Annotation-based Controller

Summary

Spring 2.5 introduces an approach for writing annotated Web MVC controllers, which we haven't been blogging about much yet.
Main difference from and improvement with existing hierarchy controllers such as SimpleFormController and MultiActionController implementing interface controller are as follows:

1. Setting using Annotation: define the information set based on XML using annotation.
2. Softened Method Signature: select the return type and parameter of Controller method more diversely as needed.
3. Controller of POJO-Style: no need to implement specific interface during Controller development or to inherit the specific class. However, can easily implement functions provided by existing hierarchy controller such as form processing or multiple actions.

Explain the example of implementing the form processing created with hierarchy Controller using @Controller.
The controller of example code easycompany creates the same function(in addition, common Service, DAO, JSP) with hierarchy Controller and @Controller respectively.

- Hierarchy Controller - package com.easycompany.controller.hierarchy
- @Controller - package com.easycompany.controller.annotation

Description

Configuration Using Annotation

It is essentially an alternative controller type supported by the DispatcherServlet / DispatcherPortlet, not implementing a specific interface but rather using annotations to express request mappings for specific handler methods. It is primarily a next-generation style for implementing multi-action controllers.
@MVC enables MVC programming more conveniently by setting annotation in Controller code.
Flexible model transfer - model transfer via a name/value Map supports easy integration with any view

Name	Description
@Controller	Controllers are the components that form the `'C'` part of the MVC
@RequestMapping	Handler mappings handle the execution of a list of pre- and post-processors and controllers that will be executed if they match certain criteria
@RequestParam	View resolvers are components capable of resolving view names to views
@ModelAttribute	View resolvers are components capable of resolving view names to views
@SessionAttributes	A multipart file resolver offers the functionality to process file uploads from HTML forms

@Controller

Only @Controller shall be attached to written classes to create the controller in @MVC. It is not necessary to realize or inherit a specific class.

```
package com.easycompany.controller.annotation;

@Controller
public class LoginController {
   ...
}
```

As mentioned in DefaultAnnotationHandlerMapping earlier, declare package containing @controller using <context:component-scan> tag.
Use filters such as include, exclude if scanning @Controller only.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
        xmlns:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
                                        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd">

        <context:component-scan base-package="com.easycompany.controller.annotation" />

</beans>
```

@RequestMapping

@RequestMapping is an annotation used to map which controller or method should be processed on requests.

Following is the property used by @RequestMapping.

Name	Type	Description
value	String[]	Grant the mapping conditions with URL value. Indicate like @RequestMapping(value=”/hello.do”) or @RequestMapping(value={”/hello.do”, ”/world.do” }), can be indicated as @RequestMapping(”/hello.do”) since it is default value. Use the pattern matching of Ant-Style like ”/myPath/*.do”.
method	RequestMethod[]	Grant the HTTP Request method value as mapping condition. Enable mapping only when HTTP request method value matches. Indicate in the form of @RequestMapping(method = RequestMethod.POST). Available methods are GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE.
params	String[]	Grant the HTTP Request parameter as mapping condition. If params=“myParam=myValue”, there should parameter of myParam in HTTP Request URL, and mapping if the value is myValue. Condition can be given with parameter name such as params=“myParam” and ”!myParam” can map the requests without parameter called myParam. If condition is given like @RequestMapping(params={“myParam1=myValue”, “myParam2”, ”!myParam3”}), HTTP Request should have myValue value called parameter myParam1. There should be myParam2 parameter and no parameter of myParam3.

@RequestMapping can be set in the class unit(type level) or method unit(method level).

type level
If there is a /hello.do request, hello method of HelloController is implemented.

```
@Controller
@RequestMapping("/hello.do")
public class HelloController {

    @RequestMapping    //mapping is properly performed when defining URL at type level and
indicating @RequestMapping over the method that will take charge of request processing even if there
is one method in controller.
    public String hello(){
        ...
    }
}
```

method level
/If hello.do request is received, hello method,

/helloForm.do   request is GET   type, helloGet method is performed. If it is POST type, helloPost method is performed.

```
@Controller
public class HelloController {

        @RequestMapping(value="/hello.do")
        public String hello(){
                ...
        }

        @RequestMapping(value="/helloForm.do", method = RequestMethod.GET)
        public String helloGet(){
                ...
        }

        @RequestMapping(value="/helloForm.do", method = RequestMethod.POST)
        public String helloPost(){
                ...
        }
}
```

type + method level

Both can be set. In this case, cannot define value(URL) of type level 에 설정한 @RequestMapping set in the type level at the method level again.
/If it is GET type at hello.do request, helloGet method is performed. If it is POST type, helloPost method is performed.

```
@Controller
@RequestMapping("/hello.do")
public class HelloController {

        @RequestMapping(method = RequestMethod.GET)
        public String helloGet(){
                ...
        }

        @RequestMapping(method = RequestMethod.POST)
        public String helloPost(){
                ...
        }
}
```

Let’s implement LoginController, the example code implemented by inheriting AbstractController as the Controller based on annotation.
Existing LoginController is the controller implementing handleRequestInternal method when the HTTP method of request to URL /loginProcess.do is POST.

```
package com.easycompany.controller.annotation;
...
@Controller
public class LoginController {

        @Autowired
        private LoginService loginService;

        @RequestMapping(value = "/loginProcess.do", method = RequestMethod.POST)
        public String login(HttpServletRequest request) {

                String id = request.getParameter("id");
                String password = request.getParameter("password");
                Account account = (Account) loginService.authenticate(id,password);

                if (account != null) {
                        request.getSession().setAttribute("UserAccount", account);
                        return "redirect:/employeeList.do";
                } else {
                        return "login";
                }
        }
}
```

@Autowired was used to call the service class in the above example code. See here for details.

@RequestParam

@RequestParam is an annotation to map controller method parameters and web request parameters.

Name	**Type**	**Description**
value	String	Parameter Name
required	boolean	a class offering caching support and, for example, the setting of the mimetype.

It is used in the method shown in the following code.
To bind the null value when the relevant parameter is not in Request object, indicate as required=false like pageNo parameter.
Since name parameter is true for equired, if the name parameter is null, org.springframework.web.bind.MissingServletRequestParameterException occurs.

```
@Controller
public class HelloController {

    @RequestMapping("/hello.do")
    public String hello(@RequestParam("name") String name, //If there is no required condition, the
default value is true, that is, required parameter. If parameter pageNo does not exist, Exception occurs.
                        @RequestParam(value="pageNo", required=false) String pageNo){ //if
parameter pageNo does not exist, String pageNo is null.
        ...
    }
}
```

See the login method of LoginController created above obtains parameter ID and password using getParameter method in Http Request.
It can be changed as follows by using @RequestParam.

```
package com.easycompany.controller.annotation;
...
@Controller
public class LoginController {

        @Autowired
        private LoginService loginService;

        @RequestMapping(value = "/loginProcess.do", method = RequestMethod.POST)
        public String login(
                        HttpServletRequest request,
                        @RequestParam("id") String id,
                        @RequestParam("password") String password) {

                Account account = (Account) loginService.authenticate(id,password);
```

```
                    if (account != null) {
                              request.getSession().setAttribute("UserAccount", account);
                              return "redirect:/employeeList.do";
                    } else {
                              return "login";
                    }
          }
}
```

@ModelAttribute

The attribute of @ModelAttribute is as below:

Name	**Typee**	**Description**
value	String	Name of Model properties to bind.

Exceptions that are thrown during processing of the request get picked up by any of the handler exception resolvers that are declared in the WebApplicationContext. Using these exception resolvers allows you to define custom behaviors in case such exceptions get thrown

@ModelAttribute functions identically as ModelMap.addAttribute. It is used in two ways in the controller.

1.The binding of method return data and model attribute.
The logic saving the result data in ModelMap object after processing business logic(DB processing) in method generally occurs.

```
...
          @RequestMapping(value = "/updateDepartment.do", method = RequestMethod.GET)
          public String formBackingObject(@RequestParam("deptid") String deptid, ModelMap model) {
                    Department department = departmentService.getDepartmentInfoById(deptid); //Get
department information data from DB.
                    model.addAttribute("department", department); //save the data in the model object.
                    return "modifydepartment";
          }
...
```

If selecting @ModelAttribute in method, the return data of relevant method is saved in ModelMap object.
Above code can be changed as follows. When GET type of /updateDepartment.do calling is received from the user, before formBackingObject method is executed, DefaultAnnotationHandlerMapping executes (@ModelAttribute is declared)getEmployeeInfo using org.springframework.web.bind.annotation.support.HandlerMethodInvoker, and saves the result in ModelMap object.
As a result, getEmployeeInfo method performs ModelMap.addAttribute("department", departmentService.getDepartmentInfoById(…)) work.

```
...
          @RequestMapping(value = "/updateDepartment.do", method = RequestMethod.GET)
          public String formBackingObject() {
                    return "modifydepartment";
          }

          @ModelAttribute("department")
          public Department getEmployeeInfo(@RequestParam("deptid") String deptid){
                    return departmentService.getDepartmentInfoById(deptid); //Get department
information from DB.
          }
          or
          public @ModelAttribute("department") Department
getDepartmentInfoById(@RequestParam("deptid") String deptid){
                    return departmentService.getDepartmentInfoById(deptid);
          }
...
```

2.The binding of method parametor and model attribute
@ModelAttribute can be used for binding with the parameter of specific attribute method of ModelMap object.
As shown below, declare **"@ModelAttribute("department") Department department"** in the parameter of method and **(Department)ModelMap.get("department")** value is bound in department.
Accordingly, if it is the code shown below, formBackingObject method parameter department contains the Department data that getDepartmentInfo method saves in ModelMap object.

```
...
          @RequestMapping(value = "/updateDepartment.do", method = RequestMethod.GET)
          public String formBackingObject(@ModelAttribute("department") Department department)
{ //department contains the data obtained from getDepartmentInfo.
                    System.out.println(employee.getEmployeeid());
                    System.out.println(employee.getName());
                    return "modifydepartment";
          }

          @ModelAttribute("department")
          public Department getDepartmentInfo(@RequestParam("deptid") String deptid){
                    return departmentService.getDepartmentInfoById(deptid); //Get department
information data from DB.
          }
...
```

@SessionAttributes

@SessionAttribute is an annotation used in saving and maintaining the model attribute in the session. @SessionAttribute can declare at the type level. The related attriuutes are as following.

Name	Type	Description
types	Class[]	Class that implements WebApplicationContext
value	String[]	Class that implements WebApplicationContext

Flexible Method Signature

The method of the controller that applied @RequestMapping can utilize the below method parameters and return types.

Since there is no restriction to expand the specific class or to implement interface, it has flexible method signature compared to hierarchy controller.

Method Parameter of @Controller

Available method parameter is as follows.

- **Servlet API** – Servlet APIs related to request, response and sessions such as ServletRequest, HttpServletRequest, HttpServletResponse, HttpSession.
- **WebRequest, NativeWebRequest** - org.springframework.web.context.request.WebRequest, org.springframework.web.context.request.NativeWebRequest
- **java.util.Locale**
- **java.io.InputStream / java.io.Reader**
- **java.io.OutputStream / java.io.Writer**

- **@RequestParam** –annotation to use to bind argument of method and parameter of HTTP Request.
- **java.util.Map / org.springframework.ui.Model / org.springframework.ui.ModelMap** – model data to deliver to view.
- **Command/form object** – can reduce alias using @ModelAttribute, command object of binding parameter delivered to HTTP Request.
- **Errors, BindingResult** – object saving the result data after org.springframework.validation.Errors / org.springframework.validation.BindingResult validation .
- **SessionStatus** – used to remove relevant session when processing org.springframework.web.bind.support.SessionStatus   session form.

Method can use the parameter in the random order. Provided, however, that if BindingResult is used as an argument of method, command object for binding should come in front.

```
public String updateEmployee(...,@ModelAttribute("employee") Employee employee,
                              BindingResult bindingResult,...) <!-- (O) -->

public String updateEmployee(...,BindingResult bindingResult,
                          @ModelAttribute("employee") Employee employee,...) <!-- (X) -->
```

To use other types as method parameter?

The spring framework provides the interface, org.springframework.web.bind.support.WebArgumentResolver, to enable custom arguments to be used as a method parameter which is not mentioned above. For examples using WebArgumentResolver, follow this link.

Method return type of @Controller

Available method return type is as follows.

- **ModelAndView** – contains view information and model object containing the return data of method applied with @ModelAttribute and command object.

```
        @RequestMapping(value = "/updateDepartment.do", method = RequestMethod.GET)
         public ModelAndView formBackingObject(@RequestParam("deptid") String deptid) {
                  Department department =
departmentService.getDepartmentInfoById(deptid);
                  ModelAndView mav = new ModelAndView("modifydepartment");
                  mav.addObject("department", department);
                  return mav;
         }
  or
         public ModelAndView formBackingObject(@RequestParam("deptid") String deptid,
ModelMap model) {
                  Department department =
departmentService.getDepartmentInfoById(deptid);
                  model.addAttribute("department", department);
                  ModelAndView mav = new ModelAndView("modifydepartment");
                  mav.addAllObjects(model);
                  return mav;
  }
```

- **Model(or ModelMap)** – Return data of method applied with @ModelAttribute and command object is contained in Model object.
  View name is determined by RequestToViewNameTranslator using URL. Following is the method that implementation class of interface RequestToViewNameTranslator, DefaultRequestToViewNameTranslator, determines the View name.
- http://localhost:8080/gamecast/display.html -> display
- http://localhost:8080/gamecast/displayShoppingCart.html -> displayShoppingCart
  http://localhost:8080/gamecast/admin/index.html -> admin/index

```
  @RequestMapping(value = "/updateDepartment.do", method = RequestMethod.GET)
  public Model formBackingObject(@RequestParam("deptid") String deptid, Model model) {
              Department department = departmentService.getDepartmentInfoById(deptid);
              model.addAttribute("department", department);
              return model;
  }
or
  @RequestMapping(value = "/updateDepartment.do", method = RequestMethod.GET)
  public Model formBackingObject(@RequestParam("deptid") String deptid) {
              Department department = departmentService.getDepartmentInfoById(deptid);
              Model model = new ExtendedModelMap();
              model.addAttribute("department", department);
              return model;
  }
```

- **Map** – return data of method applied with @ModelAttribute and command object is contained in the Map object, and the view name is also determined by RequestToViewNameTranslator.

```
            @RequestMapping(value = "/updateDepartment.do", method = RequestMethod.GET)
            public Map formBackingObject(@RequestParam("deptid") String deptid) {
                        Department department =
departmentService.getDepartmentInfoById(deptid);
                        Map model = new HashMap();
                        model.put("department", department);
                        return model;
            }
  or
            @RequestMapping(value = "/updateDepartment.do", method = RequestMethod.GET)
            public Map formBackingObject(@RequestParam("deptid") String deptid, Map model)
{
                        Department department =
departmentService.getDepartmentInfoById(deptid);
                        model.put("department", department);
                        return model;
  }
```

- **String** – String value to return becomes View name. Return data of method applied with @ModelAttribute and command object is contained in Model(or ModelMap). Model(or ModelMap) object to return should be declared in argument of relevant method.

```
           <!--(O)-->
            @RequestMapping(value = "/updateDepartment.do", method = RequestMethod.GET)
            public String formBackingObject(@RequestParam("deptid") String deptid, ModelMap
model) {
                        Department department =
departmentService.getDepartmentInfoById(deptid);
                        model.addAttribute("department", department);
                        return "modifydepartment";
            }

           <!--(X)-->
            @RequestMapping(value = "/updateDepartment.do", method = RequestMethod.GET)
            public String formBackingObject(@RequestParam("deptid") String deptid) {
                        Department department =
departmentService.getDepartmentInfoById(deptid);
                        ModelMap model = new ModelMap();
                        model.addAttribute("department", department);
                        return "modifydepartment";
  }
```

- **View** – Return View. Return data of method applied with @ModelAttribute and command object is contained in Model(or ModelMap).
- **void** – If method processes direct response using ServletResponse / HttpServletResponse. View name is determined by RequestToViewNameTranslator.

Controller of POJO-Style

@MVC does not need to inherit specific classes or implement certain interfaces in developing the controller. It is not mandatory to refer to ServletAPI on the method of the controller. The development is possible with more flexible method signature.

We'll show that the functions provided at existing controller can be still implemented even if it becomes closer to POJO-Style by implementing the form processing action of SimpleFormController with @Controller.

FormController by SimpleFormController -> @Controller

We'll create com.easycompany.controller.hierarchy.UpdateDepartmentController created as an example while describing SimpleFormController earlier, using @ModelAttribute and @RequestMapping with @Controller.

Use the same JSP source. See the example screen image and JSP code [here].
Existing UpdateDepartmentController is performed in 3 methods.

- referenceData – get the upper department information, the reference data required in input form, and save in Map object. After that, this Map object is saved in ModelMap object by spring internal logic.
- formBackingObject – return the department data to be entered in initial input form when it is GET type calling. This data is also saved in ModelMap object.
- onSubmit – called at POST transmission and process the form transmission.

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

public class UpdateDepartmentController extends SimpleFormController{

        private DepartmentService departmentService;

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

        //Since upper department list(selectbox) is not in department information class, get the upper
department list data from DB and configure with separate reference data.
        @Override
        protected Map referenceData(HttpServletRequest request, Object command, Errors errors)
throws Exception{

                Map referenceMap = new HashMap();

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

        @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);//result of inquiring DB with department ID is
reflected in the command object.
                        return department;
                }else{  // If POST request
                        //Call formBackingObject of AbstractFormController and the default data
binding occurs between command objects set and parameter of request object.
```

```
                        return super.formBackingObject(request);
                }
        }

        @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);
        }
}
```

com.easycompany.controller.annotation.UpdateDepartmentController created with @Controller consists of 3 methods.
Different from existing UpdateDepartmentController, the hierarchy controller, does not require overriding, method name can be created freely.
Select the name of method same as SimpleFormController for easy comparison.

- referenceData – get the upper department information, reference data required for input form and save in ModelMap.(by @ModelAttribute)
- formBackingObject – take charge of processing when it is GET type calling. Get the department data for initial input form configuration and save in ModelMap.
- onSubmit – called at POST transmission, and process the form transmission.

(close to POJO) framework codes are hidden and business contents can be expressed more intuitively.

```
package com.easycompany.controller.annotation;

...
@Controller
public class UpdateDepartmentController {

        @Autowired
        private DepartmentService departmentService;

        //Since upper department list(selectbox) is not in department information class, get the data
of upper department list from DB and configure as a separate reference data.
        @ModelAttribute("deptInfoOneDepthCategory")
        public Map<String, String> referenceData() {
                return departmentService.getDepartmentIdNameList("1");
        }

        // Get the department information data of relevant department number and fill out the input
form.
        @RequestMapping(value = "/updateDepartment.do", method = RequestMethod.GET)
        public String formBackingObject(@RequestParam("deptid") String deptid, ModelMap model) {
                Department department = departmentService.getDepartmentInfoById(deptid);
                model.addAttribute("department", department); //commandName of form tag should
agree with attribute name. <form:form commandName="department">.
                return "modifydepartment";
        }
```

```
	//If the user finishes modification of data and press the Save button, the service(DB) is called,
in charge of modification data.
	//If it is successfully saved, move to the department list page and if there is an error, move to
input form page again.
	@RequestMapping(value = "/updateDepartment.do", method = RequestMethod.POST)
	public String onSubmit(@ModelAttribute("department") Department department,
BindingResult bindingResult) {

		//validation code
		new DepartmentValidator().validate(department, bindingResult);
		if(bindingResult.hasErrors()){
			return "modifydepartment";
		}

		try {
			departmentService.updateDepartment(department);
			return "redirect:/departmentList.do?depth=1";
		} catch (Exception e) {
			e.printStackTrace();
			return "modifydepartment";
		}
	}
}
```

Reference

- The Spring Framework - Reference Documentation 2.5.6
- Spring Framework API Documentation 2.5.6
- SpringSource Team Blog,Annotated Web MVC Controllers in Spring 2.5, Juergen Hoeller