼티와 modelKey 프로퍼티로 전달받는다. 아래 Code는 설정 예이다. (Jaxb2Marshaller는

JAXB API를 이용해서 Java 객체와 XML 사이의 Mapping을 처리해주는 API로서 이에 대한 내용은 11장. OXM(Object XML Mapping)에서 자세히 살펴보도록 하겠다.

<bean id="pageXmlReport" class="org.springframework.web.servlet.view.xml.MarshallingView">

<property name="marshaller" ref="marshaller"/>

<property name="modelKey" ref="report"/>

</bean>

<bean id="marshaller" class="org.springframwork.oxm.jaxb.Jaxb2Marshaller">

<property name="classesToBeBound">

<list>

<value>madvirus.spring.chap07.controller.PageRankReport</value>

</list>

</property>

</bean>

<bean id="viewResolver" class="org.springframework.web.servlet.view.BeanNameViewResolver"/>

위 설정에서 Jaxb2Marshaller는 PageRankReport Class를 Mapping 대상 Class로 설정하고 있는데, PageRankReport Class 및 이 Class가 포함하고 있는

PageRank Class는 다음 같이 JAXB2 API를 사용해서 Mapping 정보를 설정하고 있다.

@XmlAccessorType(XmlAccessType.FIELD)

@XmlRootElement(name="rank-report")

public class PageRankReport{

@XmlElement(name="page-rank")

private List<PageRank) pageRanks;

...

}

@XmlAccessorType(XmlAccessType.FIELD)

@XmlType(name="", propOrder={"rank", "page"})

public class PageRank{

private int Rank;

private String page;

...

}

PageRankReport 객체와 PageRank 객체를 생성해서 report 키 값으로 모델에 저장한 뒤 MarshallingView에 전달하는 컨트롤러를 다음과 같이 구현했다고 하자.

@RequestMapping("/pageXmlReport")

public ModelAndView xmlReport(){

List<PageRank> pageRanks = new ArrayList<PageRank>();

pageRanks.add(new PageRank(1,"/bbs/mir2/list"));

pageRanks.add(new PageRank(2,"/bbs/mir3/list"));

pageRanks.add(new PageRank(3,"/bbs/changchun2/list"));

return new ModelAndView("pageXmlReport","report",new PageRankReport(pageRanks));

}

위 컨트롤러 메서드의 실행 결과를 전달받은 MarshallingView가 출력하는 응답 결과는 XML 형식이 된다.

MarshallingView가 생성하는 XML 응답의 컨텐츠 타입은 application/xml이고 캐릭터 인코딩은 UTF-8이다.

<Example012> /download/pageXmlReport

[PageReportController.java]

package madvirus.spring.chap07.controller;

import java.util.ArrayList;

import java.util.List;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.ModelAndView;

@Controller

public class PageReportController {

@RequestMapping("/pageReport")

public ModelAndView pdfReport() {

List<PageRank> pageRanks = new ArrayList<PageRank>();

pageRanks.add(new PageRank(1, "/bbs/mir2/list"));

pageRanks.add(new PageRank(2, "/bbs/mir3/list"));

pageRanks.add(new PageRank(3, "/bbs/changchun2/list"));

return new ModelAndView("pageReport", "pageRanks", pageRanks);

}

@RequestMapping("/pageXmlReport")

public ModelAndView xmlReport() {

List<PageRank> pageRanks = new ArrayList<PageRank>();

pageRanks.add(new PageRank(1, "/bbs/mir2/list"));

pageRanks.add(new PageRank(2, "/bbs/mir3/list"));

pageRanks.add(new PageRank(3, "/bbs/changchun2/list"));

return new ModelAndView("pageXmlReport", "report", new PageRankReport(

pageRanks));

}

@RequestMapping("/pageJsonReport")

public ModelAndView jsonReport() {

List<PageRank> pageRanks = new ArrayList<PageRank>();

pageRanks.add(new PageRank(1, "/bbs/mir2/list"));

pageRanks.add(new PageRank(2, "/bbs/mir3/list"));

pageRanks.add(new PageRank(3, "/bbs/changchun2/list"));

return new ModelAndView("pageJsonReport", "report", new PageRankReport(

pageRanks));

}

}

[PageRankReport.java]

package madvirus.spring.chap07.controller;

import java.util.List;

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

@XmlAccessorType(XmlAccessType.FIELD)

@XmlRootElement(name = "rank-report")

public class PageRankReport {

@XmlElement(name = "page-rank")

private List<PageRank> pageRanks;

public PageRankReport() {

}

public PageRankReport(List<PageRank> pageRanks) {

this.pageRanks = pageRanks;

}

public List<PageRank> getPageRanks() {

return pageRanks;

}

}

[dispatcherNonHtml-servlet.xml]

...

<bean id="pageXmlReport"

class="org.springframework.web.servlet.view.xml.MarshallingView">

<property name="marshaller" ref="marshaller" />

<property name="modelKey" value="report" />

</bean>

<bean id="marshaller" class="org.springframework.oxm.jaxb.Jaxb2Marshaller">

<property name="classesToBeBound">

<list>

<value>madvirus.spring.chap07.controller.PageRankReport</value>

</list>

</property>

</bean>

...

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

6.5 MappingJacksonJsonView를 이용한 JSON 응답 생성

: SPRING3 버전은 Java 객체를 JSON으로 변환해서 보여주는 뷰 구현 Class인 MappingJacksonJsonView를 제공하고 있다.

아래 Code는 MappingJacksonJsonView의 설정 예를 보여주고 있다.

<bean id="pageJsonReport" class="org.springframework.web.servlet.view.json.MappingJacksonJsonView"/>

<bean id="viewResolver" class="org.springframework.web.servlet.view.BeanNameViewResolver"/>

컨트롤러에서는 다음과 같은 뷰 이름으로 MappingJacksonJsonView 타입의 빈을 설정해주면 된다.

@RequestMapping("/pageJsonReport")

public ModelAndView jsonReport(){

List<PageRank> pageRanks = new ArrayList<PageRank>();

pageRanks.add(new PageRank(1,"/bbs/mir2/list"));

pageRanks.add(new PageRank(2,"/bbs/mir3/list"));

pageRanks.add(new PageRank(3,"/bbs/changchun2/list"));

return new ModelAndView("pageJsonReport","report",new PageRankReport(pageRanks));

}

MappingJacksonJsonView는 모델에 저장된 모든 객체를 JSON 형식으로 변환해준다. 예를 들어, 컨트롤러에서 모델에 다음 같이 값을 설정했다고 해 보자.

public String controllerMethod(Model model){

...

model.addAttribute("report", report);

model.addAttribute("summary", summary);

...

}

컨트롤러에서 위와 같이 모델을 설정한 경우 MappingJacksonJsonView는 다음과 같은 형식의 JSON 응답 결과를 생성한다.

{"report" : ... , "summary" : ...}

응답 결과로 생성되는 컨텐츠 타입은 application/json이며 캐릭터 인코딩은 UTF-8이다.

<Example013> /download/pageJsonReport

[PageReportController.java]

package madvirus.spring.chap07.controller;

import java.util.ArrayList;

import java.util.List;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.ModelAndView;

@Controller

public class PageReportController {

@RequestMapping("/pageReport")

public ModelAndView pdfReport() {

List<PageRank> pageRanks = new ArrayList<PageRank>();

pageRanks.add(new PageRank(1, "/bbs/mir2/list"));

pageRanks.add(new PageRank(2, "/bbs/mir3/list"));

pageRanks.add(new PageRank(3, "/bbs/changchun2/list"));

return new ModelAndView("pageReport", "pageRanks", pageRanks);

}

@RequestMapping("/pageXmlReport")

public ModelAndView xmlReport() {

List<PageRank> pageRanks = new ArrayList<PageRank>();

pageRanks.add(new PageRank(1, "/bbs/mir2/list"));

pageRanks.add(new PageRank(2, "/bbs/mir3/list"));

pageRanks.add(new PageRank(3, "/bbs/changchun2/list"));

return new ModelAndView("pageXmlReport", "report", new PageRankReport(

pageRanks));

}

@RequestMapping("/pageJsonReport")

public ModelAndView jsonReport() {

List<PageRank> pageRanks = new ArrayList<PageRank>();

pageRanks.add(new PageRank(1, "/bbs/mir2/list"));

pageRanks.add(new PageRank(2, "/bbs/mir3/list"));

pageRanks.add(new PageRank(3, "/bbs/changchun2/list"));

return new ModelAndView("pageJsonReport", "report", new PageRankReport(

pageRanks));

}

}

[dispatcherNonHtml-servlet.xml]

...

<bean id="pageJsonReport"

class="org.springframework.web.servlet.view.json.MappingJacksonJsonView" />

...

===================================================================================================

7. Locale 처리

: SPRING이 제공하는 <spring:message> 커스텀 태그는 WEB 요청과 관련된 언어 정보를 이용해서 알맞은 언어의 Message를 출력한다. WEB 브라우저 언어 설정을

한국어(ko\_kr)로 했을 때와 영어(en\_us)로 했을때 <spring:message> 커스텀 태그가 언어에 따라 알맞은 Message를 출력해 주는 결과 화면을 설정할 수 있다.

실제로, SPRING MVC는 LocaleResolver를 이용해서 WEB 요청과 관련된 Locale을 추출하고, 이 Locale 객체를 이용해서 알맞은 언어의 Message를 선택하게 된다.

본 절에서는 SPRING이 제공하는 LocaleResolver를 사용해서 Locale을 변경하는 방법에 대해서 살펴보도록 하겠다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

7.1 LocaleResolver Interface

: org.springframework.web.servlet.LocaleResolver Interface는 다음과 같이 정의되어 있다.

package org.springframework.web.servlet;

import java.util.Locale;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public interface LocaleResolver{

Locale resolveLocale(HttpServletRequest request);

void setLocale(HttpServletRequest request, HttpServletResponse response, Locale locale);

}

resolveLocale() 메서드는 요청과 관련된 Locale을 리턴한다. DispatcherServlet은 등록되어 있는 LocaleResolver의 resolveLocale() 메서드를 호출해서 WEB 요청을

처리할 때 사용할 Locale을 구한다.

setLocale() 메서드는 Locale을 변경할 때 사용된다. 예를 들어, 쿠키나 HttpSession에 Locale 정보를 저장할 때에 이 메서드가 사용된다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

7.2 LocaleResolver의 종류

: SPRING이 기존적으로 제공하는 LocaleResolver 구현 Class는 아래와 같다.

클 래 스 설 명

=======================================================================================

AcceptHeaderLocaleResolver WEB 브라우저가 전송한 Accept-Language 헤더로부터 Locale을 선택한다.

setLocale() 메서드를 지원하지 않는다.

=======================================================================================

CookieLocaleResolver 쿠키를 이용해서 Locale 정보를 구한다. setLocale() 메서드는 쿠키에 Locale 정보를 저장한다.

=======================================================================================

SessionLocaleResolver 세션으로부터 Locale 정보를 구한다. setLocale() 메서드는 세션에 Locale 정보를 저장한다.

=======================================================================================

FixedLocaleResolver WEB 요청에 상관 없이 특정한 Locale로 설정한다. setLocale() 메서드를 지원하지 않는다.

=======================================================================================

LocaleResolver를 직접 등록할 경우 빈의 이름을 "localeResolver"로 등록해 주어야 한다.

(1) AcceptHeaderLocaleResolver

: LocaleResolver를 별도로 설정하지 않을 경우 AcceptHeaderLocaleResolver를 기본 LocaleResolver로 사용한다. AcceptHeaderLocaleResolver는

AcceptLanguage 헤더로부터 Locale 정보를 추출한다.

헤더로부터 Locale 정보를 추출하기 때문에, setLocale() 메서드를 이용해서 Locale 설정을 변경할 수 있다.

(2) CookieLocaleResolver

: CookieLocaleResolver는 쿠키를 이용해서 Locale 정보를 저장한다. setLocale() 메서드를 호출하면 Locale 정보를 담은 쿠키를 생성하고, resolveLocale()

메서드는 쿠키로부터 Locale 정보를 가져와 Locale을 설정한다. 만약, Locale 정보를 담은 쿠키가 존재하지 않을 경우, defaultLocale 프로퍼티의 값을

Locale로 사용한다. defaultLocale 프로퍼티의 값이 null인 경우에는 Accept-Language 헤더로부터 Locale 정보를 추출한다.

CookieLocaleResolver는 쿠키와 관련해서 별도 설정을 필요로 하지 않지만, 생성할 쿠키 이름, 도메인, 경로 등의 설정을 직접 하고 싶다면, 아래와 같은

프로퍼티를 알맞게 설정해 주면 된다.

프로퍼티 설 명

====================================

cookieName 사용할 쿠키 이름

====================================

cookieDomain 쿠키 도메인

====================================

cookiePath 쿠키 경로, 기본 값은 "/"이다.

====================================

cookieMaxAge 쿠키 유효 시간

====================================

cookieSecure Security 쿠키 여부, 기본 값은 false이다.

====================================

(3) SessionLocaleResolver

: SessionLocaleResolver는 HttpSession에 Locale 정보를 저장한다. setLocale() 메서드를 호출하면 Locale 정보를 세션에 저장하고, resolveLocale() 메서드는

세션으로부터 Locale을 가져와 WEB 요청의 Locale을 설정한다. 만약, Locale 정보가 세션에 존재하지 않으면, defaultLocale 프로퍼티의 값을 Locale로 사용한다.

defaultLocale 프로퍼티의 값이 null인 경우에는 Accept-Language 헤더로부터 Locale 정보를 추출한다.

(4) FixedLocaleResolver

: FixedLocaleResolver는 WEB 요청에 상관없이 defaultLocale 프로퍼티로 설정한 값을 WEB 요청을 위한 Locale로 사용한다. FixedLocaleResolver는 setLocale() 메서드를

지원하지 않는다. setLocale() 메서드를 호출할 경우 UnsupportedOperationException 예외를 발생시킨다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

7.3 LocaleResolver를 이용한 Locale 변경

: LocaleResolver를 빈으로 등록했다면, 컨트롤러에서 LocaleResolver를 이용해서 Locale을 변경할 수 있게 된다. 예를 들어, 다음과 같이 LocaleResolver를

설정했다고 하자.

<bean class="madvirus.spring.chap07.controller.LocaleChangeController">

<property name="localeResolver" ref="localeResolver"/>

</bean>

<bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver"/>

이 경우, 컨트롤러 Class는 다음과 같이 LocaleResolver의 setLocale() 메서드를 호출해서 클라이언트의 WEB 요청을 위한 Locale을 변경할 수 있다.

import org.springframework.web.servlet.LocaleResolver;

@Controller

public class LocaleChangeController {

private LocaleResolver localeResolver;

@RequestMapping("/changeLanguage")

public String change(@RequestParam("lang") String language,

HttpServletRequest request, HttpServletResponse response) {

Locale locale = new Locale(language);

localeResolver.setLocale(request, response, locale);

return "redirect:/index.jsp";

}

public void setLocaleResolver(LocaleResolver localeResolver) {

this.localeResolver = localeResolver;

}

}

LocaleResolver를 이용해서 Locale을 변경하면, 이 후 요청에 대해서는 지정한 Locale을 이용해서 Message 등을 로딩하게 된다.

RequestContextUtils Class는 WEB 요청과 관련된 LocaleResolver를 구할 수 있는 메서드를 제공하고 있으므로, 위 Code를 다음과 같이 변경할 수도 있다.

@Controller

public class LocaleChangeController2 {

@RequestMapping("/changeLanguage2")

public String change(@RequestParam("lang") String language,

HttpServletRequest request, HttpServletResponse response) {

Locale locale = new Locale(language);

LocaleResolver localeResolver = RequestContextUtils

.getLocaleResolver(request);

localeResolver.setLocale(request, response, locale);

return "redirect:/index.jsp";

}

}

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

7.4 LocaleChangeInterceptor를 이용한 Locale 변경

: Locale을 변경하기 위해 별도의 컨트롤러 Class를 개발한다는 것은 다소 성가신 일이다. 이 경우, SPRING이 제공하는 LocaleChangeInterceptor Class를

사용하면 WEB 요청 파라미터를 이용해서 손쉽게 Locale을 변경할 수 있다.

LocaleChangeInterceptor Class는 HandlerInterceptor로서 다음과 같이 HandlerMapping의 interceptors 프로퍼티에 등록만 하면 설정이 완료된다.

아래 Code는 설정 예이다.

<bean id="localeChangeInterceptor" class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor" p:paramName="language"/>

<bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">

<property name="interceptors">

<list>

<ref bean="localeChangeInterceptor" />

</list>

</property>

</bean>

paramName 프로퍼티는 Locale을 설정할 때 사용할 파라미터 이름을 명시한다. 예를 들어, 위 Code에서는 paramName 프로퍼티의 값으로 language를

설정했는데, 이 경우 language 요청 파라미터를 사용해서 Locale을 변경할 수 있다.

http://localhost:8080/chap07/jsp/login/login.do?language=en

LocaleChangeInterceptor는 paramName 프로퍼티로 설정한 요청 파라미터가 존재할 경우, 파라미터의 값을 이용해서 Locale을 생성한 뒤 LocaleResolver를

이용해서 Locale을 변경한다. 이후, 요청에서는 변경된 Locale이 적용된다.