Courtesy: <http://krams915.blogspot.sg/2010/12/spring-3-mvc-using-modelattribute-in.html>

Spring 3 MVC: Using @ModelAttribute in Your JSPs

In this tutorial we will create a simple Spring 3 MVC application that uses JSPs for the presentation layer. We will create a simnple CRUD system using the *@ModelAttribute*

**What is @ModelAttribute**

@ModelAttribute has two usage scenarios in controllers. When you place it on a method parameter, @ModelAttribute maps a model attribute to the specific, annotated method parameter (see the processSubmit() method below). This is how the controller gets a reference to the object holding the data entered in the form.  
  
You can also use @ModelAttribute at the method level to provide reference data for the model (see the populatePetTypes() method in the following example). For this usage the method signature can contain the same types as documented previously for the @RequestMapping annotation.  
  
**Note**  
@ModelAttribute annotated methods are executed ***before*** the chosen @RequestMapping annotated handler method. They effectively pre-populate the implicit model with specific attributes, often loaded from a database. Such an attribute can then already be accessed through @ModelAttribute annotated handler method parameters in the chosen handler method, potentially with binding and validation applied to it.  
  
Source: [Spring 3 Reference Documentation](http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/mvc.html#mvc-ann-modelattrib)

The @ModelAttribute is a convenient annotation for exposing your objects to your JSP pages, and returning the object back to your Controllers.  
  
If you're familiar with the classin Spring of doing things, this annotation combines functionalities of a*formBackingObject (or commandObject)* and SimpleFormController's *referenceData*. However, as of Spring 3.0, these have been deprecated in favor of annotated controllers (Source: [Spring 3 API SimpleFormController](http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/web/portlet/mvc/SimpleFormController.html))  
  
Let's start by declaring the required Spring configurations.  
  
To enable Spring MVC we need to add it in the web.xml  
  
**web.xml**

[?](http://krams915.blogspot.sg/2010/12/spring-3-mvc-using-modelattribute-in.html?showComment=1409074022476)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <servlet>   <servlet-name>spring</servlet-name>   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>   <load-on-startup>1</load-on-startup>  </servlet>    <servlet-mapping>   <servlet-name>spring</servlet-name>   <url-pattern>/krams/\*</url-pattern>  </servlet-mapping>    <listener>   <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener> |

Take note of the URL pattern. When accessing any pages in our MVC application, the host name must be appended with

/krams

In the web.xml we declared a servlet-name spring. By convention, we must declare a spring-servlet.xml as well.  
  
**spring-servlet.xml**

[?](http://krams915.blogspot.sg/2010/12/spring-3-mvc-using-modelattribute-in.html?showComment=1409074022476)

|  |  |
| --- | --- |
| 1  2  3 | <!-- Declare a view resolver -->  <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"       p:prefix="/WEB-INF/jsp/" p:suffix=".jsp" /> |

This XML config declares a view resolver. All references to a JSP name in the controllers will map to a corresponding JSP in the /WEB-INF/jsp location.  
  
By convention, we must declare an applicationContext.xml  
  
**applicationContext.xml**

[?](http://krams915.blogspot.sg/2010/12/spring-3-mvc-using-modelattribute-in.html?showComment=1409074022476)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <!-- Activates various annotations to be detected in bean classes -->  <context:annotation-config />    <!-- Scans the classpath for annotated components that will be auto-registered as Spring beans.   For example @Controller and @Service. Make sure to set the correct base-package-->  <context:component-scan base-package="org.krams.tutorial" />    <!-- Configures the annotation-driven Spring MVC Controller programming model.  Note that, with Spring 3.0, this tag works in Servlet MVC only!  -->  <mvc:annotation-driven /> |

This XML config declares three beans to activate the Spring 3 MVC programming model.  
  
We're done with the required Spring XML configurations. We now focus on the two usage patterns of the @ModelAttribute annotation.  
  
**Pattern 1: Method Level**

You can also use @ModelAttribute at the method level to provide reference data for the model (see the populatePetTypes() method in the following example). For this usage the method signature can contain the same types as documented previously for the @RequestMapping annotation.  
Source: [Spring 3 Reference Documentation](http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/mvc.html#mvc-ann-modelattrib)

This means we declare a method in our controller and annotate the method with @ModelAttribute.  
  
We will examine this pattern by displaying a list of addresses. First let's declare our controller.  
  
**AddressController**

[?](http://krams915.blogspot.sg/2010/12/spring-3-mvc-using-modelattribute-in.html?showComment=1409074022476)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78 | package org.krams.tutorial.controller;    import java.util.List;    import javax.annotation.Resource;    import org.apache.log4j.Logger;  import org.krams.tutorial.domain.Address;  import org.krams.tutorial.service.AddressService;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.ModelAttribute;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;    /\*\*   \* Handles and retrieves Addresses related requests.   \* <p>   \* This demonstrates how we can populate a JSP page using @ModelAttribute and Model   \*/  @Controller  @RequestMapping("/address")  public class AddressController {     protected static Logger logger = Logger.getLogger("controller");     @Resource(name="addressService")   private AddressService addressService;     /\*\*    \* Retrieves all addresses and allows them to be used as a model    \* @return a model attribute containing addresses    \*/   @ModelAttribute("addresses")   public List<Address> getAllAddresses() {    // Delegate to service    return addressService.getAll();   }     /\*\*    \* Handles and retrieves a JSP page containing all addresses.    \* We use the @ModelAttribute to pass the data to the view    \*    \* @return the name of the JSP page    \*/      @RequestMapping(value="list1", method = RequestMethod.GET)      public String getAllUsingModelAttribute() {       logger.debug("Received request to show all addresses page");         // No need to add the model here       // It has been automatically added when we used the @ModelAttribute annotation earlier       // The name of the ModelAttribute is "addresses". Your JSP should reference "addresses"         // This will resolve to /WEB-INF/jsp/addressespage.jsp       return "addressespage";   }     /\*\*    \* Handles and retrieves a JSP page containing all addresses.    \* We use the Model to pass the data to the view    \*    \* @return the name of the JSP page    \*/      @RequestMapping(value="list2", method = RequestMethod.GET)      public String getAllUsingModel(Model model) {       logger.debug("Received request to show all addresses page");         // Here we add the model manually       // This should give the same result with the extra greetings       // The name of the Model is "addresses". Your JSP should reference "addresses" as well       model.addAttribute("addresses", addressService.getAll());       model.addAttribute("greetings", "I came from Model not ModelAttribute");         // This will resolve to /WEB-INF/jsp/addressespage.jsp       return "addressespage";   }    } |

The controller has a method named **getAllAddresses()** that's been annotated with @ModelAttribute. We also provide a model name **addresses**

[?](http://krams915.blogspot.sg/2010/12/spring-3-mvc-using-modelattribute-in.html?showComment=1409074022476)

|  |  |
| --- | --- |
| 1  2  3  4  5 | @ModelAttribute("addresses")   public List<Address> getAllAddresses() {    // Delegate to service    return addressService.getAll();   } |

The model attribute **addresses** is referenced in the JSP page.  
  
**addressespage.jsp**

[?](http://krams915.blogspot.sg/2010/12/spring-3-mvc-using-modelattribute-in.html?showComment=1409074022476)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | <%@ taglib uri="<http://java.sun.com/jsp/jstl/core>" prefix="c" %>  <%@ page language="java" contentType="text/html; charset=UTF-8"      pageEncoding="UTF-8"%>  <!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>Insert title here</title>  </head>  <body>    <h1>Addresses</h1>  <table>   <tr>    <td width="100">Id</td>    <td width="150">Street</td>    <td width="150">City</td>    <td width="150">Zip Code</td>   </tr>   <c:forEach items="${addresses}" var="address">    <tr>     <td><c:out value="${address.id}" /></td>     <td><c:out value="${address.street}" /></td>     <td><c:out value="${address.city}" /></td>     <td><c:out value="${address.zipCode}" /></td>    </tr>   </c:forEach>  </table>  <p>${greetings}</p></body>  </html> |

In this controller we have to mappings:

/address/list1

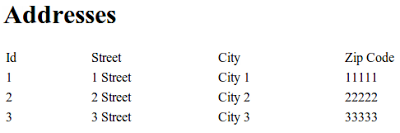
/address/list2

Both mappings will yield the same result. However the models are retrieved in different ways.  
  
The **getAllUsingModelAttribute()** method doesn't pass any model but instead relies on the @ModelAttribute for its data

[?](http://krams915.blogspot.sg/2010/12/spring-3-mvc-using-modelattribute-in.html?showComment=1409074022476)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | @RequestMapping(value="list1", method = RequestMethod.GET)     public String getAllUsingModelAttribute() {      logger.debug("Received request to show all addresses page");        // No need to add the model here      // It has been automatically added when we used the @ModelAttribute annotation earlier      // The name of the ModelAttribute is "addresses". Your JSP should reference "addresses"        // This will resolve to /WEB-INF/jsp/addressespage.jsp      return "addressespage";  } |

Here's the actual screenshot of the JSP page from /address/list1

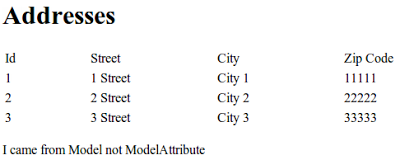
[](http://1.bp.blogspot.com/_hHhKKxOZVSc/TRhEBS4x8sI/AAAAAAAAATA/y1h56a3W8BY/s1600/list1.png)

The **getAllUsingModel()** method retrieves the model by manually adding it on the Model. It also adds an extra model data named *greetings*

[?](http://krams915.blogspot.sg/2010/12/spring-3-mvc-using-modelattribute-in.html?showComment=1409074022476)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | @RequestMapping(value="list2", method = RequestMethod.GET)     public String getAllUsingModel(Model model) {      logger.debug("Received request to show all addresses page");        // Here we add the model manually      // This should give the same result with the extra greetings      // The name of the Model is "addresses". Your JSP should reference "addresses" as well      model.addAttribute("addresses", addressService.getAll());      model.addAttribute("greetings", "I came from Model not ModelAttribute");        // This will resolve to /WEB-INF/jsp/addressespage.jsp      return "addressespage";  } |

Here's the actual screenshot of the JSP page from /address/list2

[](http://1.bp.blogspot.com/_hHhKKxOZVSc/TRhEHKtCO8I/AAAAAAAAATI/77um_txBdzg/s1600/list2.png)

They both produce the same list of addresses. So what's the difference? I don't see any except for one major difference:

@ModelAttribute annotated methods are executed ***before*** the chosen @RequestMapping annotated handler method. They effectively pre-populate the implicit model with specific attributes, often loaded from a database.   
  
Source: [Spring 3 Reference Documentation](http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/mvc.html#mvc-ann-modelattrib)

Using the @ModelAttribute automatically prepopulates your list. If you're doing some updates in your controller, you might still get the old list unless you specifically override the list from the @ModelAttribute.   
  
This is better explained when we move to the next usage pattern.  
  
**Pattern 2: Method Parameter Level**

When you place it on a method parameter, @ModelAttribute maps a model attribute to the specific, annotated method parameter (see the processSubmit() method below). This is how the controller gets a reference to the object holding the data entered in the form.rameters in the chosen handler method, potentially with binding and validation applied to it.  
  
Source: [Spring 3 Reference Documentation](http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/mvc.html#mvc-ann-modelattrib)

This means we have a method with a parameter and we annotate that parameter with a @ModelAttribute. The purpose of this pattern is to pass the data from the JSP page back to your controller. The data is automatically converted to a Java object.  
  
We will examine this pattern by displaying a list of persons that we can edit and update. This example will also showcase some of Spring 3's RESTful annotations.   
  
We begin by defining our main controller.  
  
**MainController.**

[?](http://krams915.blogspot.sg/2010/12/spring-3-mvc-using-modelattribute-in.html?showComment=1409074022476)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145 | package org.krams.tutorial.controller;    import java.util.ArrayList;  import java.util.List;    import javax.annotation.Resource;    import org.apache.log4j.Logger;  import org.krams.tutorial.domain.Person;  import org.krams.tutorial.service.PersonService;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.ModelAttribute;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;    /\*\*   \* Handles and retrieves Person related requests   \*/  @Controller  @RequestMapping("/main")  public class MainController {     protected static Logger logger = Logger.getLogger("controller");     @Resource(name="personService")   private PersonService personService;     /\*\*    \* Retrieves all persons and allows them to be used as a model    \* @return a model attribute containing persons    \*/   @ModelAttribute("persons")   public List<Person> getAllPersons() {    logger.debug("Retrieving all persons and adding it to ModelAttribute");      // Delegate to PersonService    return personService.getAll();   }     /\*\*    \* Retrieves all currency types    \* @return    \*/   @ModelAttribute("currencies")   public List<String> getAllCurrencies() {    logger.debug("Retrieving all currencies and adding it to ModelAttribute");      // Prepare data    List<String> currencies = new ArrayList<String>();    currencies.add("Dollar");    currencies.add("Yen");    currencies.add("Pound");    currencies.add("Euro");    currencies.add("Dinar");      return currencies;   }     /\*\*    \* Handles and retrieves a JSP page containing all persons    \*    \* @return the name of the JSP page    \*/      @RequestMapping(method = RequestMethod.GET)      public String getAllPage(Model model) {       logger.debug("Received request to show all persons page");         // The personspage.jsp referecences a model attribute named "persons"       // We don't need to add the model here manually       // It has been automatically added when we used @ModelAttribute("persons") earlier         // This will resolve to /WEB-INF/jsp/personspage.jsp       return "personspage";   }        /\*\*       \* Retrieves the edit page       \*       \* @return the name of the JSP page       \*/      @RequestMapping(value = "/edit/{id}", method = RequestMethod.GET)      public String getEdit(@PathVariable Integer id, Model model) {       logger.debug("Received request to show edit page");         // Retrieve person with matching id then add this person to the model       // The editpage.jsp references a model attribute named "personAttribute"       // So we add a "personAttribute" to the model.       // This "personAttribute" will be referenced again once we send the update form data       // We could have chosen a different name like "person" for the model       // If you do, make sure you update the JSP that references this name       // And update the POST method below that receives the request to do the actual update!       model.addAttribute("personAttribute", personService.get(id));         // The editpage.jsp references a model attribute named "currencies"       // This model attribute is passed automatically when used @ModelAttribute("currencies") earlier         // This will resolve to /WEB-INF/jsp/editpage.jsp       return "editpage";   }        /\*\*       \* Saves the edited person and display all persons again       \* @return       \*/      @RequestMapping(value = "/edit/{id}", method = RequestMethod.POST)      public String saveEdit(@ModelAttribute("personAttribute") Person person,        @PathVariable Integer id, Model model) {       logger.debug("Received request to update person");         // The "personAttribute" model has been passed to the controller from the JSP       // We use the name "personAttribute" because the JSP uses that name         // We manually assign the id because we disabled it in the JSP page       // When a field is disabled it will not be included in the ModelAttribute       person.setId(id);         // Delegate to PersonService for editing       // We show the all persons page again after updating the person       personService.edit(person);         // The personspage.jsp referecences a model attribute named "persons"       // We don't need to add the model here manually       // It has been automatically added when we used @ModelAttribute("persons") earlier         // However there's are two problems:         // 1. The @ModelAttribute("persons") is called first before the actual update is performed.       // When the personspage is returned, the list of persons BEFORE       // the update is the list that will be retrieved.       // You will need to refresh the browser again to see the new list         // 2. You can try redirecting but you'll hit a known bug       // See <http://stackoverflow.com/questions/2163517/how-do-i-prevent-spring-3-0-mvc-modelattribute-variables-from-appearing-in-url>         // To avoid all that hassles, we just add the updated list of persons to the Model       // The "persons" attribute here is not the same with @ModelAttribute("persons")       // However, your JSP page references the same name so it won't care       model.addAttribute("persons", personService.getAll());         // This will resolve to /WEB-INF/jsp/personspage.jsp       return "personspage";   }  } |

This controller declares two @ModelAttribute at the method level. We already know what that does. Then there's a @ModelAttribute at the method parameter level:

[?](http://krams915.blogspot.sg/2010/12/spring-3-mvc-using-modelattribute-in.html?showComment=1409074022476)

|  |  |
| --- | --- |
| 1  2  3  4  5 | @RequestMapping(value = "/edit/{id}", method = RequestMethod.POST)      public String saveEdit(@ModelAttribute("personAttribute") Person person,        @PathVariable Integer id, Model model) {  ...  } |

This basically means your JSP page is sending a model attribute named *personAttribute*. Your controller picks this model attribute and assign it to a Java object *Person*. In this way you're manipulating an object instead of HTTP request parameters.  
  
In this controller we have three mappings:

/main - retrieve all persons

/main/edit/{id} - (GET) retrieve and edit a person by his id

/main/edit/{id} - (POST) save a person based on his id

Notice we have two /main/edit/{id}. How does our controller know which one to call? The controller's @RequestMapping doesn't just rely on the mapping value but it also uses the method type. In our case, it's either POST or GET. The GET method is used when we retrieve a page; whereas, the POST method is used when we're submitting a form. For more info, please check the following blog from SpringSource[Annotated Web MVC Controllers in Spring 2.5](http://blog.springsource.com/2007/11/14/annotated-web-mvc-controllers-in-spring-25/)  
  
Also, we're using a special identifier in the mappings. We have declared a **{id}** in the path, and referenced that as @PathVariable in the method parameter. This is a URI template, one of the RESTful features of Spring 3 MVC.   
  
**What is a URI template?**

A URI template is a URI-like string, containing one or more variable names. When these variables are substituted for values, the template becomes a URI. For more information, see the proposed RFC.  
  
Source: [REST in Spring 3: @MVC](http://blog.springsource.com/2009/03/08/rest-in-spring-3-mvc/)

For a thorough description of this subject, please visit the blog from SpringSource[REST in Spring 3: @MVC](http://blog.springsource.com/2009/03/08/rest-in-spring-3-mvc/)  
  
Let's examine the associate JSP view for each mappings.  
  
**personspage.jsp**

[?](http://krams915.blogspot.sg/2010/12/spring-3-mvc-using-modelattribute-in.html?showComment=1409074022476)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | <%@ taglib uri="<http://java.sun.com/jsp/jstl/core>" prefix="c" %>  <%@ page language="java" contentType="text/html; charset=UTF-8"      pageEncoding="UTF-8"%>  <!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>Insert title here</title>  </head>  <body>    <h1>Persons</h1>  <table>   <tr>    <td width="50">Id</td>    <td width="150">First Name</td>    <td width="150">Last Name</td>    <td width="100">Money</td>    <td width="50">Currency</td>   </tr>   <c:forEach items="${persons}" var="person">    <tr>     <td><c:out value="${person.id}" /></td>     <td><c:out value="${person.firstName}" /></td>     <td><c:out value="${person.lastName}" /></td>     <td><c:out value="${person.money}" /></td>     <td><c:out value="${person.currency}" /></td>    </tr>   </c:forEach>  </table>  </body>  </html> |

This is referenced by the mapping /main. There's nothing special here. This is exactly similar with our Address example earlier.  
  
**editpage.jsp**

[?](http://krams915.blogspot.sg/2010/12/spring-3-mvc-using-modelattribute-in.html?showComment=1409074022476)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | <%@ taglib uri="<http://java.sun.com/jsp/jstl/core>" prefix="c" %>  <%@ taglib uri="<http://www.springframework.org/tags/form>" prefix="form" %>  <%@ page language="java" contentType="text/html; charset=UTF-8"      pageEncoding="UTF-8"%>  <!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>Insert title here</title>  </head>  <body>    <h1>Edit Person</h1>  <c:url var="saveUrl" value="/krams/main/edit/${personAttribute.id}" />  <form:form modelAttribute="personAttribute" method="POST" action="${saveUrl}">   <table>    <tr>     <td><form:label path="id">Id:</form:label></td>     <td><form:input path="id" disabled="true"/></td>    </tr>      <tr>     <td><form:label path="firstName">First Name:</form:label></td>     <td><form:input path="firstName"/></td>    </tr>      <tr>     <td><form:label path="lastName">Last Name</form:label></td>     <td><form:input path="lastName"/></td>    </tr>      <tr>     <td><form:label path="money">Money</form:label></td>     <td><form:input path="money"/></td>    </tr>      <tr>     <td><form:label path="currency">Currency:</form:label></td>     <td><form:select path="currency"  items="${currencies}"/></td>    </tr>   </table>   <input type="submit" value="Save" />  </form:form>  </body>  </html> |

This JSP page is returned by the following controller method:

[?](http://krams915.blogspot.sg/2010/12/spring-3-mvc-using-modelattribute-in.html?showComment=1409074022476)

|  |  |
| --- | --- |
| 1  2  3  4 | @RequestMapping(value = "/edit/{id}", method = RequestMethod.GET)      public String getEdit(@PathVariable Integer id, Model model) {  ...  } |

This method searches a person based on his id. If found, a model attribute is added in the Model. The name of the attribute is *personAttribute*. This can be any name.  
  
To access the edit page, we need to manually type the following URL in the browser:

http://localhost:8080/spring-jsp-model-attribute/krams/main/edit/2

Just make sure to change the number to match the id that you want to edit.  
  
Here's the Edit page:

[](http://2.bp.blogspot.com/_hHhKKxOZVSc/TRhQHnjlKxI/AAAAAAAAATQ/0hbzHApX_vs/s1600/edit.png)

When we're done editing the person, we submit the form.

[?](http://krams915.blogspot.sg/2010/12/spring-3-mvc-using-modelattribute-in.html?showComment=1409074022476)

|  |  |
| --- | --- |
| 1  2 | <c:url var="saveUrl" value="/krams/main/edit/${personAttribute.id}" />  <form:form modelAttribute="personAttribute" method="POST" action="${saveUrl}"> |

Notice we use POST to submit the data. We also use the same model attribute name. We also pass the id in the action URL.  
  
Once the data is sent, it is picked by the controller and assigned to the saveEdit method.

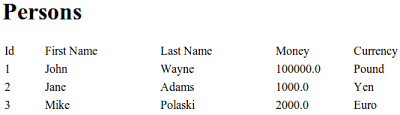
[?](http://krams915.blogspot.sg/2010/12/spring-3-mvc-using-modelattribute-in.html?showComment=1409074022476)

|  |  |
| --- | --- |
| 1  2  3  4  5 | @RequestMapping(value = "/edit/{id}", method = RequestMethod.POST)      public String saveEdit(@ModelAttribute("personAttribute") Person person,        @PathVariable Integer id, Model model) {  ...  } |

I advise my readers to read the important comments within this method. To display again the list of persons we can either pass the model manually or rely on the @ModelAttribute on the **Method Level**. However, there's a problem if you follow the Method Level. It doesn't show the updated list. You have to refresh the browser again to see it. Why?

**Note**  
@ModelAttribute annotated methods are executed ***before*** the chosen @RequestMapping annotated handler method. They effectively pre-populate the implicit model with specific attributes, often loaded from a database.   
  
Source: [Spring 3 Reference Documentation](http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/mvc.html#mvc-ann-modelattrib)

Because the list is populated even before it's processed by the method! As a workaround, you must pass the model manually so that you don't need to refresh the browser.  
  
Here's the final JSP page.

[](http://1.bp.blogspot.com/_hHhKKxOZVSc/TRhQNESa6aI/AAAAAAAAATY/1vbH8AEFRa0/s1600/persons-after.png)

Our application is now finished. We've managed to build a simple Spring 3 MVC application that uses JSPs for the presentation layer. We've also discussed the usage patterns of the @ModelAttribute.  
  
The best way to learn further is to try the actual application.  
  
**Download the project**  
You can access the project site at Google's Project Hosting at <http://code.google.com/p/spring-mvc-jsp-model-attribute/>  
  
You can download the project as a Maven build. Look for the [spring-jsp-model-attribute.zip](http://spring-mvc-jsp-model-attribute.googlecode.com/files/spring-jsp-model-attribute.zip) in the Download sections.  
  
You can run the project directly using an embedded server via Maven.  
For Tomcat: **mvn tomcat:run**  
For Jetty: **mvn jetty:run**