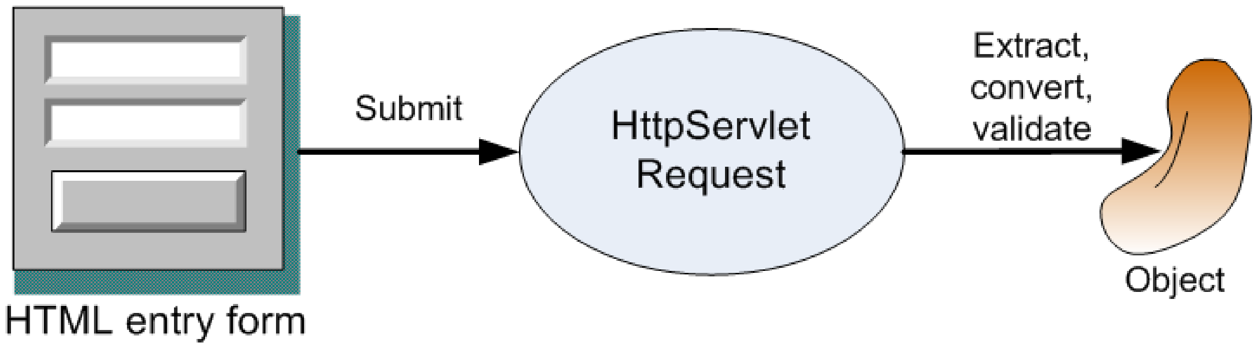
Spring Framework’s WebDataBinder

by Jim White | Feb 3, 2014

Last week, I was just outside our nation’s capital teaching Spring Web MVC Framework to a wonderful group of people working for the National Institute of Health (NIH). They are getting ready to move a collection of Struts 1 projects to Spring Web MVC. Some questions and discoveries around Spring Web MVC’s @InitBinder operations seemed a good fit for this week’s post.

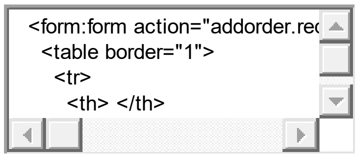
Spring Web MVC Command Beans

Part of my Spring MVC class is dedicated to teaching students how to have Spring bind data submitted by HTML form or query string parameters to a Java object (otherwise known as a Spring command bean or backing bean under this circumstance).



On the server, the data must be extracted out of the HttpServletRequest object, converted to a proper data format, loaded into an object, and validated. This can be done in code by the developer, but Spring comes with mechanisms to do this work for you automatically. The data, once in an object can be passed to the backend business components for appropriate processing.

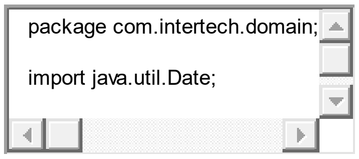
In order to have Spring collect and bind the data from an HTML form page (or query string parameter data), just create a plain old Java object/JavaBean with properties that match the request parameter names. For example, if collecting order information from an HTML form like that below…



```
<form:form action="addorder.rec
  <table border="1">
    <tr>
      <th> </th>
```

```
<form:form action="addorder.request" method="post" commandName="order">
  <table border="1">
    <tr>
      <th>&nbsp;</th>
      <th>Add Order</th>
    </tr>
    <tr>
      <td bgcolor="cyan">Customer:</td>
      <td><form:input path="customer" size="40" />
       <font color="#FF0000"><form:errors path="customer" /></font></td>
      </tr>
    <tr>
      <td bgcolor="cyan">Product:</td>
      <td><form:input path="product" size="40" />
        <font color="#FF0000"><form:errors path="product" /></font></td>
    </tr>
    <tr>
      <td bgcolor="cyan">Order date:</td>
      <td><form:input path="orderDate" size="40" />
       <font color="#FF0000"><form:errors path="orderDate" /></font></td>
    </tr>
    <tr>
      <td bgcolor="cyan">Ship date:</td>
      <td><form:input path="shipDate" size="40" />
       <font color="#FF0000"><form:errors path="shipDate" /></font></td>
    </tr>
    <tr>
      <td><input type="submit" value="Save" /></td>
      <td><input type="reset" value="Reset" /></td>
    </tr>
  </table>
  <a href="index.jsp">Home</a>
  <form:hidden path="id" />
</form:form>
```

… you would need a class like Order shown here.

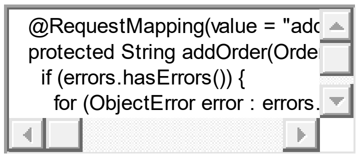


```
package com.intertech.domain;

import java.util.Date;
```

```
package com.intertech.domain;

import java.util.Date;

```

```
public class Order {

  private long id;
  private String customer;
  private String product;
  private Date orderDate;
  private Date shipDate;

  public long getId() {
    return id;
  }

  public void setId(long id) {
    this.id = id;
  }

  public String getCustomer() {
    return customer;
  }

  public void setCustomer(String customer) {
    this.customer = customer;
  }

  public String getProduct() {
    return product;
  }

  public void setProduct(String product) {
    this.product = product;
  }

  public Date getOrderDate() {
    return orderDate;
  }

  public void setOrderDate(Date orderDate) {
    this.orderDate = orderDate;
  }

  public Date getShipDate() {
    return shipDate;
  }

  public void setShipDate(Date shipDate) {
    this.shipDate = shipDate;
  }

  @Override
  public String toString() {
    return "Order [id=" + id + ", customer=" + customer + ", product="
    + product + ", orderDate=" + orderDate + ", shipDate="
    + shipDate + "]";
  }
}
```

Then a Spring controller automatically will bind request data to the properties in an instance of Order and pass it to the controller handler method. The addOrder( ) handler method shown here demonstrates how to write a handler method with a Command bean parameter.



```
@RequestMapping(value = "adc
protected String addOrder(Orde
  if (errors.hasErrors()) {
    for (ObjectError error : errors.
```

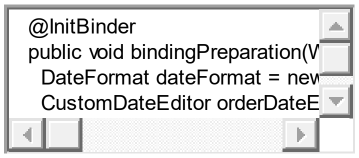
```
@RequestMapping(value = "addorder.request", method = RequestMethod.POST)
protected String addOrder(Order order, Errors errors) {
  if (errors.hasErrors()) {
    for (ObjectError error : errors.getAllErrors()) {
      System.out.println("Validation error: "
      + error.getDefaultMessage());
    }
  return "editorder";
  }
  // do the work of adding a new order
  return "successfuladd";
}
```

Spring Web MVC InitBinder

Spring automatically binds simple data (Strings, int, float, etc.) into properties of your command bean. However, what happens when the data is more complex? For example, what happens when you want to capture a String in “Jan 20, 1990” format and have Spring create a Date object from it as part of the binding operation. Perhaps you have custom types you want created from their string representation. For example, you want Spring to take a String in ###-###-#### format and populate a PhoneNumber type property you have in your Command bean.

For this work, you need to inform Spring Web MVC to use PropertyEditor instances as part of the binding process. The JavaBean API defines a java.beans.PropertyEditor interface. This interface defines methods to convert a property’s value to a String (getAsText()), and to set a property given a String (setAsText(String)). In particular, Spring Web MVC converts incoming request String data to the appropriate data type using PropertyEditors for the fields of the command beans.

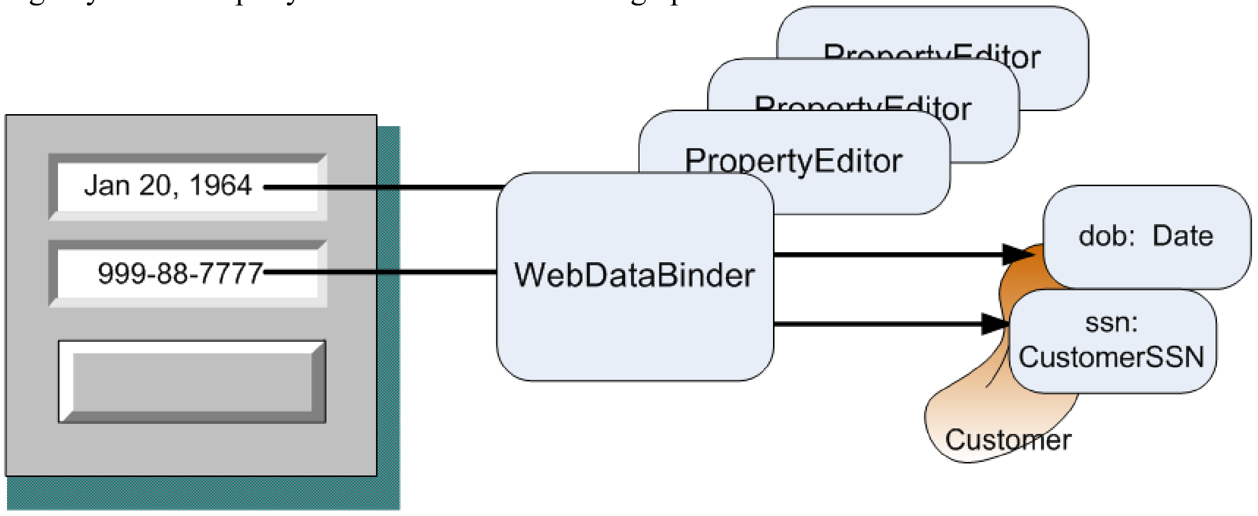
Some PropertyEditors are provided and used by Spring Web MVC by default (this is how simple data types are handled). How do you specify custom property editors for your command beans for the more complex typed fields? In a method of the controller annotated with @InitBinder, you register and configure your custom PropertyEditors for Spring Web MVC to use at the time of data binding. An example @InitBinder method is shown below to add a customized Date PropertyEditor to accept date strings in the format of "Jan 20, 1990" and convert these strings to java.util.Date objects for the Order object's fields. Note, the name of the annotated method is arbitrary. The important part of this controller method is the @InitBinder annotation on the method.



```
@InitBinder
public void bindingPreparation(V
  DateFormat dateFormat = new
  CustomDateEditor orderDateE
```

```
@InitBinder
public void bindingPreparation(WebDataBinder binder) {
  DateFormat dateFormat = new SimpleDateFormat("MMM d, YYYY");
  CustomDateEditor orderDateEditor = new CustomDateEditor(dateFormat, true);
  binder.registerCustomEditor(Date.class, orderDateEditor);
}
```

Note also the WebDataBinder (a subclass of the more generic DataBinder) parameter of the @InitBinder method. This object manages the Spring binding process and holds a registry of the PropertyEditors used in the binding operation.



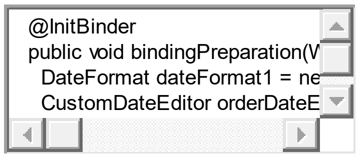
In this example, Order's orderDate and shipDate can be captured in "MMM d, YYYY" string format and be bound to java.util.Date objects by Spring.

Handling Two Properties of the Same Type

A problem arises when the binding operation must deal with two properties of the same type and String data for those properties is formatted differently. This is the problem three of my students (Nick, Erik and Nivedita) worked on and solved in last week (thanks guys!).

For example, if you note, there are two Date properties in the Order type above. Currently, the WebDataBinder has been given just one CustomDateEditor that uses the “MMM d, YYYY” format for accepting date Strings into both the shipDate and orderDate properties. So, under this design, all Date data would have to be provided in this format by the user in order for it to be accepted by Spring Web MVC for Orders. What if you have two different formats for Dates? For example, what if order dates are to be provided in “d-MM-yyyy” format and ship dates are to be provided in “MMM d, YYYY” format?

As Nick, Erik and Nivedita discovered, there is an overloaded registerCustomEditor( ) method in the DataBinder class. The registerCustomEditor( ) calls in the @InitBinder method above pass in the target class type and the PropertyEditor instance to be used to perform the binding of that type of property. Another registerCustomEditor method call allows you to specify the command bean field to which the custom PropertyEditor applies. This allows you to set up PropertyEditors per property/field versus per general property data type. Below, the @InitBinder method now contains the registration of two CustomDateEditors. One used for orderDates (in d-MM-yyyy” format) while the other is used for shipment dates (in “MMM d, YYYY”) format.



```
@InitBinder
public void bindingPreparation(V
  DateFormat dateFormat1 = ne
  CustomDateEditor orderDateE
```

```
@InitBinder
public void bindingPreparation(WebDataBinder binder) {
  DateFormat dateFormat1 = new SimpleDateFormat("d-MM-yyyy");
  CustomDateEditor orderDateEditor = new CustomDateEditor(dateFormat1, true);
  DateFormat dateFormat2 = new SimpleDateFormat("MMM d, YYYY");
  CustomDateEditor shipDateEditor = new CustomDateEditor(dateFormat2, true);
  binder.registerCustomEditor(Date.class, "orderDate", orderDateEditor);
  binder.registerCustomEditor(Date.class, "shipDate", shipDateEditor);
}
```