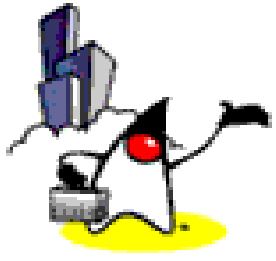
Advanced Struts



Sang Shin

sang.shin@sun.com
www.javapassion.com
Java™ Technology Evangelist
Sun Microsystems, Inc.

Disclaimer & Acknowledgments

- Even though Sang Shin is a full-time employee of Sun Microsystems, the content in this presentation is created as his own personal endeavor and thus does not reflect any official stance of Sun Microsystems.
- Sun Microsystems is not responsible for any inaccuracies in the contents.
- Acknowledgments:
  - I borrowed from presentation slides from the following sources
    - “Using the Struts framework” presentation material from Sue Spielman of Switchback Software (sspielman@switchbacksoftware.com)
  - Struts' user's guide is also used in creating slides and speaker notes
  - Source code examples are from Cookbook example originally written by
  - I also used “Programming Jakarta Struts” book written by Chuck Cavaness

Revision History

- 12/01/2003: version 1: created by Sang Shin
- 04/09/2004: version 2: speaker notes are polished a bit
- Things to do
  - speaker notes need to be added
  - more example codes need to be added
  - javascript slides need to be added

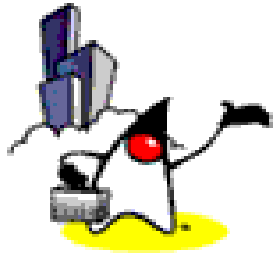
Advanced Struts Topics

- Extending & customizing Struts framework
  - Plug-In API (1.1)
- DynaActionForm (1.1)
- Validation framework (1.1)
- Declarative exception handling (1.1)
- Multiple application modules support (1.1)

Advanced Struts Topics

- Accessing database
- Struts-EL tag library
- Struts and security
- Struts utility classes
- Nested tag library
- Differences between Struts 1.0 and 1.1
- Roadmap

Extending & Customizing Struts Framework

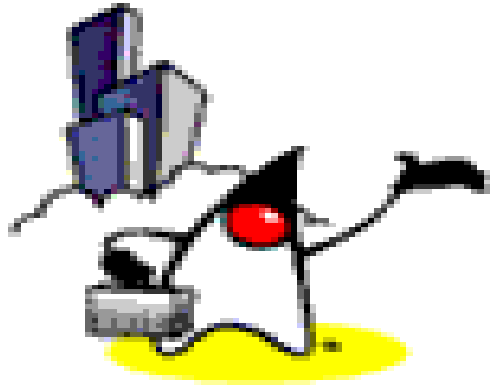


Extending Struts Framework

- Struts framework is designed with extension and customization in mind
  - It provides extension hooks
  - Validator and Tiles use these extension hooks
- Extend the framework only if default setting does not meet your needs
  - There is no guarantee on the backward compatibility in future version of Struts if you are using the extension

Extension/Customization Mechanisms of the Framework

- Plug-in mechanism (1.1)
- Extending framework classes (1.1)
  - Extending Struts configuration classes
  - Extending ActionServlet class
  - Extending RequestProcessor class
  - Extending Action class
- DispatchAction (1.1)
- Using multiple configuration files (1.1)



Extension/Customization:
Plug-in Mechanism

What is a Plug-in?

- Any Java class that you want to initialize when Struts application starts up and destroy when the application shuts down
  - Any Java class that implements org.apache.struts.action.Plugin interface

```
public interface PlugIn {
  public void init(ActionServlet servlet,
                   ApplicationConfig config)
                   throws ServletException;
  public void destroy();
}
```

Why Plug-in?

- Before Struts 1.1 (in Struts 1.0), you had to subclass ActionServlet to initialize application resources at startup time
- With plugin mechanism (in Struts 1.1), you create Plugin classes and configure them
- Generic mechanism
  - Struts framework itself uses plugin mechanism for supporting Validator and Tiles

How do you configure Plug-in's?

- Must be declared in struts-config.xml via <plug-in> element
- 3 example plug-in's in struts-example sample application

```
<plug-in className="org.apache.struts.plugins.ModuleConfigVerifier"/>

<plug-in
   className="org.apache.struts.webapp.example.memory.MemoryDatabasePlugIn">
  <set-property property="pathname" value="/WEB-INF/database.xml"/>
</plug-in>

<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property property="pathnames"
                value="/WEB-INF/validator-rules.xml,
                       /WEB-INF/validation.xml"/>
</plug-in>
```

How do Plug-in's get called?

- During startup of a Struts application, ActionServlet calls init() method of each Plug-in configured
- Plug-ins are called in the order they are configured in struts-config.xml file

init() of MemoryDatabasePlugin in struts-example sample application

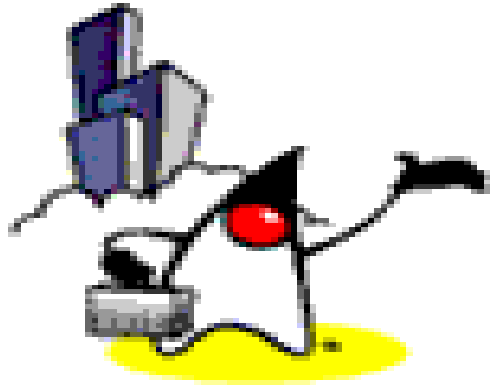
```
public final class MemoryDatabasePlugIn implements PlugIn {
    ...
    /**
     * Initialize and load our initial database from persistent storage.
     *
     * @param servlet The ActionServlet for this web application
     * @param config The ApplicationConfig for our owning module
     *
     * @exception ServletException if we cannot configure ourselves correctly
     */
    public void init(ActionServlet servlet, ModuleConfig config)
        throws ServletException {

        log.info("Initializing memory database plug in from '" +
                 pathname + "'");

        // Remember our associated configuration and servlet
        this.config = config;
        this.servlet = servlet;
```

Continued...

```
// Construct a new database and make it available
database = new MemoryUserDatabase();
try {
    String path = calculatePath();
    if (log.isDebugEnabled()) {
        log.debug(" Loading database from '" + path + "'");
    }
    database.setPathname(path);
    database.open();
} catch (Exception e) {
    log.error("Opening memory database", e);
    throw new ServletException("Cannot load database from '" +
                               pathname + "'", e);
}

// Make the initialized database available
servlet.getServletContext().setAttribute(Constants.DATABASE_KEY,
                                         database);

// Setup and cache other required data
setupCache(servlet, config);
}
```

destroy() of MemoryDatabasePlugin in struts-example1 sample code

```
public final class MemoryDatabasePlugIn implements PlugIn {
    ...
    /**
     * Gracefully shut down this database, releasing any resources
     * that were allocated at initialization.
     */
    public void destroy() {

        log.info("Finalizing memory database plug in");

        if (database != null) {
            try {
                database.close();
            } catch (Exception e) {
                log.error("Closing memory database", e);
            }
        }

        servlet.getServletContext().removeAttribute(Constants.DATABASE_KEY);
        database = null;
        servlet = null;
        database = null;
        config = null;

    }
```



Extension/Customization:

Extending Struts Framework Classes

Extending Configuration Classes

- org.apache.struts.config package contains all the classes that are in-memory representations of all configuration information in struts-config.xml file
  - ActionConfig, ActionMapping, ExceptionConfig, PluginConfig, MessageResourcesConfig, ControllerConfig, DataSourceConfig, FormBeanConfig, etc
- You extend these classes and then specify the extended class with class name attribute in struts-config.xml

Extending ActionServlet Class

- In Struts 1.0, it is common to extend ActionServlet class since
  - There was no plugin mechanism
  - ActionServlet handles all the controller functionality since there was no RequestProcessor
- Rarely needed in Struts 1.1
- Change web.xml

```
<servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>
            myPackage.myActionServlet
        </servlet-class>
</servlet>
```

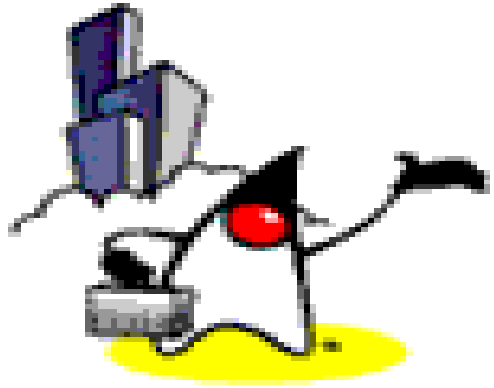
Extending RequestProcessor Class

- Via <controller> element in struts-config.xml
- process() method of RequestProcessor class is called before ActionForm class is initialized and execute() method of Action object get called
- Example in struts-cookbook sample code

```
<controller
  processorClass=”org.apache.struts.tiles.TilesRequestProcessor”/>
```

Extending Action Classes

- Useful to create a base Action class when common logic is shared among many Action classes
  - Create base Action class first then subclass it for actual Action classes



Extension/Customization:
DispatchAction

Why DispatchAction?

- To allow multiple “related” operations to reside in a single Action class
  - Instead of being spread over multiple Action classes
- To allow common logic to be shared
  - Related operations typically share some common business logic

How to use DispatchAction?

- Create a class that extends DispatchAction (instead of Action)
- In a new class, add a method for every function you need to perform on the service
  - The method has the same signature as the execute() method of an Action class
- Do not override execute() method
  - Because DispatchAction class itself provides execute() method
- Add an entry to struts-config.xml

Example: ItemAction class extends DispatchAction

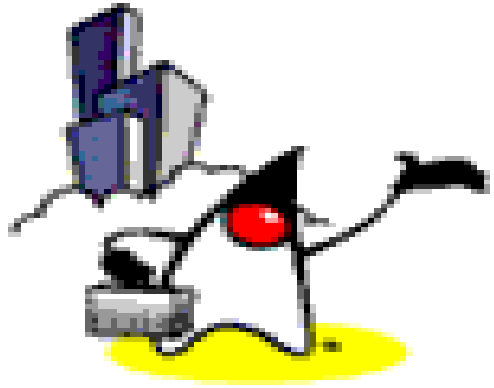
```
public class ItemAction extends DispatchAction {

    public ActionForward addItem(ActionMapping mapping,
                                 ActionForm form,
                                 HttpServletRequest request,
                                 HttpServletResponse response) throws Exception {
        try {
            // Code for item add
        } catch(Exception ex) {//exception}
    }

    public ActionForward deleteItem(ActionMapping mapping,
                                    ActionForm form,
                                    HttpServletRequest request,
                                    HttpServletResponse response) throws
    Exception {
        try {
            // Code for item deletion
        }
        catch(Exception ex){//exception}
    }
}
```

Example: struts-config.xml

```
<action  path="/item"
         type=" com.infosys.j2ee.sample.web.actions.ItemAction"
         name="itemForm"
         scope="request"
         validate="true"
         parameter="actionType"/>
```

The method can be invoked using a URL, like this:

ItemAction.do?actionType=addItem



Extension/Customization:
ForwardAction

Why ForwardAction?

- If you have the case where you don't need to perform any logic in the Action but would like to follow the convention of going through an Action to access a JSP, the ForwardAction can save you from creating many empty Action classes
- The benefit of the ForwardAction is that you don't have to create an Action class of your own
  - All you have to do is to declaratively configure an Action mapping in your Struts configuration file.

Example: ForwardAction

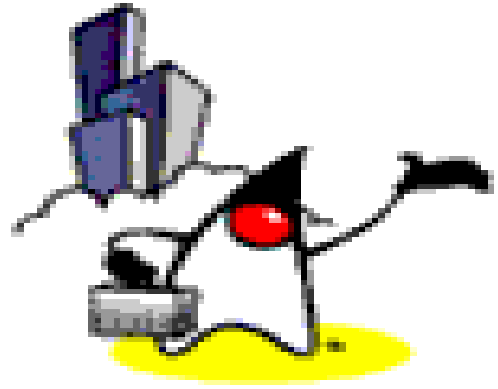
- Suppose that you had a JSP page called index.jsp and instead of calling this page directly, you would rather have the application go through an Action class
- http://hostname/appname/home.do

```
<action
  path="/home"
  parameter="/index.jsp"
  type="org.apache.struts.actions.ForwardAction"
  scope="request"
  validate="false">
</action>
```

Example: ForwardAction

- Another way for forwarding

```
<action
   path="/home"
   forward="/index.jsp">
</action>
```



Extension/Customization:
Using Multiple Struts Configuration File

Multiple Configuration Files

- Useful for multi-developer environment
  - At least, a single struts-config.xml file is not a bottleneck anymore
- Different from “Multiple modules” support

web.xml of struts-example

```
<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,
        /WEB-INF/struts-config-registration.xml
    </param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

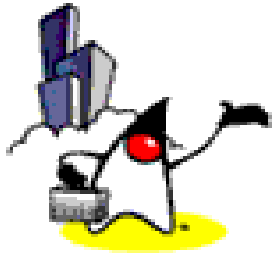
struts-config-registration.xml of struts-example sample app

```
<!--
    This is the Struts configuration file for the registration
    portion of the example application, using the proposed new syntax.
-->

<struts-config>

  <!-- ========== Form Bean Definitions ================================
     -->
  <form-beans>

    <!-- Registration form bean -->
    <form-bean      name="registrationForm"
                    type="org.apache.struts.webapp.example.RegistrationForm"/>

  </form-beans>

  <!-- ========== Global Forward Definitions ============================== --
     >
  <global-forwards>
    <forward   name="registration"        path="/registration.jsp"/>
  </global-forwards>
```

struts-config-registration.xml of struts-example

```
<!-- ========== Action Mapping Definitions ============================== -->
 <action-mappings>

  <!-- Edit user registration -->
  <action    path="/editRegistration"
             type="org.apache.struts.webapp.example.EditRegistrationAction"
        attribute="registrationForm"
            scope="request"
         validate="false">
    <forward name="success"             path="/registration.jsp"/>
  </action>

  <!-- Save user registration -->
  <action    path="/saveRegistration"
             type="org.apache.struts.webapp.example.SaveRegistrationAction"
             name="registrationForm"
            scope="request"
            input="registration"/>

 </action-mappings>

</struts-config>
```

DynaActionForm (Introduced in 1.1)



Why DynaActionForm?

- Issues with using ActionForm
  - ActionForm class has to be created in Java programming language
  - For each HTML form page, a new ActionForm class has to be created
  - Every time HTML form page is modified (a property is added or remove), ActionForm class has to be modified and recompiled
- DynaActionForm support is added to Struts 1.1 to address these issues

What is DynaActionForm?

- org.apache.struts.action.DynaActionForm
  - extends ActionForm class
- In DynaActionForm scheme,
  - Properties are configured in configuration file rather than coding
  - reset() method resets all the properties back to their initial values
  - You can still subclass DynaActionForm to override reset() and/or validate() methods
    - Version exists that works with Validator framework to provide automatic validation

How to Configure DynaActionForm?

- Configure the properties and their types in your struts-config.xml file
  - add one or more <form-property> elements for each <form-bean> element

Example from struts-examplesSteve

```
<form-beans>

 <!-- Simple ActionForm Bean -->
 <form-bean name="simpleForm"
   type="examples.simple.SimpleActionForm"/>

 <!-- DynaActionForm Bean -->
 <form-bean name="dynaForm"
   type="org.apache.struts.action.DynaActionForm">
         <form-property name="name" type="java.lang.String" />
         <form-property name="secret" type="java.lang.String" />
         <form-property name="color" type="java.lang.String" />
         <form-property name="confirm" type="java.lang.Boolean" />
         <form-property name="rating" type="java.lang.String" />
         <form-property name="message" type="java.lang.String" />
         <form-property name="hidden" type="java.lang.String" />
 </form-bean>

</form-beans>
```

Types Supported by DynaActionForm

- java.lang.BigDecimal, java.lang.BigInteger
- boolean and java.lang.Boolean
- byte and java.lang.Byte
- char and java.lang.Character
- java.lang.Class, double and java.lang.Double
- float and java.lang.Float
- int and java.lang.Integer
- long and java.lang.Long
- short and java.lang.Short
- java.lang.String
- java.sql.Date, java.sql.Time, java.sql.Timestamp

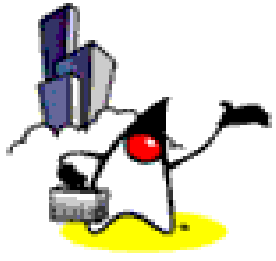
How to perform Validation with DynaActionForm?

- DynaActionForm does not provide default behavior for validate() method
  - You can subclass and override validate() method but it is not recommended
- Use Validator Framework
  - Use DynaValidatorForm class (instead of DynaActionForm class)
  - DynaValidatorForm class extends DynaActionForm and provides basic field validation based on an XML file

Example from strut-example sample application

```
<form-beans>

  <!-- Logon form bean -->
  <form-bean name="logonForm"
  type="org.apache.struts.validator.DynaValidatorForm">
      <form-property name="username"
                     type="java.lang.String"/>
      <form-property name="password"
                     type="java.lang.String"/>
  </form-bean>

  <!-- Subscription form bean -->
  <form-bean name="subscriptionForm"
  type="org.apache.struts.webapp.example.SubscriptionForm"/>

</form-beans>
```

Validation Framework (added to Struts 1.1 Core)



Why Struts Validation Framework?

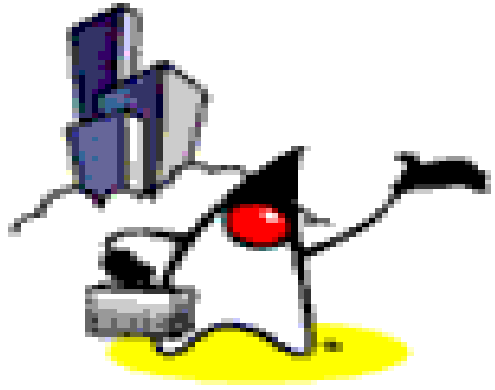
- Issues with writing validate() method in ActionForm classes
  - You have to write validate() method for each ActionForm class, which results in redundant code
  - Changing validation logic requires recompiling
- Struts validation framework addresses these issues
  - Validation logic is configured using built-in validation rules as opposed to writing it in Java code

What is Struts Validation Framework?

- Originated from “generic” Validator framework from Jakarta
- Struts 1.1 includes this by default
  - Allows declarative validation for many fields
  - Formats
    - Dates, Numbers, Email, Credit Card, Postal Codes
  - Lengths
    - Minimum, maximum
  - Ranges

Validation Rules

- Rules are tied to specific fields in a form
- Basic Validation Rules are built-in in the Struts 1.1 core distribution
  - e.g. “required”, “minLength”, “maxLength”, etc.
- The built-in validation rules come with Javascript that allows you to do client side validation
- Custom Validation rules can be created and added to the definition file.
  - Can also define regular expressions



Validation Framework:
How to configure and use Validation framework?

Things to do in order to use Validator framework

- Configure Validator Plug-in
- Configure validation.xml file (and validation-rules.xml file)
- Extend Validator ActionForms or Dynamic ActionForms
- Set validate=”true” on Action Mappings
- Include the <html:javascript> tag for client side validation in the form's JSP page

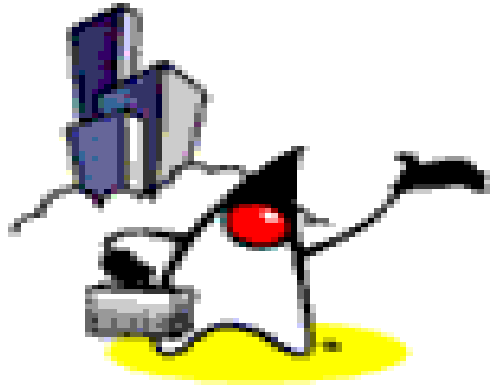
How to Configure Validator Plug-in

- Validator Plug-in should be configured in the Struts configuration file
  - Just add the following <plug-in> element

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
 <set-property property="pathnames"
                value="/WEB-INF/validator-rules.xml,
                       /WEB-INF/validation.xml"/>
</plug-in>
```

Two Validator Configuration Files

- validation-rules.xml
  - Contains global set of validation rules
  - Provided by Struts framework
- validation.xml
  - Application specific
  - Provided by application developer
  - It specifies which validation rules from validation-rules.xml file are used by a particular ActionForm
  - No need to write validate() code in ActionForm class



Validation Framework:
validation-rules.xml

Built-in (basic) Validation Rules

- required, requiredif
- minlength
- maxlength
- mask
- byte, short, integer, long, float, double
- date, range, intRange, floatRange
- creditCard, email

validation-rules.xml: Built-in “required” validation rule

```
<form-validation>

  <global>

    <validator name="required"
          classname="org.apache.struts.validator.FieldChecks"
          method="validateRequired"
          methodParams="java.lang.Object,
                org.apache.commons.validator.ValidatorAction,
                org.apache.commons.validator.Field,
                org.apache.struts.action.ActionErrors,
                javax.servlet.http.HttpServletRequest"
          msg="errors.required">

      <javascript><![CDATA[
        function validateRequired(form) {
          var isValid = true;
          ...
      </javascript>

    </validator>
```

validation-rules.xml: Built-in "minlength" validation rule

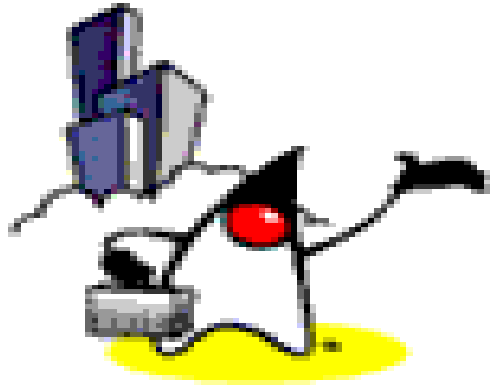
```
<validator name="minlength"
            classname="org.apache.struts.validator.FieldChecks"
            method="validateMinLength"
            methodParams="java.lang.Object,
                 org.apache.commons.validator.ValidatorAction,
                 org.apache.commons.validator.Field,
                 org.apache.struts.action.ActionErrors,
                 javax.servlet.http.HttpServletRequest"
            depends=""
            msg="errors.minlength">

  <javascript><![CDATA[
    function validateMinLength(form) {
       var isValid = true;
       var focusField = null;
       ...
  </javascript>

</validator>
```

Attributes of <validator ...> Element

- name: logical name of the validation rule
- classname, method: class and method that contains the validation logic
- methodParams: comma-delimited list of parameters for the method
- msg: key from the resource bundle
  - Validator framework uses this to look up a message from Struts resource bundle
- depends: other validation rules that should be called first

Built-in Error Messages From Validator Framework

```
# Built-in error messages for validator framework checks
# You have to add the following to your application's resource bundle
errors.required={0} is required.
errors.minlength={0} cannot be less than {1} characters.
errors.maxlength={0} cannot be greater than {2} characters.
errors.invalid={0} is invalid.
errors.byte={0} must be an byte.
errors.short={0} must be an short.
errors.integer={0} must be an integer.
errors.long={0} must be an long.
errors.float={0} must be an float.
errors.double={0} must be an double.
errors.date={0} is not a date.
errors.range={0} is not in the range {1} through {2}.
errors.creditcard={0} is not a valid credit card number.
errors.email={0} is an invalid e-mail address.
```



Validation Framework:
validation.xml

validation.xml: logonForm (from struts-examples sample code)

```
<form-validation>

  <!-- ========== Default Language Form Definitions ==================== -->
  <formset>

    <form name="logonForm">

      <field property="username"
             depends="required, minlength,maxlength">
        <arg0   key="prompt.username"/>
        <arg1   key="${var:minlength}" name="minlength"
          resource="false"/>
        <arg2   key="${var:maxlength}" name="maxlength"
          resource="false"/>
        <var>
          <var-name>maxlength</var-name>
          <var-value>16</var-value>
        </var>
        <var>
          <var-name>minlength</var-name>
          <var-value>3</var-value>
        </var>
      </field>
```

validation.xml: logonForm (from struts-examples sample code)

```
<field property="password"
       depends="required, minlength,maxlength"
       bundle="alternate">
   <arg0   key="prompt.password"/>
   <arg1   key="${var:minlength}" name="minlength"
     resource="false"/>
   <arg2   key="${var:maxlength}" name="maxlength"
     resource="false"/>
   <var>
      <var-name>maxlength</var-name>
      <var-value>16</var-value>
   </var>
   <var>
      <var-name>minlength</var-name>
      <var-value>3</var-value>
   </var>
</field>

</form>
```

validation.xml: registrationForm (from struts-examples sample code)

```
<form name="registrationForm">

  <field property="fromAddress"
         depends="required,email">
    <arg0   key="prompt.fromAddress"/>
  </field>
  <field property="fullName"
         depends="required">
    <arg0     key="prompt.fullName"/>
  </field>
  <field property="replyToAddress"
         depends="email">
    <arg0   key="prompt.replyToAddress"/>
  </field>
  <field property="username"
         depends="required">
    <arg0     key="prompt.username"/>
  </field>

</form>

</formset>
</form-validation>
```

<form ...> Element

- Defines a set of fields to be validated
  - <!ELEMENT form (field+)>
- Attributes
  - Name: Corresponds name attribute of <form-bean> in struts-config.xml

struts-config.xml: logonForm (from struts-examples sample code)

```
<form-beans>

  <!-- Logon form bean -->
  <form-bean       name="logonForm"
                type="org.apache.struts.validator.DynaValidatorForm">
    <form-property name="username" type="java.lang.String"/>
    <form-property name="password" type="java.lang.String"/>
  </form-bean>

  <!-- Subscription form bean -->
  <form-bean      name="subscriptionForm"
               type="org.apache.struts.webapp.example.SubscriptionForm"/>

</form-beans>
```

<field ...> Element

- Corresponds to a property in an ActionForm
  - <!ELEMENT field (msg?, arg0?, arg1?, arg2?, arg3?, var*)>
- Attributes
  - property: name of a property (in an ActionForm) that is to be validated
  - depends: comma-delimited list of validation rules to apply against this field
    - All validation rules have to succeed

<field ...> Element: <msg ...> child element

- Allows you to specify an alternate message for a field element
- <msg name=”..” key=”...” resource=”...”/>
  - name: specifies rule with which the msg is used, should be one of the rules in validation-rules.xml
  - key: specifies key from the resource bundle that should be added to the ActionError if validation fails
  - resource: if set to false, the value of key is taken as literal string

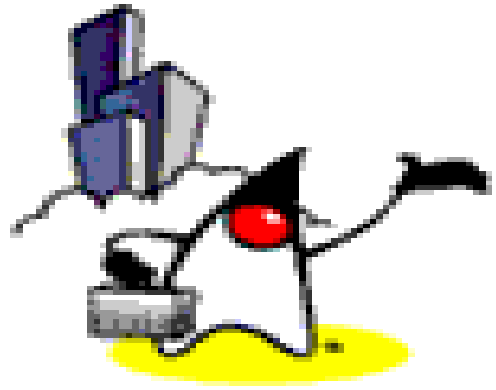
<field ...> Element: <arg0 ...> child element

- <arg0 key=”..” name=”..” resource=”..”>
  - Are used to pass additional values to the message
  - <arg0> is the 1st replacement and <arg1> is the 2nd replacement, and so on

<field ...> Element: <var ...> child element

<var>
  <var-name>...</var-name>
  <var-value>...</var-value>
</var>

- Set parameters that a field element may need to pass to one of its validation rules such as minimum and mixiumum values in a range validation
- Referenced by <argx> elements using ${var: var-name} syntax



Validation Framework:

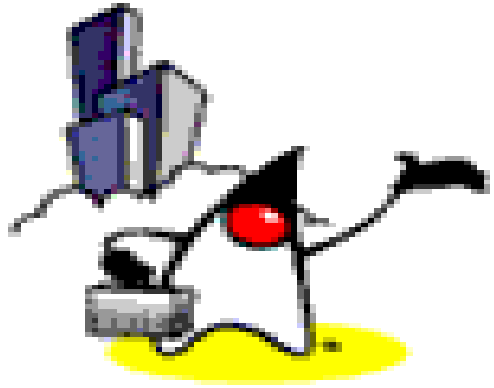
How do you use ActionForm with Validator?

ActionForm and Validator

- You cannot use ActionForm class with the Validator
  - Instead you need to use special subclasses of ActionForm class
- Validator supports two types special ActionForms
  - Standard ActionForms
    - ValidatorActionForm
    - ValidatorForm
  - Dynamic ActionForms
    - DynaValidatorActionForm
    - DynaValidatorForm

Choices You have

- Standard ActionForms or Dynamic ActionForms? (rehash)
  - If you want to specify ActionForms declaratively, use dynamic ActionForms
- xValidatorActionForm or xValidatorForm?
  - Use xValidatorActionForm if you want more fine-grained control over which validation rules are executed
  - In general, xValidatorForm should be sufficient



Validation Framework:

struts-example
Sample code

struts-config.xml: DynaValidatorForm

```
<!-- ========== Form Bean Definitions === -->
<form-beans>

  <!-- Logon form bean -->
  <form-bean  name="logonForm"
                  type="org.apache.struts.validator.DynaValidatorForm">
    <form-property name="username" type="java.lang.String"/>
    <form-property name="password" type="java.lang.String"/>
  </form-bean>

  <!-- Subscription form bean -->
  <form-bean  name="subscriptionForm"
              type="org.apache.struts.webapp.example.SubscriptionForm"/>

</form-beans>
```

validation-rules.xml: Built-in "required" validation rule

```
<form-validation>

  <global>

    <validator name="required"
          classname="org.apache.struts.validator.FieldChecks"
          method="validateRequired"
          methodParams="java.lang.Object,
                org.apache.commons.validator.ValidatorAction,
                org.apache.commons.validator.Field,
                org.apache.struts.action.ActionErrors,
                javax.servlet.http.HttpServletRequest"
          msg="errors.required">

      <javascript><![CDATA[
        function validateRequired(form) {
          var isValid = true;
          ...
      </javascript>

    </validator>
```

validation-rules.xml: Built-in "minlength" validation rule

```
<validator name="minlength"
           classname="org.apache.struts.validator.FieldChecks"
           method="validateMinLength"
           methodParams="java.lang.Object,
                org.apache.commons.validator.ValidatorAction,
                org.apache.commons.validator.Field,
                org.apache.struts.action.ActionErrors,
                javax.servlet.http.HttpServletRequest"
           depends=""
           msg="errors.minlength">

  <javascript><![CDATA[
    function validateMinLength(form) {
        var isValid = true;
        var focusField = null;
        ...
  </javascript>

</validator>
```

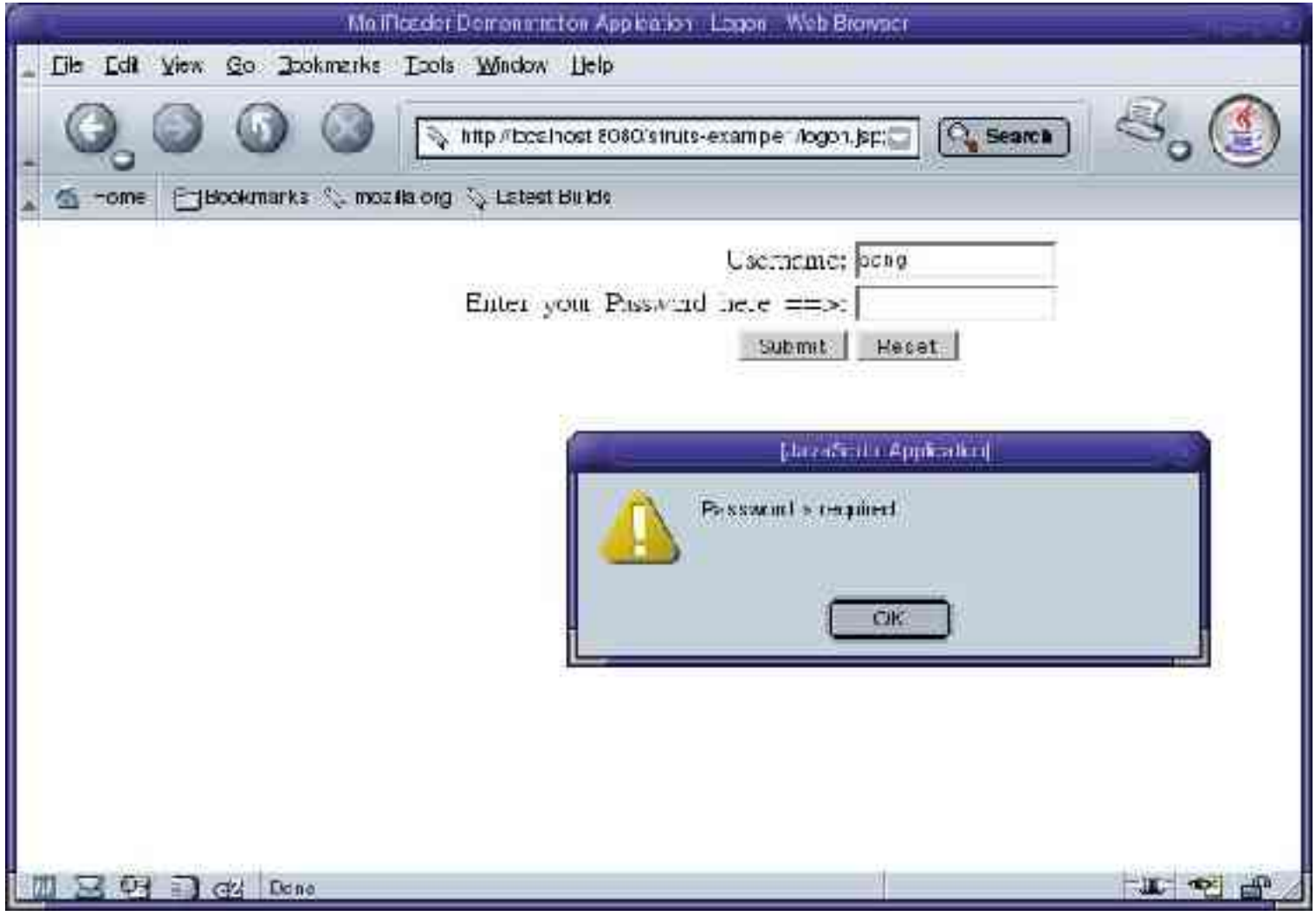
validation.xml: logonForm

```
<form-validation>

  <!-- ========== Default Language Form Definitions ==================== -->
  <formset>

    <form name="logonForm">

      <field property="username"
             depends="required, minlength,maxlength">
        <arg0   key="prompt.username"/>
        <arg1   key="${var:minlength}" name="minlength"
          resource="false"/>
        <arg2   key="${var:maxlength}" name="maxlength"
          resource="false"/>
        <var>
          <var-name>maxlength</var-name>
          <var-value>16</var-value>
        </var>
        <var>
          <var-name>minlength</var-name>
          <var-value>3</var-value>
        </var>
      </field>
```

ApplicationResources.Properties

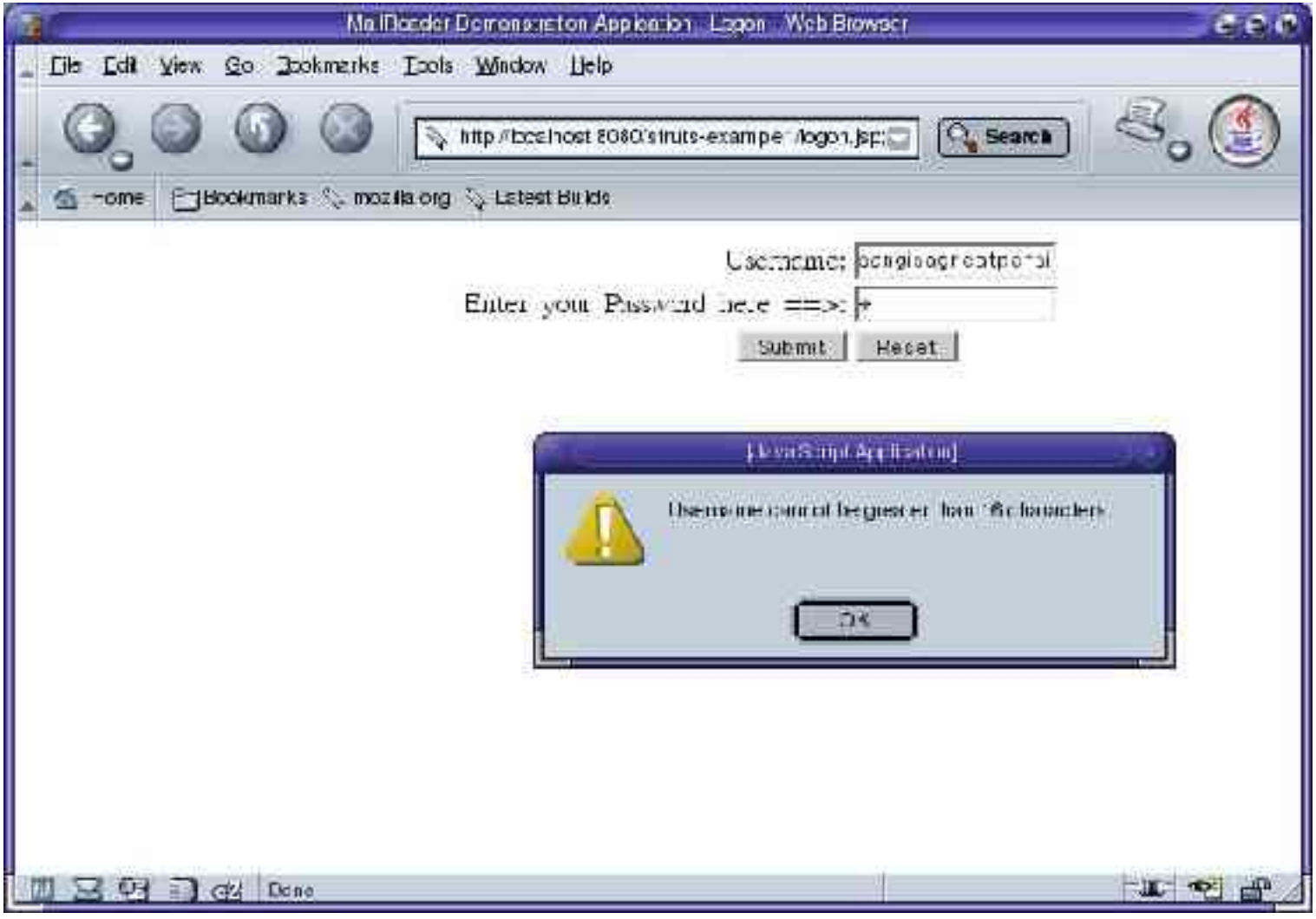
```
# Standard error messages for validator framework checks
errors.required={0} is required.
errors.minlength={0} cannot be less than {1} characters.
errors.maxlength={0} cannot be greater than {2} characters.
errors.invalid={0} is invalid.
errors.byte={0} must be an byte.
errors.short={0} must be an short.
errors.integer={0} must be an integer.
errors.long={0} must be an long.
errors.float={0} must be an float.
errors.double={0} must be an double.
errors.date={0} is not a date.
errors.range={0} is not in the range {1} through {2}.
errors.creditcard={0} is not a valid credit card number.
errors.email={0} is an invalid e-mail address.
```

LogonForm Validation: “required”



LogonForm Validation: “minlength”

LogonForm Validation: “maxlength”

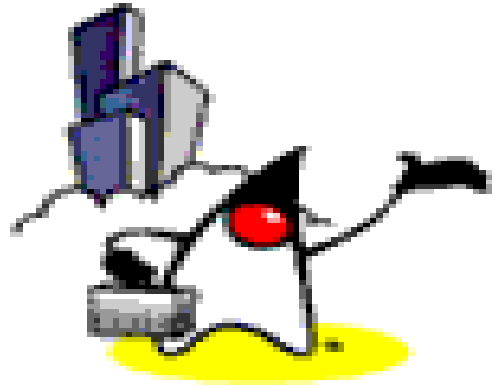


struts-config.xml: Regular ActionForm class (actually extension of it)

```
<!-- ========== Form Bean Definitions === -->
<form-beans>

  <!-- Logon form bean -->
  <form-bean  name="logonForm"
                  type="org.apache.struts.validator.DynaValidatorForm">
    <form-property name="username" type="java.lang.String"/>
    <form-property name="password" type="java.lang.String"/>
  </form-bean>

  <!-- Subscription form bean -->
  <form-bean      name="subscriptionForm"
              type="org.apache.struts.webapp.example.SubscriptionForm"/>

</form-beans>
```

SubscriptionForm class

```
public final class SubscriptionForm extends ActionForm  {
   ...
   public String getAction() {
       return (this.action);
   }
   public void setAction(String action) {
      this.action = action;
   }
   ...
   public ActionErrors validate(ActionMapping mapping,
                                HttpServletRequest request) {
       ...
   }
  ...
}
```



Validation Framework:
How to perform Client-side Validation using JavaScript?

JavaScript Support in Validator Framework

- The Validator framework is also capable of generating JavaScript for your Struts application using the same framework as for server-side validation
  - By using a set of JSP custom tags designed specifically for this purpose

Configuring validation-rules.xml for JavaScript

- A custom tag is used to generate client-side validation based on a javascript attribute being present within the <validator> element in the validation-rules.xml file
- Use the custom tag in your JSP page
  - The text from <javascript> element is written to the JSP page to provide client-side validation

Configuring validation-rules.xml for JavaScript

```
<form-validation>

  <global>

    <validator name="required"
          classname="org.apache.struts.validator.FieldChecks"
          method="validateRequired"
          methodParams="java.lang.Object,
                org.apache.commons.validator.ValidatorAction,
                org.apache.commons.validator.Field,
                org.apache.struts.action.ActionErrors,
                javax.servlet.http.HttpServletRequest"
          msg="errors.required">

      <javascript><![CDATA[
        function validateRequired(form) {
          var isValid = true;
          ...
      </javascript>

    </validator>
```

Validation Rules for the logonForm in validation.xml

```
<form name="logonForm">

   <field property="username"
          depends="required, minlength,maxlength">
      <arg0   key="prompt.username"/>
      <arg1   key="${var:minlength}" name="minlength"
        resource="false"/>
      <arg2   key="${var:maxlength}" name="maxlength"
        resource="false"/>
       ...
   </field>

   <field property="password"
          depends="required, minlength,maxlength"
          bundle="alternate">
        ...
```

Things You Have to Do in your JSP Page

- You will need to include the <html:javascript> tag with the name of the ActionForm that it's going to validate against
- The formName attribute is used to look up the set of validation rules to include as JavaScript in the page

logon.jsp Page (from struts-example sample code)

```
<html:javascript formName="logonForm"
                 dynamicJavascript="true"
                 staticJavascript="false"/>
<script language="Javascript1.1" src="staticJavascript.jsp">
</script>
```

Things You Have to Do in your JSP Page

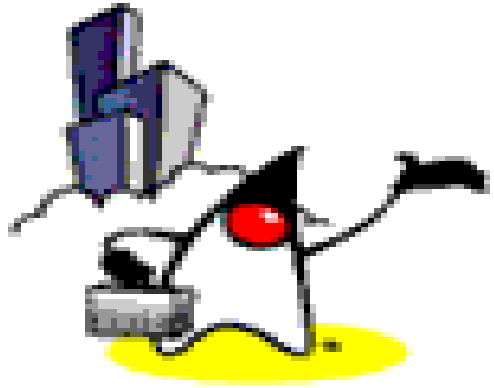
- You will have to add an onsubmit event handler for the form manually
- When the form is submitted, the validateLogonForm( ) JavaScript function will be invoked
  - The validation rules will be executed, and if one or more rules fail, the form will not be submitted.

Things You Have to Do in your JSP Page

- The <html:javascript> tag generates a function with the name validateXXX( ) , where XXX is the name of the ActionForm
  - Thus, if your ActionForm is called logonForm, the javascript tag will create a JavaScript function called validateLogonForm( ) that executes the validation logic
  - This is why the onsubmit( ) event handler called the validateLogonForm( ) function.

logon.jsp Page (from struts-example sample code)

```
<html:form action="/logon" focus="username"
        onsubmit="return validateLogonForm(this);">
<table border="0" width="100%">

  <tr>
    <th align="right">
      <bean:message key="prompt.username"/>:
    </th>
    <td align="left">
      <html:text property="username" size="16"
maxlength="18"/>
    </td>
  </tr>
...
</html>
```

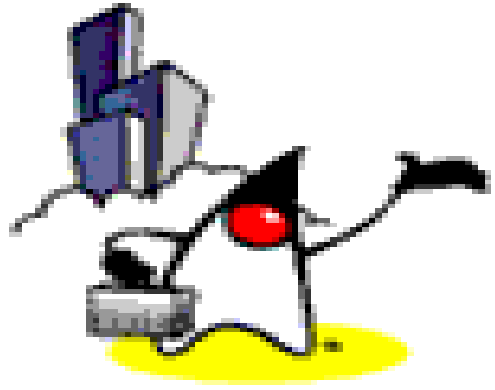


Validation Framework:
I18N and Validation

I18N and Validator

- Validator framework uses Struts resource bundle
- <formset> element in validation.xml supports language, country, variant

```
<formset>
 ...
</formset>
<formset language="fr">
 ...
</formset>
<formset language="fr" country="CA">
 ...
</formset>
```

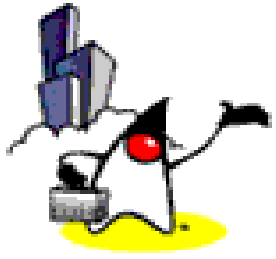


Validation Framework:
Advanced Features of Validator

Advanced Features

- Creating custom validators
- Using programmatic validation along with Validator

Declarative Exception Handling



Why Declarative Exception Handling?

- Previously (Struts 1.0), Action class has to deal with all business logic exceptions itself
  - No common exception handling
  - Rather difficult to implement

What is Declarative Exception Handling?

- Exception handling policy is declaratively specified in struts-config.xml
  - Exceptions that may occur
  - What to do if exceptions occur
- Can be global and per-Action
  - From a particular Exception class or super class
  - To an application-relative path
- Requires a small change to your Action to leverage this feature ...

Example in struts-example

```
<!-- Process a user logon -->
<action
  path="/logon"
   type="org.apache.struts.webapp.example.LogonAction"
  name="logonForm"
  scope="session"
  input="logon">
   <exception
     key="expired.password"
     type="org.apache.struts.webapp.example.ExpiredPasswordException"
     path="/changePassword.jsp"/>
</action>
```

How Does it Work?

- When an Exception is thrown inside execute() of Action class, processException() method of RequestProcessor class is called
- processException() method checks if <exception> element is configured for that specific exception type
- If there is,
  - Default exception handler creates and stores an ActionError into the specified scope
  - Control is then forwarded to the resource specified in path attribute

Example in struts-example1

```
public ActionForward execute
   (ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response)
     throws Exception {
     ...
     throw new PasswordExpiredException(...);
     ...
}
```

Customizing Exception Handling

- Create a class that extends org.apache.struts.action.ExceptionHandler
  - Override execute() method

```
public ActionForward execute( Exception ex,
                              ExceptionConfig exConfig,
                              ActionMapping mapping,
                              ActionForm formInstance,
                              HttpServletRequest request,
                              HttpServletResponse response
    ) throws ServletException;
```

Customizing Exception Handling

- Configure the Custom Exception Handler (globally or per action basis)

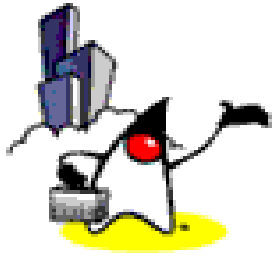
```
<global-exceptions>
  <exception
    handler="com.cavaness.storefront.CustomizedExceptionHandler"
    key="global.error.message"
    path="/error.jsp"
    scope="request"
    type="java.lang.Exception"/>
</global-exceptions>
```

Customizing Exception Handling

- Multiple Custom Exception Handlers

```
<global-exceptions>
  <exception
    handler="com.cavaness.storefront.CustomizedExceptionHandler"
    key="global.error.message"
    path="/error.jsp"
    scope="request"
    type="java.lang.Exception"/>

  <exception
    handler="com.cavaness.storefront.SecurityExceptionHandler"
    key="security.error.message"
    path="/login.jsp"
    scope="request"
    type="com.cavaness.storefront.SecurityException"/>
</global-exceptions>
```

Application Modules



What are Application Modules?

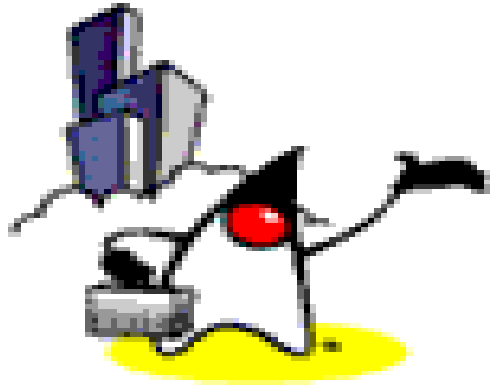
- Allows a single Struts application to be split into multiple modules
  - Each module has its own Struts configuration file, JSP pages, Actions

Multiple Application Module Support

- Necessary for large scale applications
  - Allows parallel development
- Design goals:
  - Support multiple independent Struts configuration files in the same application
  - Modules are identified by an "application prefix" that follows the context path
  - Existing Struts-based applications should be able to be installed individually, or as a sub-application, with no changes to the pages, Actions, form beans, or other code

Multiple Application Modules Support

- Implications of the design goals:
  - "Default" sub-application with a zero-length prefix (like the ROOT context in Tomcat)
  - "Context-relative" paths must now be treated as "sub-application-relative"
  - ActionServlet initialization parameters migrate to the Struts configuration file
  - Controller must identify the relevant sub-application, and make its resources available (as request attributes)
  - APIs must be reviewed for assumptions about there being (for example) only one set of ActionMappings



Application Modules:
How to configure modules

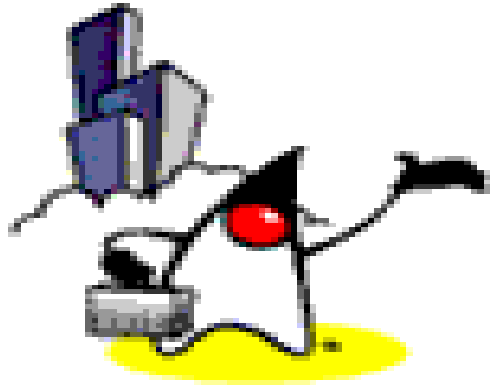
How to set up and use multiple Application Modules?

- Prepare a config file for each module.
- Inform the controller of your module.
- Use actions to refer to your pages.

Informing the Controller

- In web.xml

```
...
<init-param>
  <param-name>config</param-name>
  <param-value>/WEB-INF/conf/struts-default.xml</param-value>
</init-param>
<init-param>
  <param-name>config/module1</param-name>
  <param-value>/WEB-INF/conf/struts-module1.xml</param-value>
</init-param>
...
```



Application Modules:
How to switch modules

Methods for Switching Modules

- Two basic methods to switching from one module to another
  - Use a forward (global or local) and specify the contextRelative attribute with a value of true
  - Use the built-in org.apache.struts.actions.SwitchAction

Global Forward

```
...
<struts-config>
...
<global-forwards>
<forward name="toModuleB"
   contextRelative="true"
   path="/moduleB/index.do"
   redirect="true"/>
...
</global-forwards>
...
</struts-config>
```

Local Forward

```
...
<struts-config>
...
<action-mappings>
...
<action ... >
<forward name="success"
    contextRelative="true"
    path="/moduleB/index.do"
    redirect="true"/>
</action>
...
</action-mappings>
...
</struts-config>
```

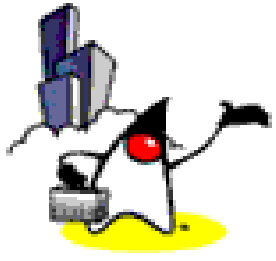
org.apache.struts.actions.SwitchAction

```
...
<action-mappings>
 <action path="/toModule"
 type="org.apache.struts.actions.SwitchAction"/>
 ...
 </action-mappings>
 ...
```

How to change module

- To change module, use URI
  - http://localhost:8080/toModule.do?prefix=/moduleB&page=/index.do
- If you are using the "default" module as well as "named" modules (like "/moduleB"), you can switch back to the "default" module with a URI
  - http://localhost:8080/toModule.do?prefix=&page=/index.do

Struts EL Tag library



Struts EL Extension

- Extension of the Struts tag library
- Uses the expression evaluation engine in the Jakarta Taglibs implementation of the JSP Standard Tag Library (version 1.0) to evaluate attribute values
- Some of the Struts tags were not ported to this library
  - their functionality was entirely supplied by the JSTL
- Requires the use of the Struts tag library, and the Java Server Pages Standard Tag Library

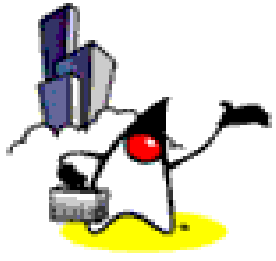
Tag Mapping

- Every Struts tag that provides a feature that is not covered by the JSTL (1.0) library is mapped into the Struts-EL library
- Bean Tag Library Tags NOT Implemented in Struts-EL
  - cookie (in Struts): c:set, EL (in JSTL)
  - define (in Struts), c:set, EL (In JSTL)

How to use Struts EL

- Struts
  - <bean:message key='<%= stringvar %>'/>
- Struts EL
  - <bean-el:message key="${stringvar}"/>

Struts & Security



JAAS

- Struts 1.1 and later offers direct support for the standard Java Authentication and Authorization Service (JAAS)
- You can now specify security roles on an action-by-action basis

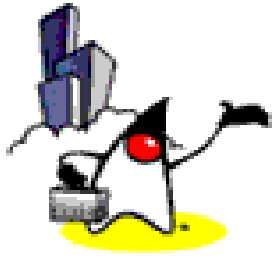
SSL Extension (sslext)

- Extension to Struts 1.1
  - http://sslext.sourceforge.net
- Extends the ActionConfig class, RequestProcessor, and Plugin classes to define a framework where developers may specify the transmission protocol behavior for Struts applications
- Within the Struts configuration file, developers specify which action requests require HTTPS transmission and which should use HTTP

SSL Extension (sslext)

- Can also specify whether to redirect "improperly-protocoled" requests to the correct protocol
- <html:link> and <html:form> tag extension
  - Struts actions specified in either of these tags are analyzed to determine the protocol that should be used in requesting that action
  - The HTML generated by these tags will specify the proper protocol

Commons: DynaBeans



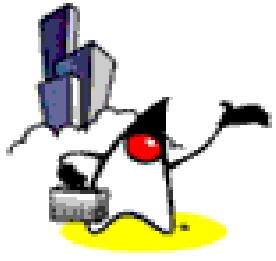
COMMONS-BEANUTILS

- Pluggable type converters
- Mapped properties
  - contact.phone(Work)
- DynaBeans
  - Transparently supported by BeanUtils and PropertyUtils

Commons-Beanutils -- DynaBeans

```
public interface DynaBean {
  public Object get(String name);
  public Object get(String name, int index);
  public Object get(String name, String key);
  public void set(String name, Object value);
  public void set(String name, int index, Object
  value);
  public void set(String name, String key,
  Object value);
}
```

Struts Utilities



Utilities

- org.apache.struts.util package
- 'Families' of classes that solve common web app problems.
- Suitable for general Java programming and are worth checking out
- Utilities classes include:
  - Beans & Properties
  - Collection Classes
  - JDBC connection pool
  - Message Resources

Utility Classes

- common-*.jar – Now required for Struts
  - Might require some package renaming for 1.0.2 Struts apps
- Many utilities are now located in the Jakarta Commons project
  - http://jakarta.apache.org/commons/
  - BeanUtils Package org.apache.commons.beanutils
  - Collections Package org.apache.commons.collections
  - Digester Package org.apache.commons.digester

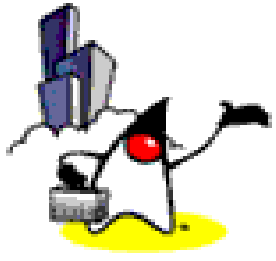
BeanUtil & PropertyUtil

- Relies on using Java Reflection to manipulate Java Beans
- Used throughout Struts (IteratorTag, WriteTag)
- Need to make sure you have a valid bean!
  - Null constructor
  - Public class declaration
  - Mutator/Accessor methods

XML Parsing

- Digester package provides for rules-based processing of arbitrary XML documents.
- Higher level, more developer-friendly interface to SAX events
- Digester automatically navigates the element hierarchy of the XML document
  - Element Matching Patterns
  - Processing Rule

Accessing Database



Recommendation First

- Database access logic belong to business logic (Model)
- Action class should be just a thin layer to Model
  - All the database access code should be encapsulated in the business logic
  - Struts doesn't know what persistent layer you are using (or even if there is a persistence layer)
- Use database access logic of your model component (EJB, JDO, etc.) if possible

However, Struts provides DataSource manager

- Struts DataSource manager is configured as an element in struts-config.xml
- Can be used to deploy any connection pool that implements the javax.sql.DataSource interface
- Is configurable totally from JavaBean properties

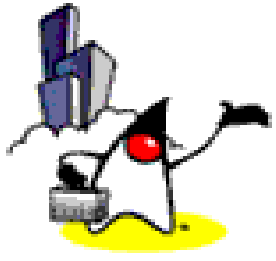
Example in struts-config.xml

```
<data-sources>
 <!-- configuration for commons BasicDataSource -->
 <data-source type="org.apache.commons.dbcp.BasicDataSource">
    <set-property
     property="driverClassName"
     value="org.postgresql.Driver" />
    <set-property
     property="url"
     value="jdbc:postgresql://localhost/mydatabase" />
    <set-property property="username" value="me" />
    <set-property property="password" value="test" />
    <set-property property="maxActive" value="10" />
    <set-property property="maxWait" value="5000" />
    <set-property property="defaultAutoCommit" value="false" />
    <set-property property="defaultReadOnly" value="false" />
    <set-property property="validationQuery"
     value="SELECT COUNT(*) FROM market" />
 </data-source>
</data-sources>
```

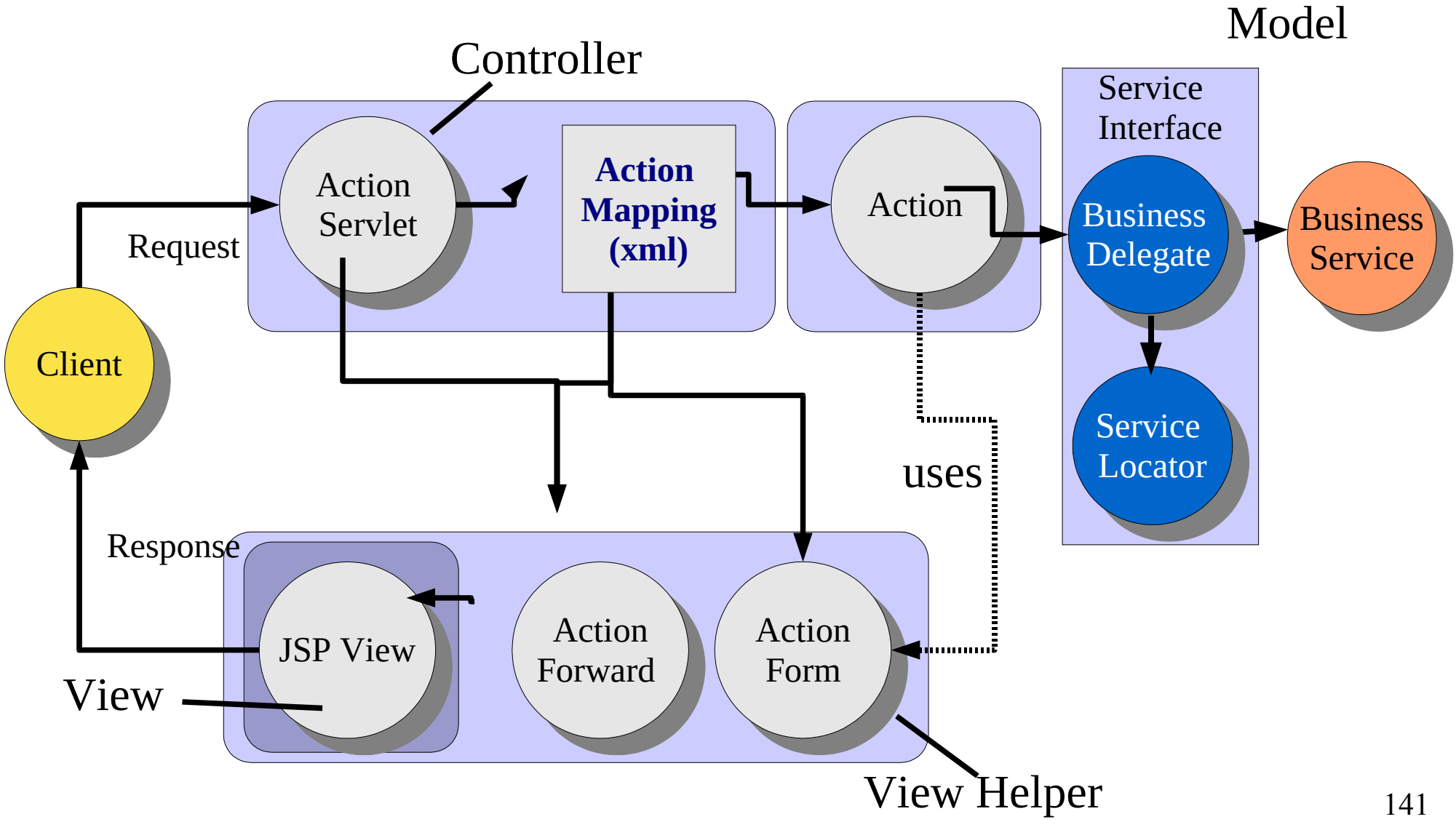
Example: Action Class

```
public ActionForward
        execute(ActionMapping mapping,
                ActionForm form,
                HttpServletRequest request,
                HttpServletResponse response) throws Exception{
 javax.sql.DataSource dataSource;
 java.sql.Connection myConnection;
 try {
  dataSource = getDataSource(request);
  myConnection = dataSource.getConnection();
  // do what you wish with myConnection
 } catch (SQLException sqle) {
    getServlet().log("Connection.process", sqle);
 } finally {
    //enclose this in a finally block to make sure the connection is closed
    try {
       myConnection.close();
    } catch (SQLException e) {
       getServlet().log("Connection.close", e);
    }
   }
}
```

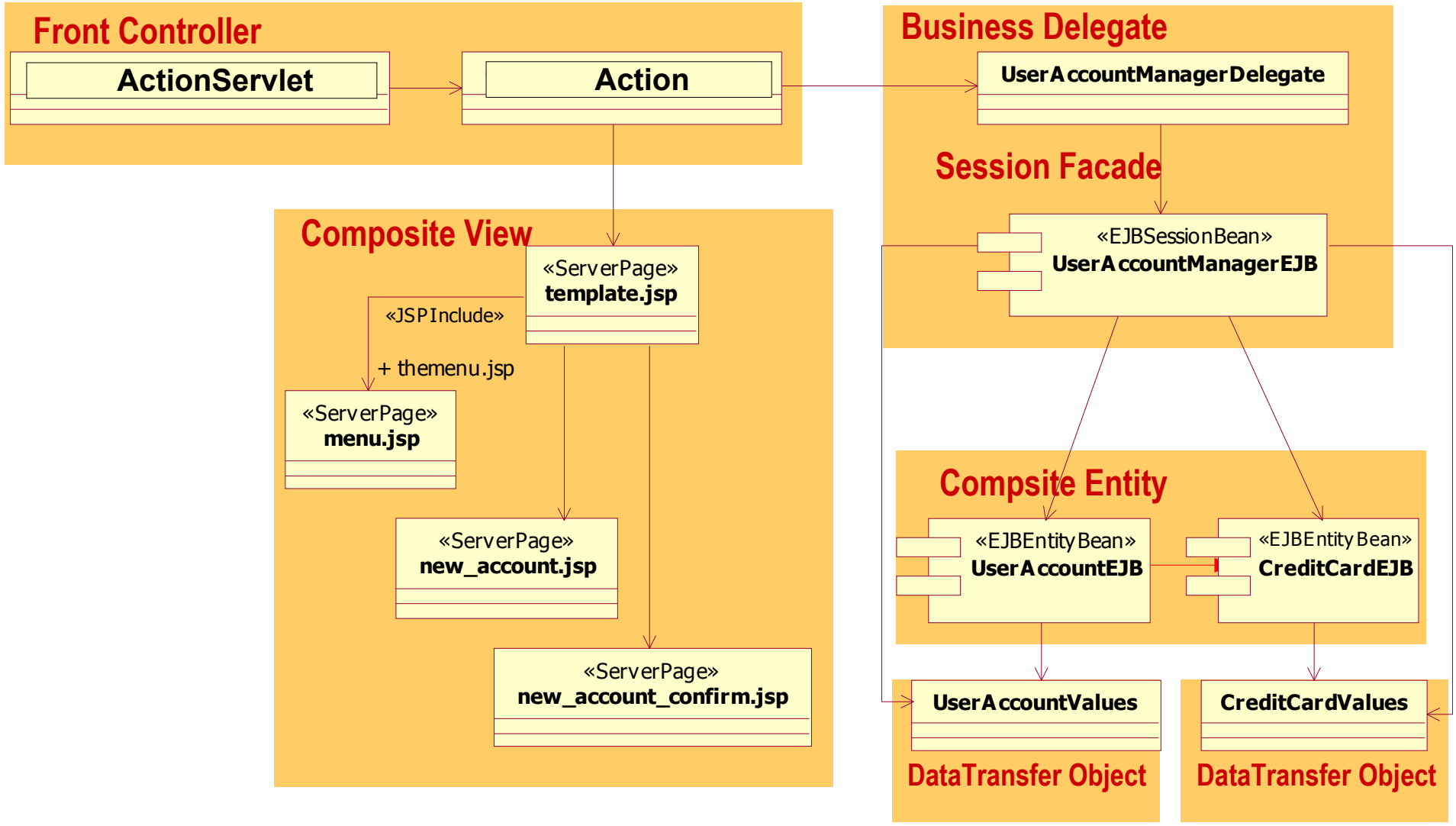
Struts and J2EE Patterns



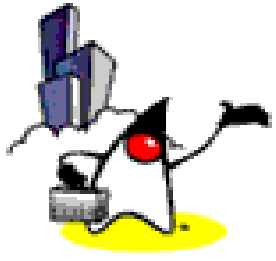
Struts and Core J2EE Patterns



Struts and Core J2EE Patterns



Nested Tag Library



The "Nested" Tag Library

- Brings a nested context to the functionality of the Struts custom tag library
- Written in a layer that extends the current Struts tags, building on their logic and functionality
  - The layer enables the tags to be aware of the tags which surround them so they can correctly provide the nesting property reference to the Struts system
  - <%@ taglib prefix="nested" uri="http://jakarta.apache.org/struts/tags-nested" %>

The "Nested" Tag Library

- Nested tags allow you to establish a default bean for nested property references
- 1:1 correspondence, and identical functionality, of other Struts tags
  - Except the "name" property gets set for you automatically ...

Example: Scenario

- Company has many departments
  - Company bean has a collection of Department beans
- Each department has many employees
  - Department bean has a collection of Employee beans

Example: Suppose You Want To Display the following...

Example: Using JSTL for Iterating Nested Beans

```
COMPANY: <html:text property="companyName"/>

<c:forEach items="${companyForm.departments}" var="dept"
   varStatus="deptStatus">

    Department: <html:text property="departments[${deptStatus.index}].
   deptName" value="${dept.deptName}" />

        <c:forEach items="${dept.employees}" var="emp" varStatus="empStatus">

            Employee: <html:text property="departments[${deptStatus.index}].
   employees[${empStatus.index}].empName" value="${emp.empName}" />

            E-mail: <html:text property="departments[${deptStatus.index}].
   employees[${empStatus.index}].email" value="${emp.email}" />

            <br/>

        </c:forEach>

</c:forEach>
```

Example: Using Nested Tag for Doing the Same Thing

```
COMPANY: <html:text property="companyName"/>
<nested:iterate property="departments">
    Department: <nested:text property="deptName"/>
        <nested:iterate property="employees">
            Employee: <nested:text property="empName"/>
            E-mail: <nested:text property="email"/>
        </nested:iterate>
</nested:iterate>
```

Example 2:

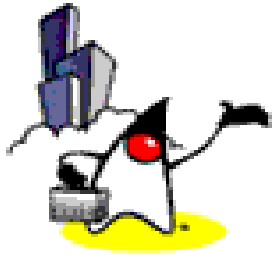
- Before using the nested tags:

```
<bean:write property="address.city"/>
<bean:write property="address.state"/>
<bean:write property="address.zip"/>
```

- After using the nested tags:

```
<nested:nest property="address">
  <nested:write property="city"/>
  <nested:write property="state"/>
  <nested:write property="zip"/>
</nested:nest>
```

Changes Between Struts 1.0 & 1.1



Struts 1.1 ActionServlet

- Introduction of request processor
  - Via RequestProcessor class
  - Handles much of the functionality of ActionServlet in Struts1.0
    - ActionServlet in 1.1 is a thin layer
  - Allows developers to customize the request handling behavior more easily

Struts 1.1 Actions

- Method perform() replaced by execute()
  - Backwards compatible
  - Change necessary to support declarative exception handling
- Addition of DispatchAction class
  - Supports multiple business methods instead of a single execute() method
  - Allows easy group of related business methods

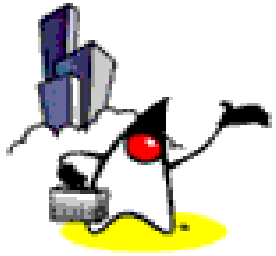
Changes to web.xml and struts-config.xml

- web.xml
  - Serveral initialization parameters removed
    - Uses elements in struts-config.xml
  - Supports a few new parameters

New Features in Struts 1.1

- Plug-ins
- Dynamic ActionForms
- Multiple application modules
- Declative exception handling
- Validator framework
- Nested tags
- Dependency to Commons projects
- Action-based authentication

Roadmap



New Features in Struts 1.2

- Some deprecations
- Validator enhancements
- DigestingPlugin added
- MappingDispatchAction added

Passion!