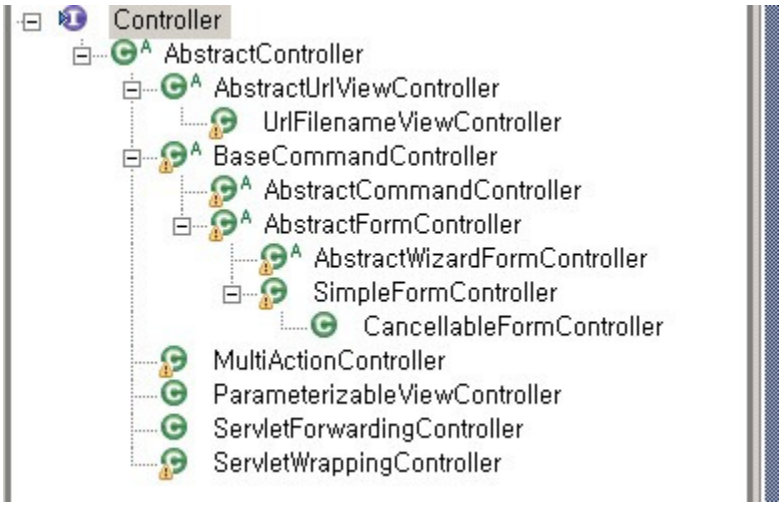
Controller

Summary

DispatcherServlet decides the controller that will process the request using HandlerMapping.
This controller processes the request and reflects the data on the model object.
Spring MVC provides many types of controllers that provide convenient functions such as data binding, form processing and multi actions.
These controllers are the classes that implemented org.springframework.web.servlet.mvc.Controller interface. (@Controller is an exception. Here, the explanation on @Controller is excluded.) When the eclipse interface controller is opened in the hierarchy view, it shows below structure.



Among them, use and feature of main controller can be summarized as shown in the table.

Class	Use and Features
Controller	Default Controller interface. Can be compared with Action of Struts. It is recommended to expand and create the following implementation class rather than directly implementing controller interface during creation of controller in Spring.
AbstractController	Default controller processing web request and response. Since WebContentGenerator is inherited, convenient functions such as designation of HTTP method (POST,GET) and necessity of session are additionally received.
AbstractCommandController	Can bind the parameter of HttpServletRequest dynamically in data object (Command). However, use SimpleFormController for HTML for processing.
SimpleFormController	It is controller used for HTML form processing. Like AbstractCommandController, not only binding parameter and command object of HttpServletRequest, but also provide convenience functions such as filling the data required in input form (referenceData, formBackingObject), view branch according to general form processing scenario (formView, successView).
MultiActionController	Controller used for processing various related actions in one controller.
UrlFilenameViewController	Used to move to view directly from Controller, without processing logic.

Description

AbstractController

In simple request processing and reflecting the result on the ModelandView object, the controller that inherited the AbstractController should be implemented. In implemented controller, the abstract method of AbstractController, handleRequestInternal should be implemented.

protected ModelAndView handleRequestInternal(HttpServletRequest request, HttpServletResponse response) throws Exception;

When creating Controller, instead of implementing interface Controller immediately, implement by inheriting AbstractController to provide convenient functions such as session required check or filtering for specific HTTP method(GET,POST).

Job flow of AbstractController is as follow:

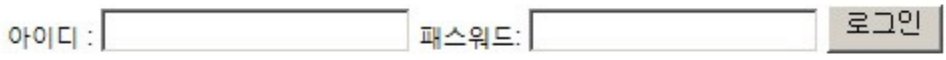
1. handleRequest() by DispatcherServlet is called.
2. Perform filtering for specific HTTP method(GET,POST).
3. If session required value is true, get session out of HttpServletRequest session.
4. Set the cache header according to the value set in cacheSeconds property.
5. Call the abstract method, handleRequestInternal(). If creating the controller class inheriting AbstractController, implemented handleRequestInternal() method is implemented.

Related property can be summarized as follows:

Name	Default Value	Description
supportedMethods	GET,POST	Separate HTTP method list(GET, POST and PUT) supported by controller with comma(,).
requireSession	false	Whether session is required at request processing at Controller. If the value is true and there is no session, ServletException occurs.
cacheSeconds	-1	The time value set in the cache header of response. The unit is second unit. If the value is 0, it has the header that does not perform cache. If it is -1(default value) and does not create any header. If setting the positive value, it creates the header for contents as much as the setting value(second).
synchronizeOnSession	false	Whether to call by synchronizing to HttpSession when calling method handleRequestInternal(). If there is no session, there is no effect.

Example

For user authentication processing, page for entering ID and password is as follows:




```
<%@ page contentType="text/html; charset=UTF-8"%>
<html>
	<head>
	<title>Login Page</title>
	<link type="text/css" rel="stylesheet" href="scripts/easycompany.css" />
</head>
<body>
	<form action="/easycompany/loginProcess.do" method="post">
		id : <input type=text name="id"> password: <input type=password
name="password"> <input type=submit value="login">
	</form>
</body>
</html>
```

Crate the LoginController for login process by expanding AbstractController.
Perform the empty setting as follows first.

```
<bean name="/loginProcess.do" class="com.easycompany.controller.hierarchy.LoginController"
        p:loginService-ref="loginService"
        p:supportedMethods="POST"/> <!—Process when HTTP method is POST only-->
```

Insert actual implementation logic in method handleRequestInternal().

```
package com.easycompany.controller.hierarchy;
...
public class LoginController extends AbstractController{

        private LoginService loginService;

        public void setLoginService(LoginService loginService){
                this.loginService = loginService;
        }

        /**
         *  The implementatio method of handleRequestInternal, abstract method of
AbstractController.
         *  If authentication is successful by receiving ID and password from the user, contain the
account information in session object and forward to the employee information list page.
         *  If authentication fails, return to the login page again.
         */
        @Override
        protected ModelAndView handleRequestInternal(HttpServletRequest request,
                        HttpServletResponse response) throws Exception {

                String id = request.getParameter("id"); //ID
                String password = request.getParameter("password"); //Password

                //After processing the login authentication, return the account related information to
the Account object if the login is successful. If login fails, return null.
                Account account = (Account) loginService.authenticate(id,password);

                if (account != null) { //Login successful
                        request.getSession().setAttribute("UserAccount", account); //save account
information in session.
                        return new ModelAndView("redirect:/employeeList.do"); //move to the
employee list page.
                } else { //failure
                        return new ModelAndView("login"); //move to login page
                }
        }
}
```

AbstractCommandController

AbstractCommandController is used in binding request parameter values to the field value of the command class. Command class should be general JavaBean and it is the difference from Strust that should use form class of dependent structure in framework like ActionForm.
Data binding of command class and parameter generally follows the indication method of JavaBeans property.
If there is a parameter called firstName, find the setFirstName([value]) method of command class and bind the value.
Parameter address.city finds getAddress().setCity([value]) method of command class and binds the value.
This function is convenient function useful for HTML form processing and uses SimpleFormController instead of AbstractCommandController for HTML form processing generally.
Implement abstract method handle() for the implementation controller inheriting AbstractCommandController.

```
protected abstract ModelAndView handle(
```

HttpServletRequest request, HttpServletResponse response, Object command, BindException errors) throws Exception;

Example

Let's create the page showing the employee list according to search conditions such as employee number, department number and employee name.

	사원번호	부서번호	이름	나이	이메일
	1	1200	김길동	28	kkd@easycompany.com
	2	1100	김길수	39	kks@easycompany.com
	3	1200	강감찬	17	kkc@easycompany.com

Bean containing search conditions is as follows.

```
package com.easycompany.domain;
public class SearchCriteria {

        private String searchEid;
        private String searchDid;
        private String searchName;

      set/get functions for above variable...
}
```

Let's create EmployeeListController that receives search conditions from the screen, enter in search condition from the search condition, put in SearchCriteria, the search condition bean, and receive results to the list.
If creating EmployeeListController using AbstractController, take out the value of parameter and directly crate the code that contains the value in the object.

```
package com.easycompany.controller.hierarchy;
...
public class EmployeeListController extends AbstractController{
        ....
      protected ModelAndView handleRequestInternal(HttpServletRequest request,
                        HttpServletResponse response) throws Exception {

                //take out the parameter value of request object
                String searchEid = request.getParameter("searchEid"); //employee number
                String searchDid = request.getParameter("searchDid"); //department number
                String searchName = request.getParameter("searchName"); //employee name
                //save in the object.
                SearchCriteria searchCriteria = new SearchCriteria();
                searchCriteria.setSearchEid(searchEid);
                searchCriteria.setSearchDid(searchDid);
                searchCriteria.setSearchName(searchName);

                List<Employee> employeelist = employeeService.getAllEmployees(searchCriteria);

                ModelAndView modelview = new ModelAndView();
                modelview.addObject("employeelist", employeelist);
                modelview.addObject("searchCriteria", searchCriteria);
                modelview.setViewName("employeelist");
```

```
                return modelview;
        }
}
```

If there is many parameter for mapping, it is very inconvenient task and the line of simple job code gets longer so that the readability of code decreases.
If implementing EmployeeListController by receiving AbstractCommandController, it will be changed as follows.

```
package com.easycompany.controller.hierarchy;
...
public class EmployeeListController extends AbstractCommandController{
        public EmployeeListController(){
                //Declaration for Command object. If there is a declaration for command c\object in
empty setting file, this code is not required.
                setCommandClass(SearchCriteria.class);
                setCommandName("searchCriteria");
        }
        ....

        @Override
        protected ModelAndView handle(HttpServletRequest request,
                        HttpServletResponse response, Object command, BindException errors)
                        throws Exception {
                //binding of parameter and command object is already performed.
                SearchCriteria searchCriteria = (SearchCriteria)command;

                List<Employee> employeelist = employeeService.getAllEmployees(searchCriteria);

                ModelAndView modelview = new ModelAndView();
                modelview.addObject("employeelist", employeelist);
                modelview.addObject("searchCriteria", searchCriteria);
                modelview.setViewName("employeelist");

                return modelview;
        }
}
```

Command class setting can be defined in empty setting file instead of setCommandClass,setCommandName method.

```
<bean id="employeeListController"
class="com.easycompany.controller.hierarchy.EmployeeListController"
        p:employeeService-ref="employeeService"
        p:commandName="searchCriteria"
        p:commandClass="com.easycompany.domain.SearchCriteria"/>
```

This data binding can be more conveniently used if uing with form tag of spring, <form:form>.
Variable **commandName of form tag should agree with name of command class.**
If command object contains the value such as employee number and department number, JSP automatically binds and shows the field value and form field value of command object.

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
...
<form:form commandName="searchCriteria" action="/easycompany/employeeList.do">
<table width="50%" border="1">
        <tr>
                <td>Employee number : <form:input path="searchEid"/></td>
                <td>Department number : <form:input path="searchDid"/></td>
                <td>Name : <form:input path="searchName"/></td>
                <td><input type="submit" value="search"
onclick="this.disabled=true,this.form.submit();" /></td>
```

```
        </tr>
</table>
</form:form>
<table>
        <tr>
                <th></th>
                <th>Employee number</th>
                <th>Department number</th>
                <th>Name</th>
                <th>Age</th>
                <th>E-mail</th>
        </tr>
<c:forEach items="${employeelist}" var="empinfo">
        <tr>
                <td></td>
                <td><a
href="javascript:getEmployeeInfo('${empinfo.employeeid}')">${empinfo.employeeid}</a></td>
                <td>${empinfo.departmentid}</td>
                <td>${empinfo.name}</td>
                <td>${empinfo.age}</td>
                <td>${empinfo.email}</td>
        </tr>
</c:forEach>
</table>
...
```

SimpleFormController

When the controller that deals with form processing such as showing or submitting html forms is written, the controller that inherited SimpleFormController should be implemented. SimpleFormController additionally provides the convenience function such as screen branch(formView, successView) according to results at the time of form transmission while using the convenience function such as initial data setting in the input form, validation of input value, session form mode and data binding of command(form) class and parameter provided by the upper classes such as BaseCommandController and AbstractFormController as they are.
The controller sending and transmitting the form may not be created separately but made as a one.

To view the job flow of SimpleFormController, see handleRequestInternal() method of AbstractFormController, the upper class. Job flow is as shown below.

GET Type Calling

1. Controller receives request for form page. (GET type calling)
2. formBackingObject() method creates and returns command objects for request by default. In general, it inserts the logic of getting the default value to fill in the form when calling GET type by overriding formBackingObject.
3. While initBinder() method is executed, enable to use custom editor for specific field of command class.
4. If property bindOnNewForm is true, fill the value of new command value with initial request parameters. ServletRequestDataBinder is applied and onBindOnNewForm() callback method is called.
5. showForm() method calls referenceData() and save the reference data(mainly, select box, checkbox type) to display in the form page in ModelAndView.
6. formView fill and display the data required for form, based on model data.

POST Type Calling

1. User sends form data(submit). (POST type calling)
2. getCommand() method returns command object.
   If sessionForm is false, call formBackingObject() method and returns the instance of command class.
   If sessionForm is true, take out and return the command object from session. If it fails to find

the relevant object from session, call handleInvalidSubmit() method. handleInvalidSubmit() creates new command object and tries to send the form again.
3. ServletRequestDataBinder is used to fill the command objects with relevant parameter.
4. Call onBind() method. Perform tasks required before performing the validation.
5. If property validateOnBinding value is true, registered Validator is called. Validator examines validation for the field value of command object.
6. Call onBindAndValidate() method. User-Defined job can be performed after inspecting the validation and binding.
7. Perform the transmission in processFormSubmission() method. If there is an error as a result of validation test, showForm() method is called to move to formView again. If there is no error, onSubmit() method is performed to submit the form.
8. If form submission is successful, move to successView.

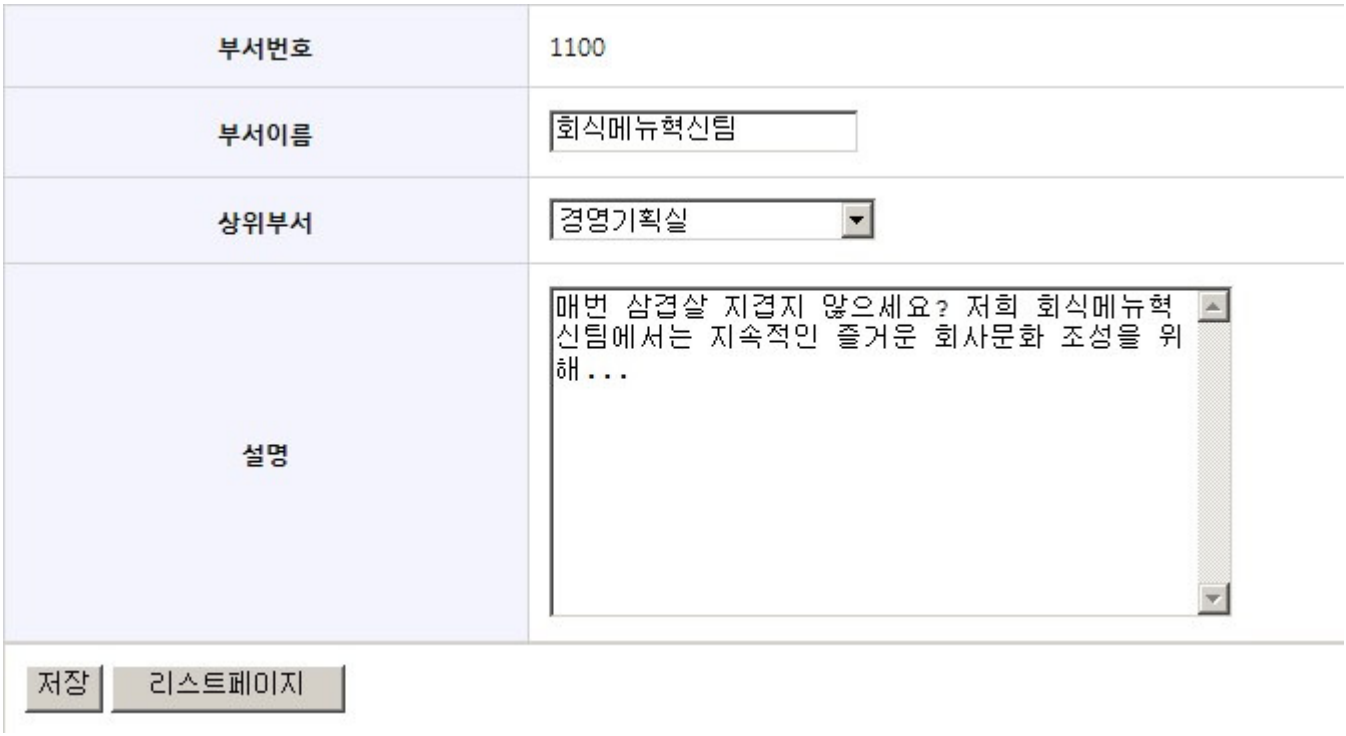
Related property is as shown below.

Name	Default Value	Description	Relevant Class
commandName	command	Name of command class (alias)	BaseCommandController
commandClass	null	Command class to bind data and request parameter	BaseCommandController
validators	null	Arrangement of Validator bean to perform the data validation of command object	BaseCommandController
validator	null	Use if Validator is one.	BaseCommandController
validateOnBinding	true	Whether to perform validation. Perform if it is true.	BaseCommandController
bindOnNewForm	false	Whether to perform data binding at the time when new form is shown.	AbstractFormController
sessionForm	false	Whether to save and use the command object in session.	AbstractFormController
formView	null	Display the view to use when an error occurs in the validation or form page entered by the user.	SimpleFormController
successView	null	Display view to display when form submission is successful.	SimpleFormController

Example

Let's make department information modification page(/easycompany/webapp/WEB-INF/jsp/modifydepartment.jsp).
The processing flow of this page is as follows.

1. Show the input for page to the user and fill the existing department information. → Processed according to process of ***GET type calling*** mentioned above.
2. The user presses the Save button after modifying the contents to modify. → Processed according to process of ***POST type calling.***
    1. If saving is successful or there is a problem in validation of input value, move to initial input form page.
    2. If it is successfully saved, move to the department information list page.




```
<form:form commandName="department">
<table>
	<tr>
		<th>Department Number</th>
		<td><c:out value="${department.deptid}"/></td>
	</tr>
	<tr>
		<th>Department Name</th>
		<td><form:input path="deptname" size="20"/></td>
	</tr>
	<tr>
		<th>Upper Department</th>
		<td>
			<form:select path="superdeptid">
				<option value="">Select the upper department.</option>
				<form:options items="${deptInfoOneDepthCategory}" />
			</form:select>
		</td>
	</tr>
	<tr>
		<th>Description</th>
		<td><form:textarea path="description" rows="10" cols="40"/></td>
	</tr>
</table>
<table width="80%" border="1">
	<tr>
		<td>
		<input type="submit" value="Save"/>
		<input type="button" value="List Page"
onclick="location.href='/easycompany/departmentList.do?depth=1'"/>
		</td>
	</tr>
</table>
</form:form>
```

Register UpdateDepartmentController to perform processing in empty setting file(xxx-servlet.xml) as shown below.

```
<bean id="updateDepartmentController"
class="com.easycompany.controller.hierarchy.UpdateDepartmentController"
        p:departmentService-ref="departmentService"
        p:commandName="department"
        p:commandClass="com.easycompany.domain.Department"
        p:formView="modifydepartment"
        p:successView="redirect:/departmentList.do?depth=1"/>
```

formBackingObject Method

protected Object formBackingObject(HttpServletRequest request) throws Exception

Generally, the revised form page is submitted after the user completes the existing data to the form and revises the necessary part. The formBackingObject method plays the role of calling the data and completing the form. formBackingObject method creates and returns the command object. As necessary, override this method and fill the necessary data to the command object.

Let's create the area shown by filling out the existing department information in department information updating page.
If it is HTTP GET method in override method, inquire the department information table with the department ID in the parameter and add the logic of putting and returning the results to the object department.

```
package com.easycompany.controller.hierarchy;
...

public class UpdateDepartmentController extends SimpleFormController{

        private DepartmentService departmentService;

        public void setDepartmentService(DepartmentService departmentService){
                this.departmentService = departmentService;
        }

        @Override
        protected Object formBackingObject(HttpServletRequest request) throws Exception {
                if(!isFormSubmission(request)){     // If it is GET request
                        String deptid = request.getParameter("deptid");
                        Department department =
departmentService.getDepartmentInfoById(deptid);//The result of inquiring DB with department ID
reflects the command object.
                        return department;
                }else{    // If it is POST request
                        //Call formBackingObject of AbstractFormController to perform default data
binding between command object set and parameter of request object.
                        return super.formBackingObject(request);
                }
        }
       ...
}
```

referenceData Method

protected Map referenceData(HttpServletRequest request) throws Exception
protected Map referenceData(HttpServletRequest request, Object command, Errors errors) throws Exception

There are data that is difficult to include in the command object which is previewed in the form page.

In department information update page, upper department information is select box. The department object, command object contains the upper department number of relevant department only, but there

is no information on which upper department exists in the company.
Use referenceData method to use this reference data required for page even if it is not located in command object.
referenceData method returns the map object. This map is contained in the model object and saved in ModelAndView.
We just need to override referenceData method and put the reference data in the map object.

```
package com.easycompany.controller.hierarchy;
...
public class UpdateDepartmentController extends SimpleFormController{

	private DepartmentService departmentService;

	public void setDepartmentService(DepartmentService departmentService){
		this.departmentService = departmentService;
	}

	@Override
	protected Map referenceData(HttpServletRequest request, Object command, Errors errors)
throws Exception{

		Map param = new HashMap();
		param.put("depth", "1");
		Map referenceMap = new HashMap();

	referenceMap.put("deptInfoOneDepthCategory",departmentService.getDepartmentIdNameList
(param));	//Get the upper department information and put in the Map.
		return referenceMap;
	}
	...
}
```

If the upper department value of relevant department should be shown as "selected" by default in the select box among the upper department information list brought when opening department information modification page, hard work may be required for comparing the values matching command objects with condition sentence one by one. But, it can be solved simply using the spring form tag.
IF referenceData method matches upper department of relevant department in the information map data of upper department, <form:options> prints the relevant option as "selected".

```
<form:select path="superdeptid">
	<option value="">Please select the upper department.</option>
	<form:options items="${deptInfoOneDepthCategory}" />
</form:select>
```

onSubmit Method

protected ModelAndView onSubmit(Object command) throws Exception
protected ModelAndView onSubmit(Object command, BindException errors) throws Exception
protected ModelAndView onSubmit(HttpServletRequest request, HttpServletResponse response, Object command, BindException errors) throws Exception

Let's apply the onSubmit() logic of creating overridden method and submitting the form.

Create the method overriding onSubmit() method and put the logic of transmitting the contents of form.
Reflect the contents of department information object(Department), command object in DB. If it is successful, move to successView and if it fails, call showForm method.
This showForm method inserts the data and error information required to show form page again in ModelAndView and move to formView.

```
package com.easycompany.controller.hierarchy;
...
```

```
public class UpdateDepartmentController extends SimpleFormController{

	private DepartmentService departmentService;

	public void setDepartmentService(DepartmentService departmentService){
		this.departmentService = departmentService;
	}
	...
	@Override
	protected ModelAndView onSubmit(HttpServletRequest request,
			HttpServletResponse response, Object command, BindException errors)
throws Exception{

		Department department = (Department) command;

		try {
			departmentService.updateDepartment(department);
		} catch (Exception ex) {
			return showForm(request, response, errors);
		}

		return new ModelAndView(getSuccessView(), "department", department);
	}
	...
}
```

MultiActionController

The controller examined so far creates controller one by one in one action. (even if SimpleFormController performs branch processing according to GET, POST method for one url.)
To collect related several actions in one controller, inherit MultiActionController and create Controller to create one action with one method.
That method has the following format.

```
public (ModelAndView | Map | String | void) actionName(HttpServletRequest request,
HttpServletResponse response);
```

Then, it is required to determine which method is to process which action(url). The interface that helps now is MethodNameResolver.
MethodNameResolver returns the method name which method will process the request. Spring provides 3 kinds of MethodNameResolver implementation class as follows.

- ParameterMethodNameResolver : gives method name to specific parameter. Default parameter is action.
- InternalPathMethodNameResolver : the area excluding the extension in the last part of URL path becomes the method name.
- PropertiesMethodNameResolver : set the URL and method name in mapping property.

Example

Let's create DepartmentListController that processes the action to get upper department list and the action to gets lower department list that belongs to specific upper department.

```
package com.easycompany.controller.hierarchy;
...
public class DepartmentListController extends MultiActionController {

	...
	//Get the upper department list.
	public ModelAndView departmentList(HttpServletRequest request, HttpServletResponse
response){
```

```
                String depth = request.getParameter("depth");

                Map paramMap = new HashMap();
                paramMap.put("depth", depth);

                List<Department> departmentlist =
departmentService.getDepartmentList(paramMap);

                ModelAndView mav = new ModelAndView("departmentlist");
                mav.addObject("departmentlist", departmentlist);
                return mav;
        }

        //Get the list of lower departments that belong to specific upper department.
        public ModelAndView subDepartmentList(HttpServletRequest request, HttpServletResponse
response){

                String superdeptid = request.getParameter("superdeptid");
                String depth = request.getParameter("depth");

                Map paramMap = new HashMap();
                paramMap.put("depth", depth);
                paramMap.put("superdeptid", superdeptid);

                List<Department> departmentlist =
departmentService.getDepartmentList(paramMap);

                ModelAndView mav = new ModelAndView("departmentsublist");
                mav.addObject("departmentlist", departmentlist);
                return mav;
        }
}
```

If using ParameterMethodNameResolver

```
<bean name="/departmentList.do"
class="com.easycompany.controller.hierarchy.DepartmentListController"
        p:departmentService-ref="departmentService"
        p:methodNameResolver-ref="paramResolver"/> <!—Use ParameterMethodNameResolver as
MethodNameResolver-->

<bean id="paramResolver"
        class="org.springframework.web.servlet.mvc.multiaction.ParameterMethodNameResolver"
        p:paramName="method"/> <!—Parameter name is "method"-->
```

ParameterMethodNameResolver is to designate the information which method to call in the parameter. The parameter name is the value of property “paramName”.

- /easycompany/departmentList.do?depth=1&method=departmentList → departmentList()
- /easycompany/departmentList.do?superdeptid=1000&depth=2&method=subDepartmentList → subDepartmentList()

If using InternalPathMethodNameResolver

```
<bean id="departmentController"
class="com.easycompany.controller.hierarchy.DepartmentListController"
        p:departmentService-ref="departmentService"
        p:methodNameResolver-ref="pathResolver"/> <!—Use InternalPathMethodNameResolver as
MethodNameResolver-->

<bean id="pathResolver"
```

```
        class="org.springframework.web.servlet.mvc.multiaction.InternalPathMethodNameResolver"/
>
```

Use InternalPathMethodNameResolver and it is mapped to public ModelAndView foo(HttpServletRequest, HttpServletResponse) method when request is received at URL /abc/foo.do. In the example, mapping between URL and method is performed as follows.

- /easycompany/departmentList.do?depth=1 → departmentList()
- /easycompany/subDepartmentList.do?superdeptid=1000&depth=2 → subDepartmentList()

If using PropertiesMethodNameResolver

```
<bean id="departmentController"
class="com.easycompany.controller.hierarchy.DepartmentListController"
        p:departmentService-ref="departmentService"
        p:methodNameResolver-ref="propResolver"/> <!—Use InternalPathMethodNameResolver as
MethodNameResolver-->

<bean id="propResolver"
        class="org.springframework.web.servlet.mvc.multiaction.PropertiesMethodNameResolver">
       <property name="mappings">
           <props>
               <prop key="/departmentList.do">departmentList</prop>
               <prop key="/subDepartmentList.do">subDepartmentList</prop>
           </props>
       </property>
</bean>
```

PropertiesMethodNameResolver indicates the mapping relation between URL and method in "mappings" property.

UrlFilenameViewController

UrlFilenameViewController is the controller used to move to view directly without processing logic in Controller.
While it should go through DispatcherServlet, it is used to show static page focusing on html.
As shown below, URL path becomes the view name, and prefix and suffix can be designated too.

```
"/index" -> "index"
"/index.html" -> "index"
"/index.html" + prefix "pre_" and suffix "_suf" -> "pre_index_suf"
"/products/view.html" -> "products/view"
```

It is not required to create Controller separately, but map the url and UrlFilenameViewController in the empty setting file as shown below.

```
        <bean id="urlMapping"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
                <property name="mappings">
                <props>
                        <prop key="/login.do">urlFilenameViewController</prop>
                        <prop key="/validator.do">urlFilenameViewController</prop>
                </props>
                </property>
        </bean>
        <bean id="urlFilenameViewController"
class="org.springframework.web.servlet.mvc.UrlFilenameViewController" />
```

If InternalResourceViewResolver is declared as follow,

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

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

when a request is received at URL http://localhost:8080/easycompany/login.do, find and show /easycompany/webapp/WEB-INF/jsp/login.jsp.

Reference

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