



Struts Architecture

The Struts MVC Design



The Struts Framework

- Struts implements a MVC infrastructure on top of J2EE
 - One Servlet acts as the 'Front Controller'
 - Base classes are provided for value objects and controllers
 - Tag libraries simplify interface design
 - Request dispatching and error handling is standardised
- Applications can be built faster and are less fragile
 - Struts versions 1.0 and 1.1 are in widespread commercial use
 - Major changes occurred between the two versions
 - Version 1.2 is the latest release and gradually being adopted
 - Many in-house frameworks are cut down versions of Struts
 - There are several simplified alternatives to Struts



The Struts Architecture

- In Struts applications one Servlet handles all requests
 - This is an example of the 'Front Controller' pattern
 - The Servlet receiving the requests is of type 'ActionServlet'
 - Located in the package 'org.apache.struts.action'
- The Action Servlet must be configured in 'web.xml'
 - A wildcard is used to direct all requests to the Servlet
 - The Servlet takes several initialization parameters
 - The most important of which is the location of a struts specific configuration file (usually given the name 'struts-config.xml')



Configuring the Action Servlet

```
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
  <init-param>
    <param-name>config</param-name>
    <param-value>/WEB-INF/struts-config.xml</param-value>
  </init-param>
  <init-param>
    <param-name>debug</param-name>
    <param-value>0</param-value>
  </init-param>
  <load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>/actions/*</url-pattern>
</servlet-mapping>
```



The Struts Architecture

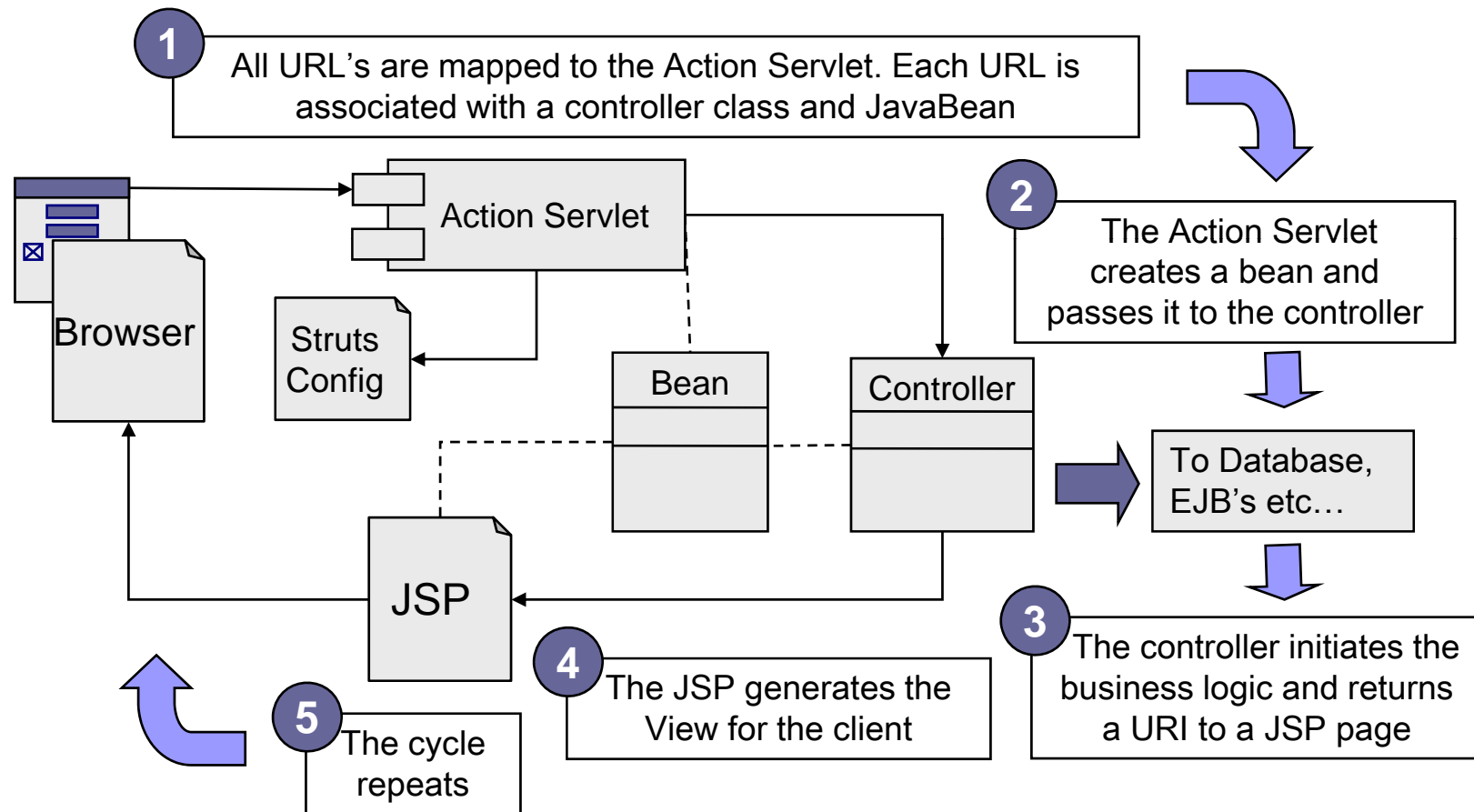
- When a request arrives its URL is inspected
 - 'struts-config.xml' associates each URL with an 'Action' class
 - This is the controller part of the Struts MVC design
 - However it is designed to be a bridge to non presentation layer components, such as ELB's
 - Each action class is in turn associated with an 'ActionForm'
 - This is a JavaBean that acts as a Model 2 value object
 - It provides a convenient container for parameters
 - It is not designed to be a permanent data object but just a vehicle by which parameters can be encapsulated and transported



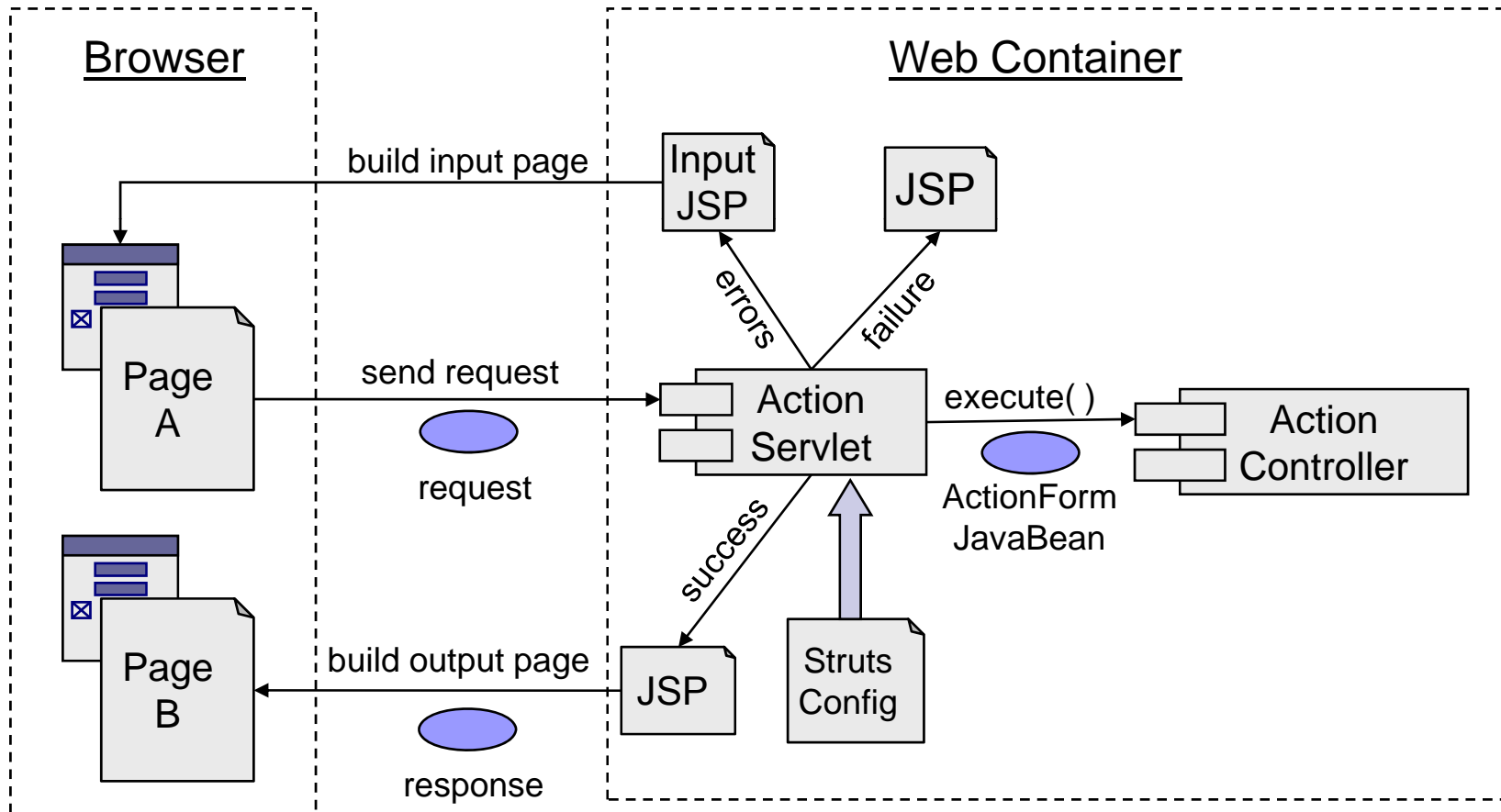
The Struts Architecture

- When a new request arrives:
 1. The 'ActionServlet' determines the associated Action (Controller) and Action Form (Value Object)
 2. An Action Form object is created and the request parameters are stored inside it (the bean can also be validated)
 3. An Action object is created and its 'execute' method is called (the Action Form object is passed as a parameter)
 4. The Action object triggers the business logic and returns the URL of the JSP that will produce the output
 5. The request is forwarded to the JSP, along with the Action Form
 6. The JSP generates the HTTP response for the client
- This cycle is the 'heartbeat' of a Struts application
 - The mechanism for error handling complicates it slightly...

The Struts MVC Architecture



The Struts Interpretation of MVC





Creating a Struts Config File

- The structure of the file is controlled by a DTD
 - Lots of info can be included but only a small amount is required
- There are two essential components
 - 'form-beans' contains a collection of 'form-bean' elements
 - Each of which describes an Action Form value object
 - The 'name' attribute gives a label which identifies it elsewhere
 - The 'type' attribute gives the fully qualified class name
 - 'action-mappings' contains a collection of 'action' elements
 - Each of which describes a Action controller class
 - The 'type' attribute gives the fully qualified class name
 - The 'name' attribute is the name of the associated Action Form
 - The 'path' attribute gives the URL that will trigger this action



Creating a Struts Config File

```
<?xml version="1.0" encoding="UTF-8"?>
<struts-config>
  <form-beans>
    <form-bean name="sampleForm" type="demos.struts.SampleActionForm"/>
    <form-bean name="flight" type="demos.struts.FlightForm"/>
    <form-bean name="booking" type="demos.struts.BookingForm"/>
  </form-beans>
  <action-mappings>
    <action path="/sample" type="demos.struts.SampleAction"
      name="sampleForm" scope="request"/>
    <action path="/travel" type="demos.struts.TravelAction"
      name="flight" scope="request"/>
    <action path="/booking" type="demos.struts.BookingAction"
      name="booking" scope="request"/>
  </action-mappings>
</struts-config>
```



The Struts Architecture

- Most people find Struts confusing at first
 - The configuration file can be especially difficult
 - Remember that the 'name' attribute of '<action>' refers to the name of the associated '<form-bean>'

Name	Config Element	Role
Action class	<action>	Controller (Facade)
Action Form class	<form-bean>	Value Object



Changes From Struts 1.0 to 1.1

- In Struts 1.0 the Action Servlet was the controller
 - It managed the different stages of processing a request
 - The only way to modify the cycle was to extend 'ActionServlet'
- Struts 1.1 introduced the 'RequestProcessor'
 - This class was refactored out of 'ActionServlet'
 - The Action Servlet now delegates the processing of requests
- You can configure the 'RequestProcessor'
 - Using an element called 'controller' inside 'struts-config.xml'
 - This has no content but many attributes
- You can create your own request processor
 - In order to customize how the Struts 'heartbeat'



Configuring the Request Processor

Controller Attribute	Meaning
bufferSize	Input buffer size for file uploads
className	Bean class which holds the controller configuration
contentType	Default content type (may be overridden in a JSP)
debug	Debugging level (from zero upwards)
forwardPattern	Controls how paths in forward elements are interpreted
inputForward	Controls how inputs in action elements are interpreted
locale	Stores the users preferred locale in the session object
maxFileSize	Maximum size of a file upload
multipartClass	The name of the multipart request handler class
nocache	If true a 'nocache' HTTP header is sent in every response
pagePattern	Controls how page attributes of custom tags are interpreted
processorClass	The class which will be the Request Processor
tempDir	Specifies the working directory for file uploads



Changes From Struts 1.0 to 1.1

- Struts 1.1 uses the Jakarta Commons subproject
 - A set of libraries for functionality found in most Java apps
 - This replaces bespoke code in the original release
- Hence Struts 1.1 is dependant on:
 - Commons BeanUtils (manipulating JavaBeans)
 - Commons Digester (processing XML configuration files)
 - Commons Collections (enhancements of 'java.util' collections)
 - Commons Logging (abstracts Log4J, JDK logging etc...)
- Learning about these libraries is a very good idea
 - Your debugging may take you into them
 - You may find them useful in your own developments



Creating Action Classes

- All Action classes extend the 'Action' class
 - Found in the package 'org.apache.struts.action'
- Action objects are multithreaded
 - Only a single instance of each Action is created
 - As with Servlets your code must be thread safe
- You only need to override a single business method
 - Which will be called by the Request Processor
 - In Struts 1.0 this method is called 'perform'
 - In 1.1 this method is deprecated in favour of 'execute'
 - The change was due to enhancements to exception handling
 - Otherwise the signature of the methods is the same



Creating Action Classes

```
public ActionForward perform (ActionMapping mapping,  
                                ActionForm form,  
                                HttpServletRequest request,  
                                HttpServletResponse response)  
    throws java.io.IOException,  
           javax.servlet.ServletException {}
```

```
public ActionForward execute (ActionMapping mapping,  
                                ActionForm form,  
                                HttpServletRequest request,  
                                HttpServletResponse response)  
    throws Exception {}
```



Creating Action Classes

- Action classes trigger a single business process
 - Usually they don't contain business logic themselves
 - Instead they are a façade to your business components
 - These could be POJO's, Enterprise Beans, Web Services etc...
- The 'ActionForm' is passed as a parameter
 - So you can extract the typed and validated information
- The request and response objects are also passed
 - So you can also manipulate the current context
- The execute method returns an 'ActionForward' object
 - This represents the resource that will generate the response



Using Action Forwards

- 'ActionForward' objects can be created in two ways
 - By manually creating a new instance
 - Passing the destination URL as a parameter
 - By using the 'ActionMapping' parameter
 - This holds the predefined mappings from 'struts-config.xml'
 - The second option avoids hard coding URL's
- Mappings can be both local and global
 - Local mappings apply only to one action
 - They appear as '<forward>' elements inside '<action>'
 - Global mappings can be used with all actions
 - They appear as '<forward>' elements inside '<global-forwards>'



Action Forward Mappings

```
<?xml version="1.0" encoding="UTF-8"?>
<struts-config>
  <global-forwards>
    <forward name="exit" path="/exit.jsp"/>
  </global-forwards>
  <action-mappings>
    <action path="/sample" type="demos.struts.SampleAction"
      name="sampleForm" scope="request">
      <forward name="result" path="/xx/displayResult.jsp"/>
    </action>
  </action-mappings>
</struts-config>
```



Custom Actions

- Struts can assume too much complexity
 - Such as requiring separate action classes for each URL
 - Often we want to combine or eliminate actions
 - Where the logic is just simple request dispatching
 - Where we want to group related business logic together
- Struts includes seven helper Action classes
 - All are found in the package 'org.apache.struts.actions'
- Two actions automatically perform request dispatching
 - These are 'ForwardAction' and 'IncludeAction'
 - The target URL is provided in 'struts-config.xml'
 - This prevents writing trivial actions or bypassing Struts



Custom Actions

- Using 'DispatchAction' a request can trigger one of many business methods from the same action object
 - Via a request parameter holding the method name
 - The parameter name is configured in 'struts-config.xml'
 - A disadvantage is that the client then knows the method name
 - Each method must have the same parameters as 'perform'
- Using 'MappingDispatchAction' the business methods can be bound to separate URL's
 - The mapping is configured in 'struts-config.xml'
 - This class was introduced in Struts 1.2
 - You could create your own version using the Reflection API



Using Dispatch Action

```
<a href="/webapp/actions/dispatchDemo?methodName=op1">Call op1</a>
<a href="/webapp/actions/dispatchDemo?methodName=op2">Call op2</a>
<a href="/webapp/actions/dispatchDemo?methodName=op3">Call op3</a>
```

```
<action path="/dispatchDemo" type="actions.DispatchActionDemo" parameter="methodName"/>
```

```
public class DispatchActionDemo extends DispatchAction {
    public ActionForward op1(ActionMapping m,ActionForm f,HttpServletRequest req,
        HttpServletResponse resp) throws Exception {
        //Implementation omitted
    }
    public ActionForward op2(ActionMapping m,ActionForm f,HttpServletRequest req,
        HttpServletResponse resp) throws Exception {
        //Implementation omitted
    }
    public ActionForward op3(ActionMapping m,ActionForm f,HttpServletRequest req,
        HttpServletResponse resp) throws Exception {
        //Implementation omitted
    }
}
```



Using Mapping Dispatch Action

```
<action path="/callOp1" type="MappingDispatchActionDemo" parameter="op1"/>
<action path="/callOp2" type="MappingDispatchActionDemo" parameter="op2"/>
<action path="/callOp3" type="MappingDispatchActionDemo" parameter="op3"/>
```

```
public class MappingDispatchActionDemo extends MappingDispatchAction {
    public ActionForward op1(ActionMapping m,ActionForm f, HttpServletRequest req,
        HttpServletResponse resp) throws Exception {
        //Implementation omitted
    }
    public ActionForward op2(ActionMapping m,ActionForm f,HttpServletRequest req,
        HttpServletResponse resp) throws Exception {
        //Implementation omitted
    }
    public ActionForward op3(ActionMapping m,ActionForm f,HttpServletRequest req,
        HttpServletResponse resp) throws Exception {
        //Implementation omitted
    }
}
```



Summary of Custom Actions

Action Class	Description
ForwardAction	Forwards the request to a URL specified in 'struts-config.xml'
IncludeAction	Includes the output of a URL specified in 'struts-config.xml'
DispatchAction	Actions inheriting from this class contain many business methods. When clients call the action they can specify a method via a request parameter
MappingDispatchAction	Similar to DispatchAction except that many URL's link to your Action and the business method to be called is determined by (mapped to) the URL
LookupDispatchAction	Used for forms with multiple submit buttons with the same name
SwitchAction	Switches control to a different Struts application module
LocaleAction	Changes the Locale used by Struts for the current user



Creating Action Form Classes

- To create an Action Form class:
 1. Create a class which extends 'ActionForm'
 - From the package 'org.apache.struts.action'
 2. Create accessor methods for each request parameter
 - The bean properties will be initialized by the Request Processor in the same way as the 'jsp:setProperty' action
 3. Implement the 'reset' method to re-initialize your bean
 - This enables the framework to reuse bean objects
 4. Optionally implement the 'validate' method
 - This is called to make sure the request data is consistent
 - If all is OK return 'null' or an empty 'ActionErrors' object



Dynamic ActionForms

- Creating ActionForms can be tedious
 - Many classes need to be created, with many accessor methods
- Struts can create an ActionForm for you:
 - Specify the class of the 'form-bean' as 'DynaActionForm'
 - Inside 'form-bean' place 'form-property' elements
 - One for each property you want the ActionForm to have
 - The 'form-property' has attributes for the name of the property, its class type and the initial value (optional)
- A 'DynaActionForm' object stores the properties for you
 - Unfortunately it will not hold any validation logic
 - The Struts Validator framework can be used to add this



Dynamic ActionForms

```
<struts-config>
  <form-beans>
    <!-- Configuring a Dynamic Action Form -->
    <form-bean name="dyna" type="org.apache.struts.action.DynaActionForm">
      <form-property name="forename" type="java.lang.String"/>
      <form-property name="surname" type="java.lang.String"/>
      <form-property name="age" type="java.lang.Integer"/>
      <form-property name="salary" type="java.lang.Double"/>
    </form-bean>
  </form-beans>
  <action-mappings>
    <!-- Mapping to show Dynamic Action Forms -->
    <action path="/dynaActionFormDemo" type="actions.DynaDemoAction"
      name="dyna" scope="request">
      <forward name="success" path="/jsp/tools/dynaActionFormOutput.jsp"/>
    </action>
  </action-mappings>
</struts-config>
```



Error Handling in Struts

- Struts error handling has two components
 - An 'ActionErrors' object added at request scope
 - Which contains one or more 'ActionError' objects
 - The 'errors' action in the HTML Tag Library
- When 'validate' returns a non-empty 'ActionErrors'
 - The request is forwarded to the original JSP
 - The JSP whose output triggered the original request
 - The 'input' attribute of the 'action' element supplies this information
 - Any 'errors' actions display the appropriate messages
 - The 'property' attribute can be used to display a single message



Error Handling in Struts

```
public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
    ActionErrors errors = new ActionErrors();
    if(getAddress() == null) {
        errors.add("address",new ActionError("demo.error.NoAddress"));
    }
    if(getPostcode() == null) {
        errors.add("postcode",new ActionError("demo.error.NoPostcode"));
    }
    return errors;
}
```

```
<html:errors property="address"/>
<html:errors property="postcode"/>
```



Error Handling in Struts

- When adding errors to an 'ActionErrors' objects
 - The constructor of 'ActionError' takes the key to a message in a message bundle
 - The first parameter of the 'add' method is a name that can be used by the 'errors' action
- Action Errors can also be used in Actions
 - Inside the 'execute' method you must manually:
 - Add the 'ActionErrors' object to the request object
 - Return an 'ActionForward' that points to the input JSP



Declarative Exception Handling

- Exception handling can be configured in 'struts-config'
 - Both for individual actions and for the application as a whole
- The '<exception>' element has attributes for:
 - The type of exception that is being configured
 - The URL that the request should be forwarded to
 - If and when this exception type is thrown
 - A key to be used to construct an 'ActionMessage'
 - The key indexes the appropriate error message in a properties file
- When the exception is thrown Struts:
 - Creates an 'ActionMessage' and adds it to an 'ActionErrors'
 - Forwards the request to the specified URL



Declarative Exception Handling

```
<action path="/exceptionsDemo" type="actions.ExceptionsDemoAction">  
  <exception key="exceptions.MessageA" type="TestExceptionA" path="/jsp/exceptionsOutput.jsp"/>  
  <exception key="exceptions.MessageB" type="TestExceptionB" path="/jsp/exceptionsOutput.jsp"/>  
</action>
```

Properties File

```
exceptions.MessageA = Message text for TestExceptionA  
exceptions.MessageB = Message text for TestExceptionB
```

exceptionsOutput.jsp

```
<%@page language="java" contentType="text/html" import="flights.*"%>  
<%@ taglib uri="/WEB-INF/tlds/struts-html.tld" prefix="forms" %>  
<h3>Exceptions Output Page</h3>  
<forms:errors/>
```