StrutsCatalogMultipleButtonSolutions

This presentation is somewhat detailed, so we begin with a Table of Contents for your ease in navigating the presentation. Enjoy! There is a comment section at the very end.

Table of Contents

Universal Button Solutions

There is a persistent problem in web applications in handling multiple buttons, and this is especially a problem when the buttons are images.

Struts has provided a solution for this with DispatchAction and its progeny. I have found this solution to be too heavy, too complicated, and too obtuse. So, I have engendered my own.

The principle reason for this class is the problem with <input type='image' name='whatever'> tags in HTML. Those tag send the values in the name attribute as *whatever.x=9* and *whatever.y=26*. DispatchAction and its progeny provide a melange of solutions but no common way to deal with *<a href='whatever.do'>*, *<input type='submit'>*, *<input type='image'>*, *<input type='file'>*, or whatever. This class, DispatchUtil does that. If you prefer to use a subclass, just provide the functionality in this class in a class extending Action. Herbert Rabago came up with the idea of providing the functionality in a utility class and I really like that idea. So, here we go.

Uses

N.B. the uses of "dispatch". If you do not like using "dispatch", then you can rewrite the code in the getMethodName(...) method. Essentially, you will need to replace ".dispatch" with something else in the tag names which you can identify in the enumeration. You have to use a suffix with this logic. But, if you want a prefix, e.g. "dispatch.update" or "method.update", then you will have to change the logic too. Suffix logic seems to best fit the existing methods in String. Otherwise, I would have used a prefix. You may want to anyway.

Action Class Code to Use

In all the following tag uses, you use the Action class code in execute(...) or process(...) as given directly below.

```
  ActionForward = forward = new DispatchUtil().dispatch(this,mapping,form,request,response);
```

Multiple Image Tags

Multiple image tags are the primary reason for this class. Integrating this into a common solution is the difficulty.

```
<input type='image' name='update.dispatch' src='update.gif'>
<input type='image' name='delete.dispatch' src='delete.gif'>
```

Link Tags

```
<a href='profile.do?update.dispatch=whatever'>
<a href='profile.do?delete.dispatch=whatever'>
```

Submit Tags

```
<input type='submit' name='update.dispatch' value='whatever'>
<input type='submit' name='delete.dispatch' value='whatever'>
```

File Browse Tags

```
<input type='file' name='update.dispatch'>
<input type='file' name='delete.dispatch'>
```

Button Tags

```
<input type='button' name='update.dispatch' value='whatever'>
<input type='button' name='delete.dispatch' value='whatever'>
```

struts-config.xml Uses

If you like to do things the way they are done with DispatchActions and use Struts xml for commands, then the following are possibilities. **Please note that you never have to do this. Struts xml configuration is wholly voluntary for this solution.** If you want the sort of behavior you get with MappingDispatchAction using DispatchUtil, then you do as we indicate. This does not mean, of course, that you have to use MappingDispatchAction. DispatchUtil completely absorbs all the functionality of the DispatchActions. You will see, further, that no struts-config.xml configuration is required at all for LookupDispatchActions.

MappingDispatchAction

This solution depends not on the value of the name attributes in the html tags, but strictly on the value of the path attributes in the html tags, anytime /updateProfile.do is called in the example below, any name attributes will be overridden and the value of the ActionMapping property called "parameter" will be operative instead.

{{{<action path="/updateProfile"
type="org.example.ProfileAction"
parameter="update.dispatch"
name="profileForm"
validate="true"
input="/editProfile.jsp"
scope="request">
<forward name="success" path="/updatedProfile.jsp"/>
</action>

<action path="/deleteSubscription"
type="org.example.ProfileAction"
name="profileForm"
scope="request"
input="/subscription.jsp"
parameter="delete.dispatch">
<forward name="success" path="/deletedSubscription.jsp"/>
</action>
}}}

LookupDispatchAction

This solution depends not on the value of the path attribute or the parameter property in the ActionMapping, notice that no parameter attribute exists below, but strictly on the values of the name attributes as indicated in our section on the tags. In other words, we don't have to configure struts-config.xml for what used to be LookupDispatchActions.

```
<action path="/updateProfile"
          type="org.example.ProfileAction"
          name="profileForm"
          validate="true"
          input="/editProfile.jsp"
          scope="request">
      <forward name="success" path="/updatedProfile.jsp"/>
  </action>

  <action path="/deleteSubscription"
          type="org.example.ProfileAction"
          name="profileForm"
          scope="request"
          input="/subscription.jsp"
      <forward name="success" path="/deletedSubscription.jsp"/>
  </action>
```

SOLUTON ONE: DispatchUtil Solution Code

```
package com.crackwillow.struts.util.dispatch;

import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.Enumeration;
import java.util.HashMap;

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

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.util.MessageResources;

public class DispatchUtil {
  protected static    Log              log      = LogFactory.getLog(DispatchUtil.class);
  protected static    MessageResources messages = MessageResources.getMessageResources ("org.apache.struts.
actions.LocalStrings");
  protected           HashMap          methods  = new HashMap();
  protected           Class []         types    = { ActionMapping.class,
                                                    ActionForm.class,
                                                    HttpServletRequest.class,
                                                    HttpServletResponse.class };

  public ActionForward dispatch(Action action,
                                ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
```

```
                            HttpServletResponse response)
                      throws Exception {
  String methodName  = getMethodName(request,mapping);
  Class  clazz       = action.getClass();
  if ("execute".equals(methodName) || "perform".equals(methodName)){
    // Prevent recursive calls
    String message = messages.getMessage("dispatch.recursive", mapping.getPath());
    log.error(message);
    throw new ServletException(message);
  }
  return dispatchMethod(action,clazz,mapping, form, request, response, methodName);
}

protected ActionForward dispatchMethod(Action action,
                                       Class clazz,
                                       ActionMapping mapping,
                                       ActionForm form,
                                       HttpServletRequest request,
                                       HttpServletResponse response,
                                       String name)
                      throws Exception {
  if (name == null) {
    return this.unspecified(mapping, form, request, response);
  }

  Method method = null;

  try {
    method = getMethod(clazz,name);
  } catch(NoSuchMethodException nsme) {
    String message = messages.getMessage("dispatch.method", mapping.getPath(), name);
    log.error(message, nsme);
    throw nsme;
  }

  ActionForward forward = null;

  try {
    Object args[] = { mapping, form, request, response };
    forward = (ActionForward)method.invoke(action, args);
  } catch(ClassCastException cce) {
    String message = messages.getMessage("dispatch.return", mapping.getPath(), name);
    log.error(message, cce);
    throw cce;
  } catch(IllegalAccessException iae) {
    String message = messages.getMessage("dispatch.error", mapping.getPath(), name);
    log.error(message, iae);
    throw iae;
  } catch(InvocationTargetException ite) {
    Throwable t = ite.getTargetException();
    if (t instanceof Exception) {
      throw ((Exception) t);
    } else {
      String message = messages.getMessage("dispatch.error", mapping.getPath(), name);
      log.error(message, ite);
      throw new ServletException(t);
    }
  }
  return (forward);
}

protected static String getMethodName(HttpServletRequest request,  ActionMapping mapping) {
  String methodName  = null;
  String buttonValue = null;
  String paramProperty = mapping.getParameter();
  if((paramProperty != null)) {
    methodName = paramProperty.substring(0,paramProperty.indexOf('.'));
  } else {
    Enumeration enum = request.getParameterNames();
    while(enum.hasMoreElements()) {
      buttonValue = (String)enum.nextElement();
```

```
        StdOut.log("log.dispatch","DispatchUtil buttonValue = " + buttonValue);
        if(buttonValue.indexOf(".dispatch") >= 0) {
          methodName = buttonValue;
          break;
        }
      }
    }
    return methodName.substring(0,methodName.indexOf('.'));
  }

  protected Method getMethod(Class clazz,String name)
      throws NoSuchMethodException {
    synchronized(methods) {
      Method method = (Method) methods.get(name);

      if (method == null) {
        method = clazz.getMethod(name, types);
        methods.put(name, method);
      }

      return (method);
    }
  }

  protected ActionForward unspecified(ActionMapping mapping,
                                      ActionForm form,
                                      HttpServletRequest request,
                                      HttpServletResponse response)
                       throws Exception {
    String message = messages.getMessage( "dispatch.parameter", mapping.getPath(), getMethodName(request,
mapping));
    log.error(message);

    throw new ServletException(message);
  }
}
```

(new) DispatchAction

Discussion

If, rather than having a utility class provide the functionality, you prefer an Action class. Here is a redoing of the DispatchAction class to let you do that. You use this class for **ALL** the progeny of DispatchAction class, i.e. this class replaces completely DispatchAction, LookupDispatchAction, and MappingDispatchAction. You can alter this class to get whatever subsidiary functionality you might want and which some might consider essential.

SOLUTION TWO: (new) DispatchAction Solution Code

```
package com.crackwillow.struts.action;

import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.Enumeration;
import java.util.HashMap;

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

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.util.MessageResources;

public abstract class DispatchAction
    extends Action {
```

```
  protected          Class           clazz    = this.getClass();
  protected static   Log             log      = LogFactory.getLog(SimpleDispatchAction.class);
  protected static   MessageResources messages = MessageResources.getMessageResources ("org.apache.struts.
actions.LocalStrings");
  protected          HashMap         methods  = new HashMap();
  protected          Class[]         types    = { ActionMapping.class,
                                                  ActionForm.class,
                                                  HttpServletRequest.class,
                                                  HttpServletResponse.class };

  public ActionForward execute(ActionMapping mapping,
                               ActionForm form,
                               HttpServletRequest request,
                               HttpServletResponse response)
      throws Exception {
    String name = getMethodName(request,mapping);

    if ("execute".equals(name) || "perform".equals(name)){
      // Prevent recursive calls
      String message = messages.getMessage("dispatch.recursive", mapping.getPath());
      log.error(message);
      throw new ServletException(message);
    }

    return dispatchMethod(mapping, form, request, response, name);
  }

  protected ActionForward dispatchMethod(ActionMapping mapping,
                                         ActionForm form,
                                         HttpServletRequest request,
                                         HttpServletResponse response,
                                         String name)
      throws Exception {

    if (name == null) {
      return this.unspecified(mapping, form, request, response);
    }

    Method method = null;

    try {
      method = getMethod(name);
    } catch(NoSuchMethodException nsme) {
      String message = messages.getMessage("dispatch.method", mapping.getPath(), name);
      log.error(message, nsme);
      throw nsme;
    }

    ActionForward forward = null;

    try {
      Object args[] = {mapping, form, request, response};
      forward = (ActionForward) method.invoke(this, args);
    } catch(ClassCastException cce) {
      String message = messages.getMessage("dispatch.return", mapping.getPath(), name);
      log.error(message, cce);
      throw cce;
    } catch(IllegalAccessException iae) {
      String message = messages.getMessage("dispatch.error", mapping.getPath(), name);
      log.error(message, iae);
      throw iae;

    } catch(InvocationTargetException ite) {
      Throwable t = ite.getTargetException();

      if (t instanceof Exception) {
        throw ((Exception) t);
      } else {
        String message = messages.getMessage("dispatch.error", mapping.getPath(), name);
        log.error(message, ite);
        throw new ServletException(t);
```

```
      }
    }
    return (forward);
  }

  protected static String getMethodName(HttpServletRequest request,  ActionMapping mapping) {
    String methodName  = null;
    String buttonValue = null;
    String paramProperty = mapping.getParameter();
    if((paramProperty != null)) {
      methodName = paramProperty.substring(0,paramProperty.indexOf('.'));
    } else {
      Enumeration enum = request.getParameterNames();
      while(enum.hasMoreElements()) {
        buttonValue = (String)enum.nextElement();
        StdOut.log("log.dispatch","DispatchUtil buttonValue = " + buttonValue);
        if(buttonValue.indexOf(".dispatch") >= 0) {
          methodName = buttonValue;
          break;
        }
      }
    }
    return methodName.substring(0,methodName.indexOf('.'));
  }

  protected Method getMethod(String name)
      throws NoSuchMethodException {
    synchronized(methods) {
      Method method = (Method) methods.get(name);

      if (method == null) {
        method = clazz.getMethod(name, types);
        methods.put(name, method);
      }

      return (method);
    }
  }

  protected ActionForward unspecified(ActionMapping mapping,
                                      ActionForm form,
                                      HttpServletRequest request,
                                      HttpServletResponse response)
      throws Exception {
    String message = messages.getMessage( "dispatch.parameter", mapping.getPath(), getMethodName(request,
mapping));
    log.error(message);

    throw new ServletException(message);
  }

  protected ActionForward cancelled(ActionMapping mapping,
                                    ActionForm form,
                                    HttpServletRequest request,
                                    HttpServletResponse response)
      throws Exception {

    return null;
  }
}
```

POJO: Solution for Tags without Dispatch, without Reflection, without ActionMapping or Even without Struts

Discussion

You can use the same strategy to determine what button tag has been clicked without employing Struts. The best way to do this, in my opinion, in the following solution.

SOLUTION THREE: ButtonTagUtil Solution Code

```
public class ButtonTagUtil {
  public static String getName(HttpServletRequest request) {
    String methodName  = null;
    String buttonValue = null;
    Enumeration enum = request.getParameterNames();
    while(enum.hasMoreElements()) {
      buttonValue = (String)enum.nextElement();
      if(buttonValue.indexOf(".dispatch") >= 0) {
        methodName = buttonValue;
        break;
      }
    }
    return methodName.substring(0,methodName.indexOf('.'));
  }
}
```

ButtonForm

Note that this solution does not use ".dispatch" and is not tied to Struts

```
<input type='image' name='update' src='update.gif'>
<input type='image' name='delete' src='delete.gif'>
```

Discussion

This solution is for <input type='image'> only. The other cases, of course, do not pose special problems with determining which button was clicked. This solution is superior to the Button solution which follows because only one button object has to be created. If you use buttons for navigation extensively within large forms, this because crucial.

SOLUTION FOUR: ButtonForm Code

```
public class ButtonForm
    extends ActionForm {
  protected CrackWillowButton button;
  protected String command;

  public CrackWillowButton getButton() {
    if(button == null) { this.button = new CrackWillowButton(); }
    return button;
  }

  public String getCommand() {
    String hold = command; command = null;
    return hold;
  }

  public void reset() {
    button = null;
  }

  public class CrackWillowButton {
    private Integer x, y;
    public CrackWillowButton() { }
    public CrackWillowButton getSubmit() { command = "submit";    return button; }
    public CrackWillowButton getClear()  { command = "clear";    return button; }

    public void setX(Integer x) {
      if(y != null) { reset(); }
      else          { this.x = x; }
    }

    public void setY(Integer y) {
      if(x != null) { reset(); }
      else          { this.y = y; }
    }
  }
} ///;-)
```

Button

This solution, too, is only for <input type='image'>

Discussion

This solution is as objectionable to me as (old) DispatchAction in Struts and its progeny, MappingDispatchAction and LookupDispatchAction. However, it is the first solution I proposed on this wiki, so I am putting it in here as a quasi-historical brain matter for those with quasi-hysterical brain matter.

SOLUTION FIVE: Button Code

First we have the button:

```
public final class ButtonCommand {
  private ButtonCommand() {
  }

  public final static String getCommand(HttpServletRequest request) {
    Enumeration enum = request.getParameterNames();
    String parameterName = null;
    while(enum.hasMoreElements()) {
      parameterName = (String)enum.nextElement();
      if(parameterName.endsWith(".x")) {
        return parameterName.substring(0,parameterName.indexOf('.'));
      }
    }
    return parameterName;
  }
}
```

Then we have the form:

```
public class ButtonForm
    extends ActionForm {
  protected CrackWillowButton button;
  protected String command;

  public CrackWillowButton getButton() {
    if(button == null) { this.button = new CrackWillowButton(); }
    return button;
  }

  public String getCommand() {
    String hold = command; command = null;
    return hold;
  }

  public void reset() {
    button = null;
  }

  public class CrackWillowButton {
    private Integer x, y;
    public CrackWillowButton() { }
    public CrackWillowButton getSubmit() { command = "submit";    return button; }
    public CrackWillowButton getClear()  { command = "clear";    return button; }

    public void setX(Integer x) {
      if(y != null) { reset(); }
      else          { this.x = x; }
    }

    public void setY(Integer y) {
      if(x != null) { reset(); }
      else          { this.y = y; }
    }
  }
} ///;-)
```

In the Action class, we check out which button type has been pressed as follows:

```
  String command = buttonForm.getCommand();
  if("submit".equals(command)) {
    // do whatever
  } else if ("clear".equals(command)) {
    // do whatever
  } else {
    // some assert code
  }
```

Links

StrutsCatalogInstrumentableForms

StrutsCatalogMappedBeans

StrutsCatalogHidingImagesAndOtherResourcesUnderWEBINF

Comments

CommentPageForCritiques

Thank you for using the above link for comments instead of this page, since there is no easy way to accommodate everyone's style in one place.

AUTHOR

If you have any questions contact me at mike@michaelmcgrady.com .

Michael McGrady

If your comments are extensive, you might want to provide a link instead of adding to this already incredibly verbose presentation.