Département d'Informatique

# Applications Web

# Framework(s) STRUTS

**Cours IHM**

Frédéric MOAL
Université d'Orléans

Master 1ère année

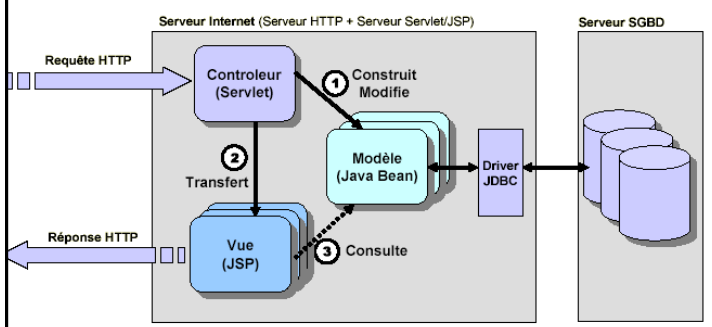
année 2011/2012

## Ressources

- Le site officiel :
  - http://struts.apache.org/
  - lib : liste des librairies
  - apps : exemples d'application (vide)
- Struts version 1, http://struts.apache.org/1.3.10/
- Struts version 2, http://struts.apache.org/2.2.3.1/
- Tutoriaux developpez.com
  - http://mbaron.developpez.com/javaee/struts/
  - http://java.developpez.com/faq/struts/
  - http://tahe.developpez.com/java/web3tier/



## Architecture MVC (2 – Web)

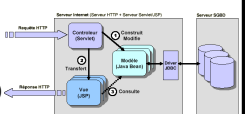





## Architecture MVC (2 – Web)

- Bonne séparation entre

les composants d'une application Web



- Facilités de développement partagés, de maintenance…
- MAIS :
  - Tâches répétitives et « sans intérêt »
    eg programmation du contrôleur
  - similarités dans les vues

**Utilisation d'un Framework**

## Architecture MVC (2 – Web)

- nombreux **frameworks** pour faciliter le développement d'applications Web

- Dans le monde Java :
- *Struts (Apache)*
- *Java Server Faces (SUN)*
- *Spring MVC*
- *Tapestry (Apache)*
- *Stripes*
- *Wicket (Apache)*
- …

- Dans d'autres langages :
  - *Symfony (PHP)*
  - *Ruby On Rails (Ruby)*
  - *Django (Python)*
  - *Grails (Groovy)*
  - …

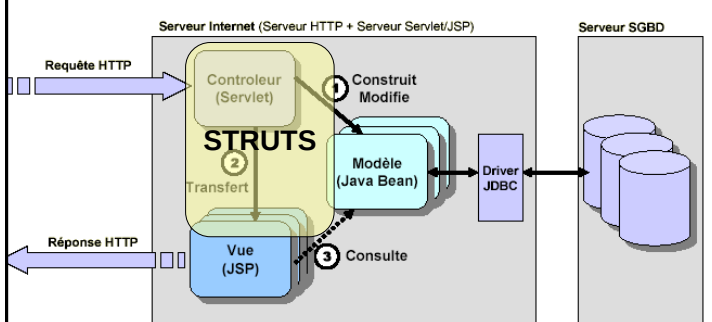


## Struts, c'est quoi ?

- Un framework Open Source (Apache) qui repose sur l'architecture MVC 2
- Le coeur du framework Struts est une couche contrôleur
  - basée sur les technologies les plus utilisées Servlet/JSP, JavaBeans, ResourceBundles, XML.
  - Struts fournit son propre composant contrôleur
- Struts offre une certaine liberté pour la conception du Modèle et de la Vue
  - Pour le Modèle, Struts peut interagir avec toutes les techniques d'accès aux données (eg Spring)
  - Pour la Vue, Struts fonctionne bien avec les JSP, les Velocity Templates, le XSLT et d'autres systèmes de présentation.

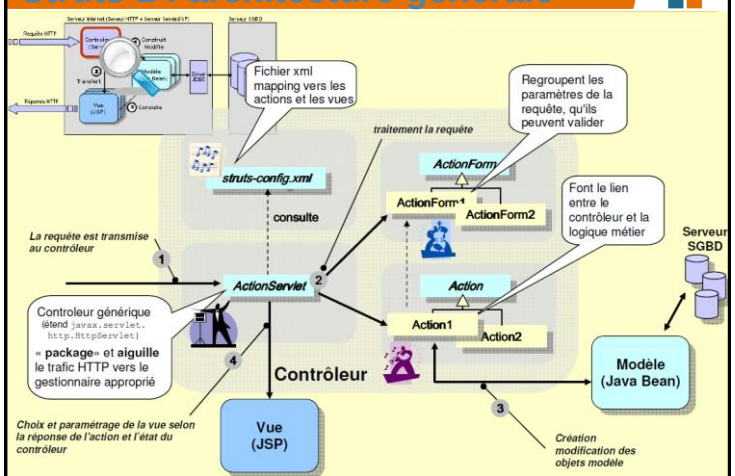


## Struts

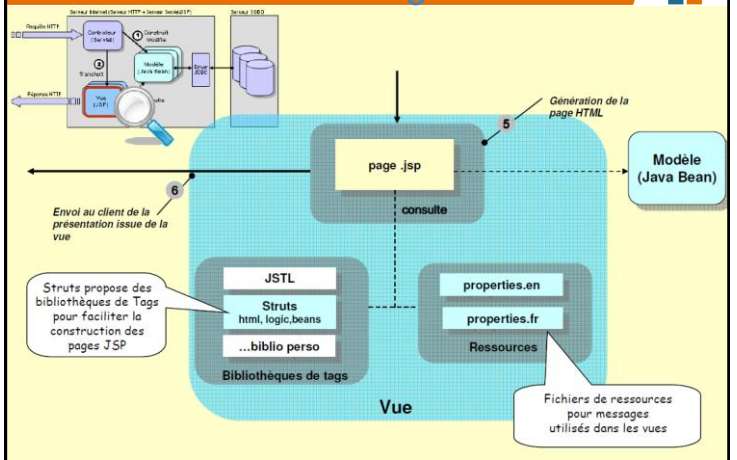





## Struts 1 : architecture générale

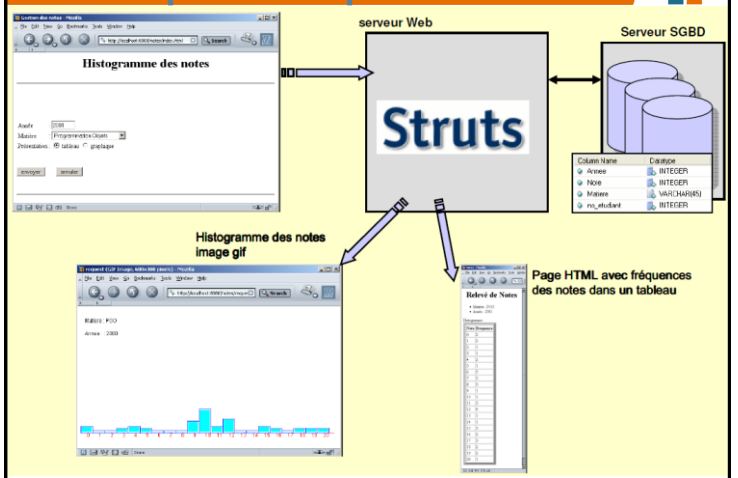


## Struts 1 : architecture générale






## Struts 1 par l'exemple






## Paramétrage : le contrôleur Struts

- Définition du contrôleur de l'application dans le fichier `web.xml`

```
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"...>
<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>
   ...
   </servlet>
   <servlet-mapping>
          <servlet-name>action</servlet-name>
          <url-pattern>*.do</url-pattern>
   </servlet-mapping>
   ...
</web-app>
```

Tout est redirigé vers le controleur Struts

Rem : Il est possible de définir PLUSIEURS contrôleurs

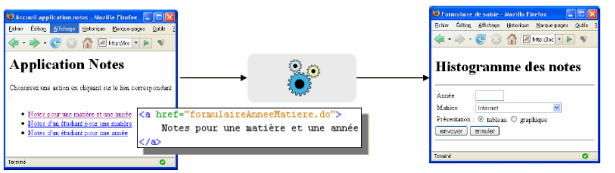


## Paramétrage : les actions

- Une action est un traitement lancé après le passage d'une requête au contrôleur
- Les actions sont décrites dans le fichier **struts-configs**
  - *dans la balise* **<action-mappings>**
  - *Au moyen de la balise* **<action>**
- différents attributs de la balise <action> sont à renseigner selon que
  - la requête avec ou sans paramètres
  - la requête nécessite un traitement ou une simple redirection

## Les actions : exemple simple

- Action correspondant à une requête sans paramètres nécessitant une simple redirection



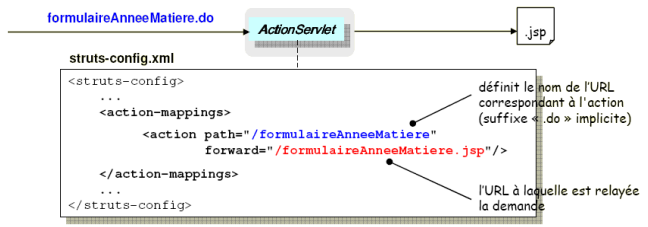

accueil.html

formulaireAnneeMatiere.jsp



## Les actions : redirection

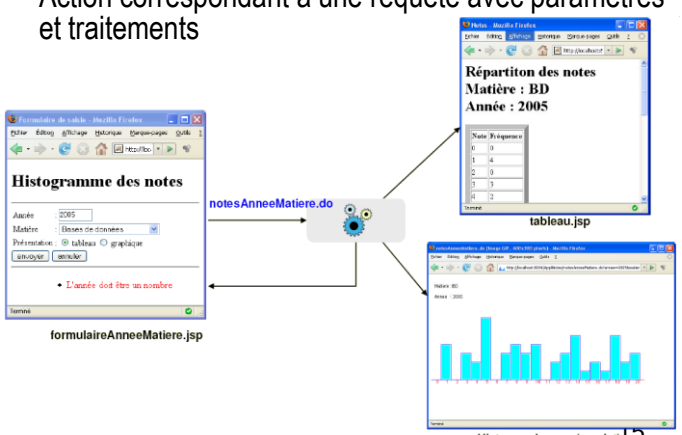
- Redirection d'un .do vers une jsp





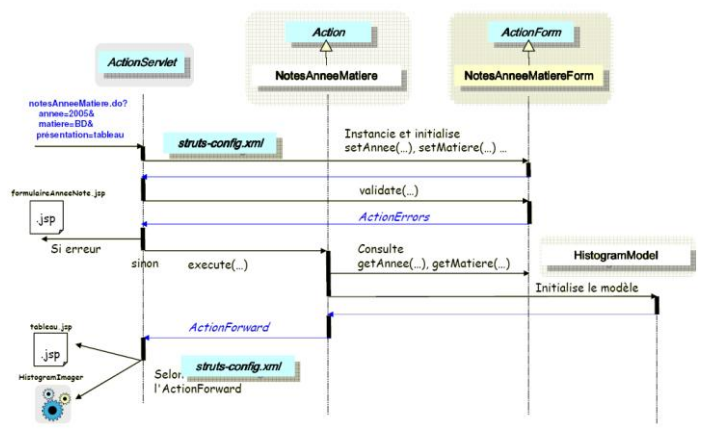

## Les actions : traitements

- Action correspondant à une requête avec paramètres et traitements






## Les actions : traitements



## ActionForm ?

- JavaBean qui permet de stocker les propriétés des formulaires
- Hérite de **org.apache.struts.action.ActionForm**
- vérifie la validité des propriétés par sa méthode **validate**
  - ***ActionErrors validate(ActionMapping, HttpServletRequest)***
  - ***ActionMapping :*** *objet image de la configuration de l'action en cours stockée dans struts-config.xml*
  - ***HttpServletRequest :*** *requête du client transmise par la Servlet de contrôle*
  - ***ActionErrors :*** *permet de retourner des messages erreurs au client*
- La classe dispose également d'autres méthodes
  - ***ActionServlet getServlet()*** *: retourne la Servlet qui gère le contrôle*
  - ***reset(ActionMapping, HttpServletRequest)*** *: initialise les propriétés*

17

## ActionForm et formulaires

```
<form action="notesAnneeMatiere.do" method="post">
  Année :   <input type="text" size="6" name="annee"/>
  Matière : <input type="text" size="4" name="matiere"/>
  Présentation :
    <input type="radio" name="presentation" value="tableau"/> tableau
    <inpu
  <input
  <input
</form>
```

```
public class NotesAnneeMatiereForm extends ActionForm {
   private String annee;
   private String presentation;
   private String matiere;

   public void setAnnee(String annee) {
     this.annee = annee;
   }
   public String getAnnee() {
     return annee;
   }
   public void setMatiere(String matiere) {
     this.matiere = matiere;
   }
   ...
}
```

## ActionForm : validation

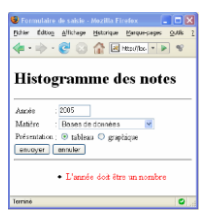
```
...
@Override
public ActionErrors validate( ActionMapping mapping,
                              HttpServletRequest request) {
 ActionErrors errors = new ActionErrors();
 if (annee == null || annee.length() < 1) {
  errors.add("Année", new ActionMessage("error.annee.required"));
 } else {
  try {
   numeroAnnee = Integer.parseInt(annee);
  } catch (NumberFormatException e) {
   errors.add("Année", new ActionMessage("error.annee.isNotANumber"));
  }
 }
 if ((!presentation.equals("graphic")) &&
     (!presentation.equals("tableau"))) {
  errors.add("Présent.", new ActionMessage("error.present.required"));
 }
  return errors;
}
```

19

## ActionForm : gestion des erreurs

```
errors.add("Année", new
   ActionMessage("error.annee.isNotANumber"));
```

```
errors.header=<UL>
errors.prefix=<LI><span style="color: red">
errors.suffix=</span></LI>
...
error.annee.required=Indiquez l'année
error.annee.isNotANumber=L'année doit être un nombre
```




```
<%@ taglib uri="/WEB-INF/tlds/tags-html.tld" prefix="html" %>
...
<html>
<body>
<H1>Histogramme des notes</H1>
<HR>
<form action="notesAnneeMatiere.do" method="post">
…
</form>
<HR>
<html:errors/>
</body>
</html>
```

## ActionForm : localisation

l'utilisation du fichier properties facilite la localisation (i18n) de l'application : un fichier de ressources par langue (locale)

ApplicationResource.properties

```
errors.header=<UL>
errors.prefix=<LI><span style="color: red">
errors.suffix=</span></LI>
...
error.annee.required=Indiquez l'année
error.annee.isNotANumber=L'année doit être un
formHisto.title=Histogramme de notes
```

ApplicationResource.properties.en

```
errors.header=<UL>
errors.prefix=<LI><span style="color: red">
errors.suffix=</span></LI>
...
error.annee.required=Year Required
error.annee.isNotANumber=Year is not a number
formHisto.title=Marks Histogram
```

```
<%@ taglib uri="/WEB-INF/tlds/tags-html.tld" prefix="html" %>
... <%@ taglib uri="/WEB-INF/tlds/struts-bean.tld" prefix="bean" %>
<html>
<body>
<H1><bean:message key="formHisto.title"/></H1>
<HR>
<form action="notesAnneeMatiere.do" method="post">
…
</form>
<HR>
<html:errors/>
</body>
</html>
```

## ActionForm : déclaration

- Les ActionForms doivent être déclarées dans le fichier **`struts-config.xml`**

```
<struts-config>
<form-beans>
  <form-bean
    name="NotesAnneeMatiereForm"
    type="miage.forms.NotesAnneeMatiereForm"/>
  ...
</form-beans>
...
<struts-config>
```

22

## Action

- Permet d'associer un traitement à une requête
- Hérite de **org.apache.struts.action.Action**
- Effectue le traitement par sa méthode **execute**

***ActionForward execute(ActionMapping, ActionForm,HttpServletRequest, HttpServletResponse)***

- ***ActionMapping*** *: objet image de la configuration de l'action en cours stockée dans struts-config.xml*
- ***ActionForm*** *: JavaBean qui stocke l'information du formulaire*
- ***HttpServletRequest*** *: référence de la requête*
- ***HttpServletResponse*** *: référence de la réponse*
- ***ActionForward*** *:objet identifiant la destination que le contrôleur (l'* ***ActionServlet****) doit choisir*

23

## Action : définition

```
public class NotesAnneeMatiere extends Action {
  public ActionForward execute(ActionMapping mapping,
     ActionForm form, HttpServletRequest request,
     HttpServletResponse response) throws Exception {
   NotesAnneeMatiereActionForm f =
     (NotesAnneeMatiereForm) form;
  HistogramModel histo =
    new HistogramModel(f.getNumeroAnnee(),f.getMatiere() );
  request.setAttribute("notes", histo);

  if (f.getPresentation().equals("graphic")) {
    return mapping.findForward("histographique");
  } else {
    return mapping.findForward("histotableau");
   }
 }
}
```

24

## Action : déclaration

- Dans le fichier struts-config.xml

```
<struts-config>
  <form-beans>
     <form-bean name="NotesAnneeMatiereActionForm"
           type="miage.forms.NotesAnneeMatiereActionForm"/>
  </form-beans>

  <action-mappings>
  <action forward="/formulaireAnneeMatiere.jsp" path="/formulaireAnneeMatiere"/>
  <action path="/notesAnneeMatiere"
     type="miage.actions.NotesAnneeMatiere"
     name="NotesAnneeMatiereForm"
     input="/formulaireAnneeMatiere.jsp"
     scope="request" >
           <forward name="histotableau" path="/tableau.jsp"/>
           <forward name="histographique" path="/histographic"/>
  </action>
</action-mappings>
```

Vue vers laquelle le contrôleur redirige en cas d'échec de la validation



## Action : utilisation

- Dans `execute` de l'action :

```
if (f.getPresentation().equals("graphic")) {
   return mapping.findForward("histotagraphique");
} else {
   return mapping.findForward("histotableau");
}
```

- Dans web.xml :

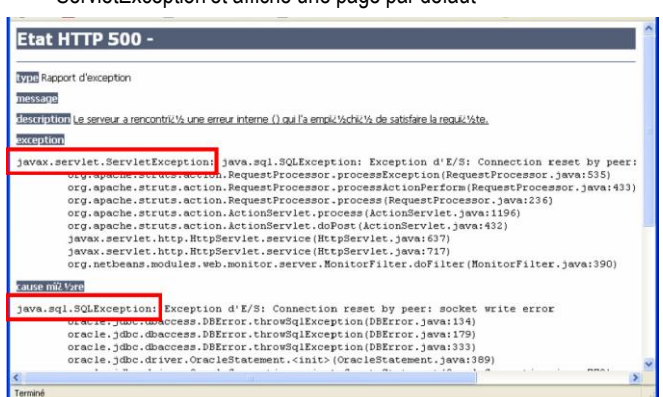
```
<servlet>
  <servlet-name>graphic</servlet-name>
  <servlet-class>
        servlets.HistogramImager
  </servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>graphic</servlet-name>
  <url-pattern>/histographic</url-pattern>
</servlet-mapping>
```



## Exceptions

- Une exception levée non traitée : le contrôleur lance une ServletException et affiche une page par défaut






## Exceptions

- Possibilité de mettre en œuvre un traitement spécifique des exceptions *en redirigeant l'application vers une page spécifique*

```
<global-exceptions>
      <exception
            key="error.messageException"
            path="/erreur_exception.jsp"
            type="miage.exceptions.DAOException"/>
…
</global-exception>
```

*Exemple : Erreur d'accès BD (DAO)*

*Avec une page JSP de traitement d'erreur classique*

## Exceptions

- Possibilité de mettre en œuvre un traitement spécifique des exceptions *en exécutant un gestionnaire d'erreur : une classe héritant de* *org.apache.struts.actions.ExceptionHandler*

```
<global-exceptions>
        <exception
                key="error.messageException"
                type="miage.exceptions.DAOException"/>
                handler="miage.exception.ExceptionHandler"
…
</global-exception>
```

*miage.exception.ExceptionHandler.execute (a redéfinir) sera exécutée à chaque levée d'exception*

- *Il est possible de redéfinir un gestionnaire local spécifique dans chaque action avec <exception … />*

29

## Struts : TagLibs

- Struts propose des bibliothèques de Tags Personnalisés qui aident les développeurs d'applications basées sur des formulaires (jsp)
- Struts propose 4 bibliothèques de tag
  - HTML : Tags pour création d'interface uilisateur HTML, en particulier pour créer des formulaires de saisie
  - Logic : Tags pour la génération conditionnelle de texte, génération répétitive de texte en itérant sur des collections d'objets, gestion du flux de contrôle de l'application
  - Bean : Tags pour la définition de nouveaux objets JavaBeans dans différentes portées (application, session, requête…) et à partir de différentes sources Tags pour afficher un bean (ou une proriétés d'un bean) sur la réponse de sortie.
  - Nested : Tags qui étendent les tags de base de Struts pour leur mise en relation lors d'imbrication

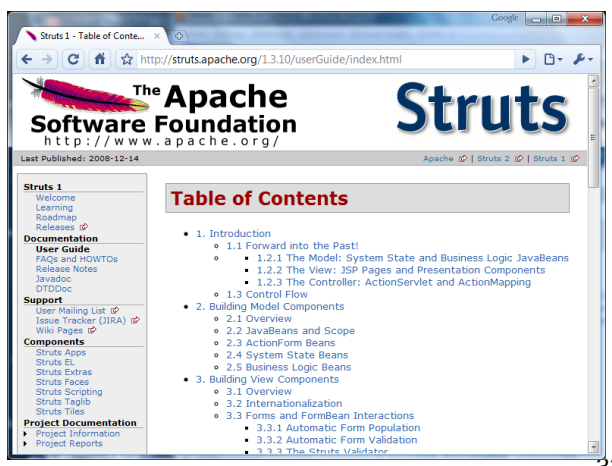
30

## Struts : exemple de Tags

- Les tags HTML Struts prennent en charge l'initialisation des éléments du formulaire en cas de retour sur erreur :

```
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
    <title>Formulaire de saisie</title>
  </head>
  <body>
    <H1>Histogramme des notes</H1>
    <HR>
    <html:form action="notesAnneeMatiere.do">
        Année : <html:text property="annee" size="6"/>
        Matière : <html:select property="matiere">
                <html:option value="AI">Internet</html:option>
                <html:option value="bd">Bases de donn&eacute;es</html:option>
                <html:option value="IHM">Interface Homme Machine</html:option>
                <html:option value="POO">Programmation Objets</html:option>
            </html:select>
        Présentation : <html:radio property="presentation" value="tableau"  /> tableau
                    <html:radio property="presentation" value="graphic"/> graphique
        <html:submit value="envoyer"/> <html:reset value="annuler"/>
    </html:form>
    <HR>
    <html:errors/>
```

## Struts : user doc en ligne




32

## Extension : DynaActionForm

- Les objets *ActionForm* sont des Beans qui permettent de stocker des propriétés statiques et de les valider
- Un constat
  - les valeurs d'un formulaire sont des chaînes de caractères : *String* pour les valeurs uniques et *String[]* pour les valeurs à champs multiples
  - il faut redéfinir à chaque fois des « get » et des « set » pour les propriétés
- La solution est d'utiliser des formulaires
  - dont la structure est déclarée dans le fichier struts-config.xml
  - qui sont créés dynamiquement par l'environnement Struts
- Réalisation
  - utilisation de la classe *DynaActionForm*
  - modification de la balise <*form-bean*>



## DynaActionForm

- La classe *DynaActionForm* possède la même méthode *validate* que *ActionForm* cependant l'accès aux attributs se fait par un objet *Map*
- Utiliser les méthodes suivantes
  - *Object get(String)* : retourne la valeur de la propriété donnée en paramètre

```
String ma_propriete = (String)this.get("nom");
```

  - *void set(String, Object)* : modifie la valeur de la propriété donnée en paramètre
- Pour chaque champ du formulaire on définit une balise <*form-property*> dans le corps de la balise <*form-bean*>
  - *String name* : le nom du champ

```
<form-bean name="nomFormulaire" type="package.DynaForm" >
<form-property name="nom" type="java.lang.String" />
<form-property name="prenom" type="java.lang.String" />
</form-bean>
```



## DynaActionForm

- Bean dynamique :

```
public class PersonneDynaActionForm extends DynaActionForm {
public ActionErrors validate(ActionMapping arg0, HttpServletRequest arg1
  ActionErrors erreurs = new ActionErrors();
  String nom = (String)this.get("nom");
  String age = (String)this.get("age");
  if (nom == null || nom.trim().equals("")) {
    erreurs.add("nomvide", new ActionMessage("formulaire.nom.vide"));
  }
  if (age == null || age.trim().equals("")) {
    erreurs.add("agevide", new ActionMessage("formulaire.age.vide"));
  } else {
  ...
  }
return erreurs;
}
}
```

```
<struts-config>
<form-beans>
<form-bean name="formPersonne" type="monpackage.PersonneDynaActionForm" >
<form-property name="age" type="java.lang.String" initial="" />
<form-property name="nom" type="java.lang.String" initial="" />
</form-bean>
</form-beans>
<action-mappings>
...
```



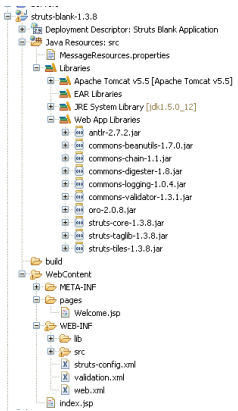
## DynaActionForm

- Bean dynamique :

```
public class FormulaireAction extends Action {
  public ActionForward execute(ActionMapping mapping, ActionForm form,
    HttpServletRequest req, HttpServletResponse res) throws Exception {
    PersonneDynaActionForm formulaire = (PersonneDynaActionForm)form;
    req.setAttribute("nom", formulaire.get("nom"));
    req.setAttribute("age", formulaire.get("age"));
    return mapping.findForward("response");
  }
}
```

- Avantage : description de la structure du Bean dans un fichier de configuration XML

## Struts et Eclipse




- Importation d'un war Struts-blank-1.3.10.war
- Contient la structure de base d'une application Struts
  - Struts-config.xml
  - Web.xml
  - JSP dans sous-répertoires pages



## STRUTS 2

Département d'Informatique

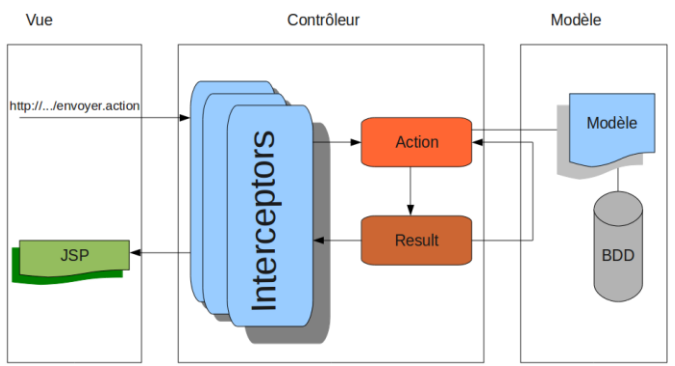
## Struts 2

- "Fusion" de deux frameworks : WebWork 2, avec Struts. But : simplifier les développement.


- Différences importantes entre Struts 1 et Struts 2



## Struts 2 : Architecture



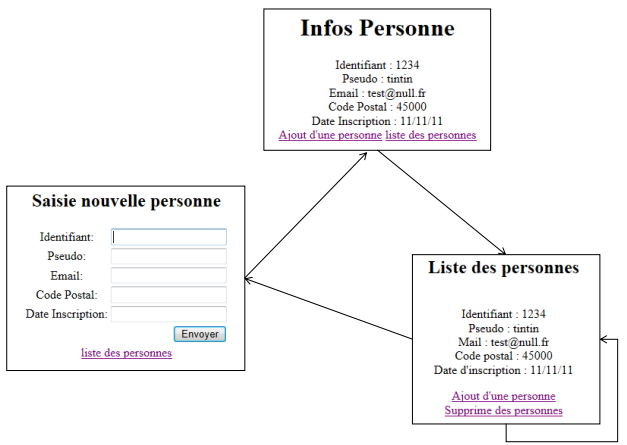
## Struts 2 : extensible par Plugins




| Struts 1 | | Struts 2 |
|---|---|---|
| Action | ⇒ | Action |
| ActionForm | ⇒ | Action or POJO's |
| ActionForward | ⇒ | Result |
| struts-config.xml | ⇒ | struts.xml |
| ActionServlet | ⇒ | FilterDispatcher |
| RequestProcessor | ⇒ | Interceptors |



## Exemple simple






## Exemple simple : web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
id="WebApp_ID" version="2.5">
  <display-name>ExempleSimple</display-name>
  <filter>
  <filter-name>struts2</filter-name>
  <filter-
class>org.apache.struts2.dispatcher.FilterDispatcher</filter-
class>
  </filter>
  <filter-mapping>
  <filter-name>struts2</filter-name>
  <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
```

## Exemple simple : struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
   "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
   <constant name="struts.enable.DynamicMethodInvocation" value="false" />
   <constant name="struts.devMode" value="false" />


<package name="exemple.actions"  namespace="/" extends="struts-default">

      <!-- Action de l'action de réference -->
       <default-action-ref name="saisir_Personne"/>

       <action name="saisir_Personne">
         <result>/jsp/saisir_Personne.jsp</result>
       </action>

…
```



## Exemple simple : struts.xml

```
      <action name="enregistrer_Personne" class="exemple.actions.PersonneAction"
            method="enregistrer">
         <result name="success" >/jsp/enregistrer_Personne.jsp</result>
         <result name="input">/jsp/saisir_Personne.jsp</result>
      </action>

      <action name="lister_Personne" class="exemple.actions.PersonneAction"
      method="lister">
      <result name="success">/jsp/lister_Personne.jsp</result>
      </action>

      <action name="supprimer_Personne" class="exemple.actions.PersonneAction"
      method="supprimer">
      <result name="success">/jsp/lister_Personne.jsp</result>
      </action>

</package>
</struts>
```



## Exemple simple : PersonneAction.java

**Bean Personne + Action :**

```
package exemple.actions;

public class PersonneAction extends ActionSupport {
        private int identifiant;
        private String pseudo;
        private String mail;
        private String codePostal;
        private java.util.Date dateInscription;

        private static  ArrayList<Personne> listPersonnes
                = new ArrayList<Personne>( );

        // getters et setters
```



## Exemple simple : PersonneAction.java

```
        public String enregistrer() {
                if(this.pseudo.equals("")) {
                        addFieldError("pseudo", "le pseudo ne doit pas être vide");
                        return "input";
                }
                Personne pers = new Personne ();
                pers.setIdentifiant(identifiant);
                pers.setPseudo(pseudo);
                pers.setMail(mail);
                pers.setCodePostal(codePostal);
                pers.setDateInscription(dateInscription);
                listPersonnes.add(pers);
                return "success";
        }
        public String lister(){
                return "success";
        }
        public String supprimer(){
                listPersonnes.removeAll(getListPersonnes());
                return "success";
        }
}
```

## Struts 2

```
<s:form action="Meeting" validate="true">
  <s:token />
  <s:textfield label="Name"    name="name" />
  <s:textfield label="Date"    name="date"/>
  <s:select    label="Invitees" name="invitees" list="employees"/>
  <s:textarea label="Description" name="description"
              rows="4" cols="50" />
  <s:submit value="Save" method="save"/>
</s:form>
```



## Quelques exemples

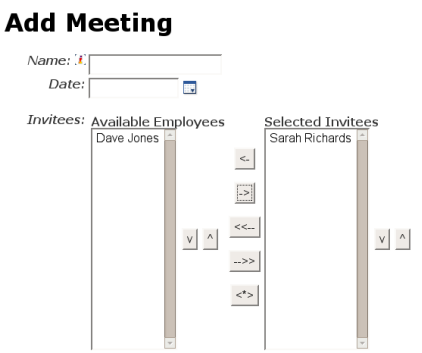



```
<s:textfield label="Name"
  name="name" tooltip="Meeting name" />
```



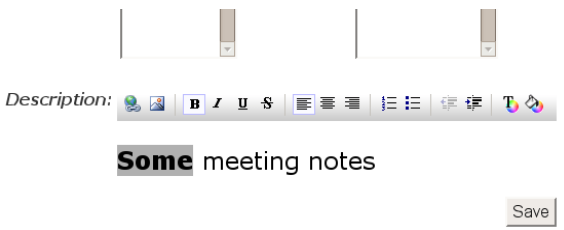

```
<s:datepicker label="Date" name="date"/>
```

<s:optiontransferselect ... />




53

```
<jsp:include
    page="/ajax/commonInclude.jsp"/>
...
<s:textarea theme="ajax" label="Description"
  name="description" rows="4" cols="50" />
```




54

## Struts 1 vs Struts 2

- Comparaison en 10 points :
  - **Action classes**
  - **Threading Model**
  - **Servlet Dependency**
  - **Testability**
  - **Harvesting Input**
  - **Expression Language**
  - **Binding values into views**
  - **Type Conversion**
  - **Validation**
  - **Control Of Action Execution**

55

## Struts 1 vs Struts 2

- **Action classes**
- Struts 1 impose aux Actions d'hériter d'une classe abstraite.
- Struts 2 Action *peuvent* implémenter l'interface Action, et/ou d'autres interfaces. Struts 2 fournit une classe de base ActionSupport d'implémentation.

  Sinon, utilisation d'objet POJO.

56

## Struts 1 vs Struts 2

- **Threading Model**
- Struts 1 Actions are singletons and must be thread-safe since there will only be one instance of a class to handle all requests for that Action. The singleton strategy places restrictions on what can be done with Struts 1 Actions and requires extra care to develop. Action resources must be thread-safe or synchronized.
- Struts 2 Action objects are instantiated for each request, so there are no thread-safety issues. (In practice, servlet containers generate many throw-away objects per request, and one more object does not impose a performance penalty or impact garbage collection.)

57

## Struts 1 vs Struts 2

- **Servlet Dependency**
- Struts 1 Actions have dependencies on the servlet API since the HttpServletRequest and HttpServletResponse is passed to the execute method when an Action is invoked.
- Struts 2 Actions are not coupled to a container. Most often the servlet contexts are represented as simple Maps, allowing Actions to be tested in isolation. Struts 2 Actions can still access the original request and response, if required. However, other architectural elements reduce or eliminate the need to access the HttpServetRequest or HttpServletResponse directly.

58

## Struts 1 vs Struts 2

- **Testability**
- A major hurdle to testing Struts 1 Actions is that the execute method exposes the Servlet API. A third-party extension, Struts TestCase, offers a set of mock object for Struts 1.
- Struts 2 Actions can be tested by instantiating the Action, setting properties, and invoking methods. Dependency Injection support also makes testing simpler.

.

59

## Struts 1 vs Struts 2

- **Harvesting Input**
- Struts 1 uses an ActionForm object to capture input. Like Actions, all ActionForms must extend a base class. Since other JavaBeans cannot be used as ActionForms, developers often create redundant classes to capture input. DynaBeans can used as an alternative to creating conventional ActionForm classes, but, here too, developers may be redescribing existing JavaBeans.
- Struts 2 uses Action properties as input properties, eliminating the need for a second input object. Input properties may be rich object types which may have their own properties. The Action properties can be accessed from the web page via the taglibs. Struts 2

60

## Struts 1 vs Struts 2

- **Expression Language**
- Struts 1 integrates with JSTL, so it uses the JSTL EL. The EL has basic object graph traversal, but relatively weak collection and indexed property support.
- Struts 2 can use JSTL, but the framework also supports a more powerful and flexible expression language called "Object Graph Notation Language" (OGNL).

61

## Struts 1 vs Struts 2

- **Binding values into views**
- Struts 1 uses the standard JSP mechanism for binding objects into the page context for access.
- Struts 2 uses a "ValueStack" technology so that the taglibs can access values without coupling your view to the object type it is rendering. The ValueStack strategy allows reuse of views across a range of types which may have the same property name but different property types.

62

## Struts 1 vs Struts 2

- **Type Conversion**
- Struts 1 ActionForm properties are usually all Strings. Struts 1 uses Commons-Beanutils for type conversion. Converters are per-class, and not configurable per instance.
- Struts 2 uses OGNL for type conversion. The framework includes converters for basic and common object types and primitives.

63

## Struts 1 vs Struts 2

- **Validation**
- Struts 1 supports manual validation via a validate method on the ActionForm, or through an extension to the Commons Validator. Classes can have different validation contexts for the same class, but cannot chain to validations on sub-objects.
- Struts 2 supports manual validation via the validate method and the XWork Validation framework. The Xwork Validation Framework supports chaining validation into sub-properties using the validations defined for the properties class type and the validation context.

64

## Struts 1 vs Struts 2

- **Control Of Action Execution**
- Struts 1 supports separate Request Processors (lifecycles) for each module, but all the Actions in the module must share the same lifecycle.
- Struts 2 supports creating different lifecycles on a per Action basis via Interceptor Stacks. Custom stacks can be created and used with different Actions, as needed.

65