View

Summary

When the controller processes the request, and the view name and date model is stored on ModelAndView and returned to the DispatcherServlet, the DispatcherServlet gets the view name, receive the real view object from the ViewResolver. The view prints the information of model object stored by the controller.

Here, we'll explain Spring form tag library provided by the spring for convenient data output at View and ViewResolver, and JSP.

- ViewResolver
- View
- Spring Tag Library
- e-government framework Tag Library

Description

ViewResolver

The controller does not create the actual view object within the code but decide the view name. It enables decoupling of controller and view.

```
package com.easycompany.controller.hierarchy;
...
public class EmployeeListController extends AbstractCommandController{
        ...
         @Override
         protected ModelAndView handle(HttpServletRequest request,
                              HttpServletResponse response, Object command, BindException errors)
                              throws Exception {
                  ...
                  List<Employee> employeelist = employeeService.getAllEmployees(commandMap);

                  ModelAndView modelview = new ModelAndView();
                  modelview.addObject("employeelist", employeelist);
                  ...
                  //Without creating View object,
                  //View view = new InternalResourceView("/jsp/employeelist.jsp");
                  //modelview.setView(view);
                  //Save the View name.
                  modelview.setViewName("employeelist");

                  return modelview;
         }
}
```

At this time, ViewResolver, rather than Controller, gets actual view object in DispatcherServlet.
ViewResolver is the interface that returns actual view object with the view name contained in ModelAndView object returned by Controller.
ViewResolver implementation class provided by Spring is as follows:

ViewResolver	Description
XmlViewResolver	Get the view corresponding to view name from XML with mapping information between View name and View class. Default setting file is /WEB-INF/views.xml.
ResourceBundleViewResolver	Get the view corresponding to View name from resource bundle(property file) containing mapping information between View name and

	View class. Default setting file is views.properties.
InternalResourceViewResolver/UrlBasedViewResolver	Can be used conveniently when calling JSP file of specific directory path. View class basically used is InternalResourceView and View name becomes JSP file name.
VelocityViewResolver /FreeMarkerViewResolver	Used at the time of Velocity/FreeMarker connection.

InternalResourceViewResolver/UrlBasedViewResolver

If there is a controller that forwards to the JSP file in the "/jsp/main/abc.jsp" route after processing the business logic, it could write the controller as below and configure on the bean definition file using internalREsourceViewResolver/UrlBasedViewResolver.

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"/>
or
<bean class="org.springframework.web.servlet.view.UrlBasedViewResolver "/>
@Controller
public class HelloController {
        @RequestMapping("...")
        public String hello(){
                ... //business logic processing.
                return "/jsp/main/abc.jsp"; // View name is the route of JSP file.
        }
}
```

Can process more simply using property prefix, suffix of InternalResourceViewResolver/UrlBasedViewResolver. JSP is under the specific directory path and if it is under /jsp/main directory and extension is .jsp ,

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
        p:prefix="/jsp/main/" p:suffix=".jsp" />
or
<bean class="org.springframework.web.servlet.view.UrlBasedViewResolver"
        p:prefix="/jsp/main/" p:suffix=".jsp" />
@Controller
public class HelloController {
        @RequestMapping("...")
        public String hello(){
                ...
                return "abc"; //indicate the area excluding prefix and suffix.
        }
}
```

View name can be simply set.

View

Even though you cans use the view class provided by the Spring, there are cases where the view class should be written directly due to connecting to UI tools.
It is possible to create the view class by implementing the interface view directly. But let's try implementing by expanding AbstractView. You need to implement rederMergedOutputModel method. It contains the method signature as below.

```
protected abstract void renderMergedOutputModel(Map model,
                                                HttpServletRequest request,
                                                HttpServletResponse response) throws Exception
```

We created the view class of rendering the data of model objects in the form of 'text/xml' for using Ajax related open source of Ajax Tags.

```
package com.easycompany.view;

import java.io.PrintWriter;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.view.AbstractView;

public class AjaxXmlView extends AbstractView {

	@Override
	protected void renderMergedOutputModel(Map model,
			HttpServletRequest request, HttpServletResponse response)
			throws Exception {

		response.setContentType("text/xml");
		response.setHeader("Cache-Control", "no-cache");
		response.setCharacterEncoding("UTF-8");

		PrintWriter writer = response.getWriter();
		writer.write((String) model.get("ajaxXml"));
		writer.close();
	}
}
```

Spring Tag Library

meassage tag(<spring:message>)

The spring provides the <spring:message> tag to print the message easily by bringing the message from the message resource file.
Let's create the example of displaying the title of JSP page using <spring:message>.

Resource bundle related setting should be performed in empty definition file.

```
<!-- Message Source-->
<bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource"
	p:basename="messages"/>
```

Set the code value in the message related resource file. Korean can be entered-edited conveniently with help of utilities such as PropertiedEditor.

/easycompany/webapp/WEB-INF/classes/messages_ko.properties

```
...
# -- spring:message --
easaycompany.loginform.title=log in page
easaycompany.employeelist.title=employee information list page
easaycompany.updateemployee.title=employee information update page
easaycompany.insertemployee.title=employee information input page
easaycompany.departmentlist.title=department information list page
easaycompany.updatedepartment.title=department information update page
easaycompany.insertdepartment.title=department information input page
```

Declare the library to use the custom tag in JSP page. And give message key value to code value of <spring:message> tag.

```
...
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title><spring:message code="easaycompany.departmentlist.title"/></title>
...
```

Title of relevant screen will be indicated as "department information list page".

form tag(<form:form>,<form:input>,...)

It is convenient to use the form tag provided by the Spring in developing form related applications. Spring form tag helps to easily output the reference data or command object of model data on the screen.
First of all, use the spring form tag to declare the custom tag library at the page.

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

Spring form tag has the following tags.

<form:form>
<form:form> saves the model attribute defined in property commandName in PageContext so as to allow the access of tags such as <form:input> or <form:hidden>.
Related properties are as follow:

- commandName : model attribute name to refer.
- action : URL to send form
- method : HTTP method(GET,POST) at the time of form transfer
- enctype: data encoding type at the time of form transfer

Use <form:form> tag and open HTML code with source view in the displayed page and HTML FORM tag is displayed as follows.

```
<form:form commandName="department" action="http://myUrl..." method="post"> -> <form id="department" action="http://myUrl..." method="post">

<form:form commandName="department"> -> <form id="department" action="current page URL" method="post">
```

<form:input>
Used to bind the object property designated in commandName to the input tag of HTML text type.
If you write the property name in path, id and name value of input tag of text type becomes the property name and value becomes the relevant property value.

```
<form:form commandName="department">
        <tr>
                <th>Department name</th>
                <td><form:input path="deptname" size="20"/></td>
        </tr>
</form:form>
```

Displayed to HTML as shown below.

```
<form id="department" action="/easycompany/updateDepartment.do?deptid=1100" method="post">
        <tr>
                <th>Department name</th>
```

```
                <td><input id="deptname" name="deptname" type="text" value="Get-together
Menu Innovation Team" size="20"/></td>
        </tr>
</form>
```

<form:password>
Used to bind the object property designated in commandName in input tag of HTML password type.
To indicate binding value, showPassword property should be designated as showPassword="true".

<form:hidden>
Used to bind the object property designated in commandName in the input tag of HTML hidden type.

<form:select>, <form:options>, <form:option>
Used to bind the object property designated in commandName in HTML select, option tag.
Designate property of commandName object in path property of <form:select> as shown below, and give the collection object of List and Map in item property of <form:options> as value.

```
<form:form commandName="department">
        <tr>
                <th>Upper Department</th>
                <td>
                        <form:select path="superdeptid">
                                <option value="">Select the upper department.</option>
                                <form:options items="${deptInfoOneDepthCategory}" />
                        </form:select>
                </td>
        </tr>
</form:form>
```

It is displayed in HTML as shown below. If there is an option value that matches the path property value of <form:select>, selected="selected".

```
<form id="department" action="/easycompany/updateDepartment.do?deptid=1100" method="post">
        <tr>
                <th>Upper department</th>
                <td>
                        <select id="superdeptid" name="superdeptid">
                                <option value="">Select the upper department.</option>
                                <option value="5000">Finance business department</option>
                            <option value="3000">IT Research Center</option>
                            <option value="4000">Public Business Department</option>
                            <option value="1000" selected="selected">Management Planning
Section</option>
                            <option value="2000">Management Support Section</option>
                        </select>
                </td>
        </tr>
</form>
```

<form:checkboxes>

<form:checkbox>

error tag(<form:errors>)

eGov Framework Tag Library

<ui:pagination/>

<ui:pagination/> tag is provided for convenient paging.
Main property of <ui:pagination/> is as follows.

Name	Description	Required or not
paginationInfo	Data required to create paging list. Data type is egovframework.rte.ptl.mvc.tags.ui.pagination.PaginationInfo.	yes
type	Id of class to take charge of paging list rendering. The ID of class that takes charge of paging list rendering. This ID is the key value of property rendererType declared in the empty setting file.	yes
jsFunction	Function name of JavaScript that will hang on the page number. Page number is delivered to default argument.	yes

Declare the library for ui tag and use at the place where paging list is located as shown below. Write down the attribute name of PaginationInfo saved in the Model object in controller in paginationInfo property, and the function name of JavaScript, the link to be caught in each page number of paging list.
type property writes down the entry key value of rendererType property. It is to determine the rendering type in the tag.

1. Set the related class bean.

```
	<!-- For Pagination Tag -->
	<bean id="imageRenderer" class="com.easycompany.tag.ImagePaginationRenderer"/>

	<bean id="textRenderer"
class="egovframework.rte.ptl.mvc.tags.ui.pagination.DefaultPaginationRenderer"/>

	<bean id="paginationManager"
class="egovframework.rte.ptl.mvc.tags.ui.pagination.DefaultPaginationManager">
		<property name="rendererType">
			<map>
				<entry key="image" value-ref="imageRenderer"/>
				<entry key="text" value-ref="textRenderer"/>
			</map>
		</property>
	</bean>
```

2. Use after declaring the library in JSP.

```
<%@ taglib prefix="ui" uri="http://egovframework.gov/ctl/ui"%>
...
<script type="text/javascript">
	function linkPage(pageNo){
		location.href = "/easycompany/employeeList.do?pageNo="+pageNo;
	}
</script>
<body>
...
		<ui:pagination paginationInfo = "${paginationInfo}"
			type="image"
			jsFunction="linkPage"/>
...
</body>
```

See **here** for detailed explanation, how to use or expand <ui:pagination/>.

Reference

- The Spring Framework - Reference Documentation 2.5.6
- Spring Framework API Documentation 2.5.6