



Java pour le développement d'applications Web : Java EE

Struts



Mickaël BARON - 2006

<mailto:baron.mickael@gmail.com> ou <mailto:mickael.baron@serli.com>

- SERLI : www.serli.com
 - Société de services en informatique
 - Fondé en 1981
 - Située sur le site du Futuroscope, Poitiers
 - Réalisation de logiciels et assistance technique
- Domaines de compétences
 - Systèmes d'informations
 - Embarqué et temps réel
 - Systèmes et réseaux
 - Gestion Electronique de Document (GED, PDM / PLM)
- Plateformes de développement (celles qui concernent les cours)
 - Plateforme Java : Java EE, Java SE, Eclipse
 - Plateforme Microsoft : C#, VB .NET
 - ...



- Equipes impliquées dans l'Open Source
 - Utilisation massive de briques Open Source
 - Formation / diffusion de cours concernant Java et l'Open Source
 - RMLL : 7^{ème} rencontres mondiales du logiciel libre
 - Solutions Linux 2007 : Salon des Solutions Open Source pour l'entreprise
- Membre du consortium ObjectWeb
- Gestion de projets Open Source
 - JaasLounge : interopérabilité JAAS pour Java EE
 - JShaft : gestion de clusters Java EE
 - JWT Gen : tests fonctionnels automatisés



Struts : qu'est-ce-que c'est ...

- La bibliothèque Struts est un framework qui permet de construire des applications web respectant le modèle d'architecture MVC
- C'est une méthode de développement qui fournit le minimum de règles pour construire une application web professionnelle (séparation des métiers et donc des compétences)
- Struts est un projet soutenu par l'Apache Software Foundation. Des informations complémentaires dans *struts.apache.org*
- Logique de fonctionnement
 - la structure de l'application web est décrite dans un fichier *struts-config.xml*
 - l'utilisation de Servlets est transparente via des classes adaptées
 - les pages JSP exploitent des balises personnalisées de Struts. Il est conseillé d'utiliser en parallèle la bibliothèque JSTL

Struts : utile ou pas ?

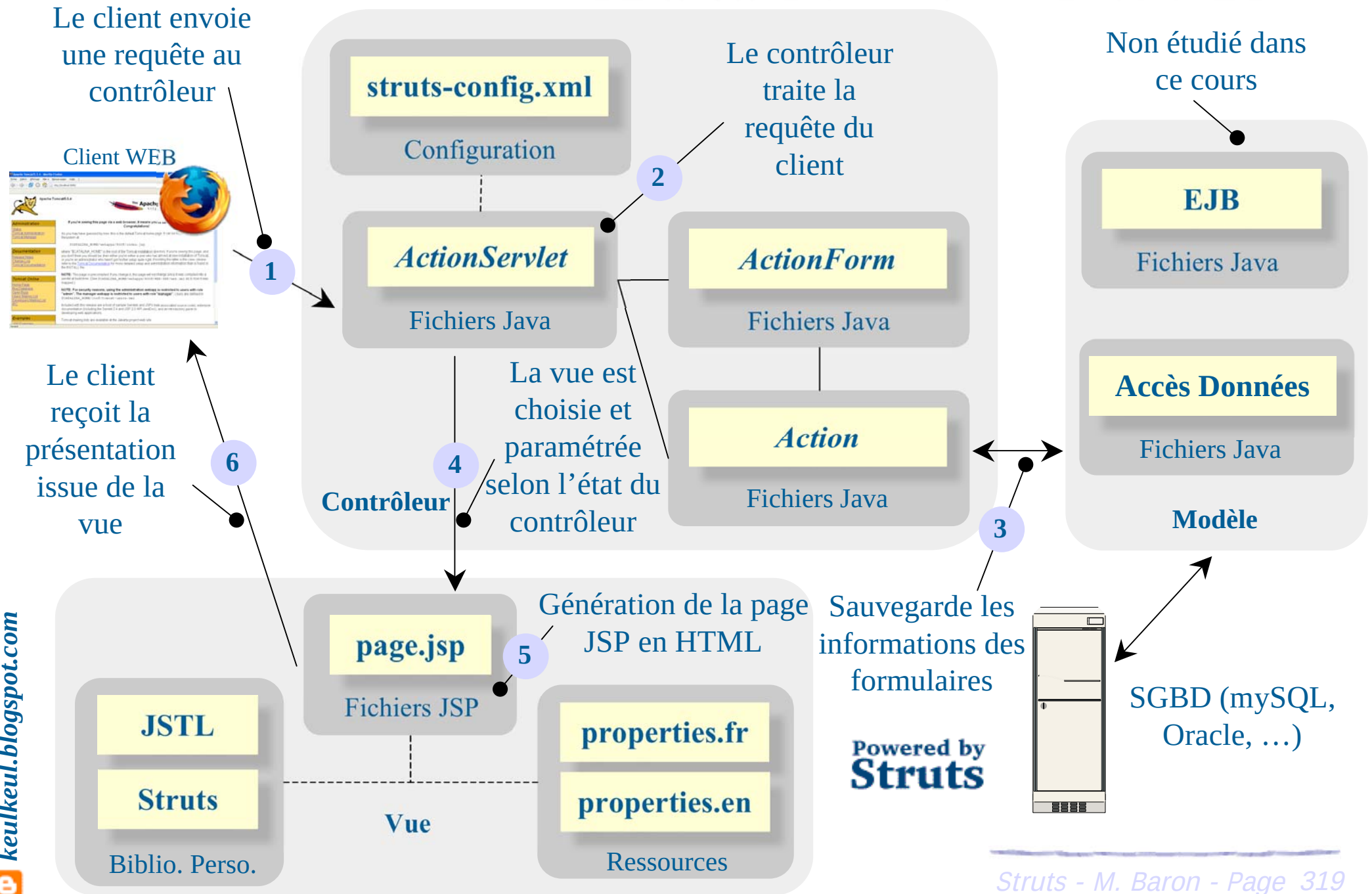
- Struts est un framework assez « lourd » pour une simple application (un formulaire et une réponse par exemple)
- Il introduit un niveau de complexité et de mise en route importants et les bénéfices de son utilisation se font ressentir dès que l'application prends de l'importance
- Sur le web de nombreuses formations quant à son utilisation montrent son importance dans les projets industriels
- Une nouvelle bibliothèque est en passe de supplanter Struts : Java Server Faces (JSF) mais trop récente et pas assez mûre pour en discuter dans ce cours (à vérifier ...)

Struts : documentation ...

- De nombreuses ressources sont disponibles concernant Struts
 - La FAQ de Developpez.com : java.developpez.com/faq/struts
 - Struts par l'exemple : taha.developpez.com
 - Jakarta Struts - précis & consis (O'Reilly), ...
- Les nombreuses ressources présentent principalement la version 1.1 de la bibliothèque de Struts (incompatibilité avec la 1.2)
- Téléchargement de Struts à struts.apache.org/download.cgi
- Présentation basée sur un exemple
 - un formulaire qui demande la saisie d'un nom et d'un age
 - contraintes d'intégrité : présence des paramètres, age est un entier positif
 - différentes vues : erreurs, formulaire, validation

Powered by
Struts

Struts : principe générique de la méthode



Struts : principe générique de la méthode : Contrôleur

- Le contrôleur est le cœur de l'application web. Toutes les demandes du client transitent par lui
- Il est défini par une Servlet générique de type *ActionServlet* fournie par l'API de Struts
- Le contrôleur prend les informations dont il a besoin dans le fichier *struts-config.xml*
- Si la requête du client contient des paramètres, ceux-ci sont transmis dans un objet de type *ActionForm*
- Selon l'état retourné par l'*ActionForm* précédent, le contrôleur traite une action spécifique par un objet de type *Action*

Struts : intégration dans l'application web

- De manière à intégrer le framework Struts dans une application web, il est nécessaire d'enrichir le fichier *web.xml*
- Par principe le contrôleur Struts est atteint par toutes les URL's se terminant par le suffixe « .do »

```
...  
<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>  
...
```

Le contrôleur est défini par la Servlet générique *ActionServlet*

En paramètre de la Servlet le fichier *struts-config.xml*

Toute URL terminant par « .do » est traitée par le contrôleur

Possibilité de définir plusieurs *ActionServlet* pour une même application Web

Struts : le fichier configuration struts-config.xml

- Le fichier gérant la logique de l'application web s'appelle par défaut *struts-config.xml*
- Il est placé dans le répertoire WEB-INF au même niveau que *web.xml*
- Il décrit essentiellement trois éléments :
 - les objets Bean associés aux formulaires JSP (*ActionForm*)
 - les actions à réaliser suite aux résultats des objets *ActionForm* (*Action*)
 - les ressources éventuelles suites à des messages
- Le fichier de configuration est un fichier XML décrit par une DTD. La balise de départ est `<struts-config>`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
    "http://struts.apache.org/dtds/struts-config_1_2.dtd">
```

```
<struts-config>
    ...
</struts-config>
```

Description de fonctionnement de
l'architecture de l'application web

Struts : Action

- Une action est un traitement obtenu suite au passage de la requête au contrôleur
- Nous distinguons deux sortes de requête client
 - requête sans paramètre issue par exemple d'une re-direction
 - requête avec paramètres issue par exemple d'un formulaire
- Les actions sont décrites dans la balise `<action-mappings>` au moyen de la balise `<action>`
- Selon le type de requête (avec ou sans paramètre) différents attributs de la balise `<action>` sont à renseigner

Struts : Action

- Dans le cas d'une requête sans paramètre le rôle du contrôleur est de relayer la demande du client à une URL
- La balise `<action>` dispose alors des attributs suivants
 - *String path* : définit le nom de l'URL (suffixe « .do » implicite)
 - *String type* : définit le nom de la classe *Action* qui doit traiter la demande. Utilisez la classe *org.apache.struts.actions.ForwardAction* dans ce cas précis de re-direction
 - *String parameter* : le nom de l'URL à qui doit être relayée la demande

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
    "http://struts.apache.org/dtds/struts-config_1_2.dtd">
<struts-config>
  <action-mappings>
    <action
      path="/monnom"
      parameter="/vues/mapage.jsp"
      type="org.apache.struts.actions.ForwardAction" />
  </action-mappings>
</struts-config>
```

Quand le client transmet l'URL « ../monnom.do »
au contrôleur celui-ci redirige vers
« /vues/mapage.jsp »

Il s'agit d'une re-direction

Struts : Action

➤ Épisode 1 : appelle du formulaire de saisie du nom et de l'âge

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
    "http://struts.apache.org/dtds/struts-config_1_2.dtd">
<struts-config>
    <action-mappings>
        <action
            path="/formulaire"
            parameter="/vues/formulaire.jsp"
            type="org.apache.struts.actions.ForwardAction" />
    </action-mappings>
</struts-config>
```

Le formulaire est
défini dans la
page
« formulaire.jsp »

Personne - Formulaire

Nom

Age

Envoyer Retablir Effacer

Terminé

Le client envoie la requête
suivante

C'est la page formulaire.jsp
qui est retournée au client

Struts : Action

- Dans le cas d'une requête avec paramètres le rôle du contrôleur est double
 - transmettre les informations dans un objet Bean de type *ActionForm*
 - réaliser une action spécifique (autre qu'une simple redirection)
- La balise `<action>` dispose en plus des attributs déjà étudiés des attributs suivants
 - *String scope* : les valeurs du formulaire sont stockées en session
 - *String name* : référence le nom d'une section `<form-bean>` (voir ci-après)
 - *String validate* : indique si la méthode *validate* de l'objet *ActionForm* doit être appelée ou non (voir ci-après)
 - *String input* : indique la vue qui sera appelée s'il y a erreur dans l'objet *ActionForm*

Struts : Action

- Les formulaires sont déclarés dans la balise `<form-beans>` au moyen de la balise `<form-bean>`
- La balise `<form-bean>` possède les attributs suivants
 - *String name* : nom du formulaire de la page JSP
 - *String type* : classe de type *ActionForm* qui stocke les paramètres du Bean

```
<?xml version="1.0" encoding="ISO-8859-1"?>
...
<struts-config>
  <form-beans>
    <form-bean name="monformulaire" type="monpackage.ClassActionForm" />
  </form-beans>
  <action-mappings>
    <action
      path="/monnom"
      name="monformulaire"
      scope="session"
      validate="true"
      input="/pageerreurs.do"
      parameter="/vues/mapage.jsp"
      type="org.apache.struts.actions.ForwardAction" />
    </action-mappings>
  </struts-config>
```

Quand le client transmet l'URL
« ../monnom.do » au contrôleur
celui-ci redirige vers
« /vues/mapage.jsp » si aucun
problème

Dans le cas où les paramètres sont mauvais le
contrôleur redirige vers « /pageerreurs.do »

Struts : Action

➤ Épisode 2 : envoi d'une requête de type POST du formulaire

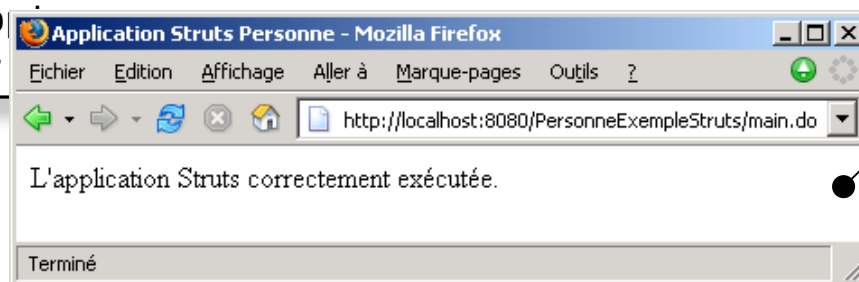
Trois actions et un formulaire sont actuellement définis

Le formulaire est défini par
« formPersonne »

```
...  
<struts-config>  
  <form-beans>  
    <form-bean name="formPersonne" type="monpackage.FormulaireBean" />  
  </form-bean>  
  
  <action-mappings>  
    <action  
      path="/main" name="formPersonne" scope="session" validate="true"  
      input="/erreurs.do" parameter="/vues/main.jsp"  
      type="org.apache.struts.actions.ForwardAction" />  
    <action  
      path="/formulaire"  
      parameter="/vues/formulaire.jsp"  
      type="org.apache.struts.actions.ForwardAction" />  
    <action  
      path="/erreurs"  
      parameter="/vues/erreurs.jsp"  
      type="org.apache.struts.actions.ForwardAction" />  
  </action-map>  
</struts-config>
```

Les valeurs sont stockées dans
monpackage.FormulaireBean

Si les données sont
correctes direction
« /vues/main.jsp »
sinon direction
« /erreurs.do »



Struts : Action

➤ Épisode 2 (suite) : envoie d'une requête issue du formulaire

```
<%@ taglib uri="htmlstruts" prefix="html" %>
...
<body>
<center>
<h2>Personne - Formulaire</h2><hr>
<html:form action="/main" >
<table>
  <tr>
    <td>Nom</td>
    <td><html:text property="nom" size="20" /></td>
  </tr>
  <tr>
    <td>Age</td>
    <td><html:text property="age" size="3"/></td>
  </tr>
</table>
<table>
  <tr>
    <td><html:submit value="Envoyer" /></td>
    <td><html:reset value="Retablir" /></td>
    <td><html:button property="btnEffacer" value="Effacer" onclick="effacer()" /></td>
  </tr>
</table>
</html:form>
...
```

Bibliothèque de balises personnalisées Struts:HTML

L'action du formulaire est d'appeler la ressource « /main » associée

Les deux paramètres transmis en paramètre de la requête

?

i

Nous reviendrons plus tard sur la compréhension de la bibliothèque « struts-html »

Struts : *ActionForm*

- L'objectif d'un objet de type *ActionForm* est de stocker les informations issues d'un formulaire
- Les objets de type *ActionForm* sont construits comme un objet Bean
Utilisation du principe de la réflexivité
- La classe Bean devra donc hériter de la classe *ActionForm* du package *org.apache.struts.action*
- C'est le contrôleur via la Servlet qui se charge de créer les instances des objets de type *ActionForm*
- Pour chaque propriété, le Bean doit définir un attribut et deux méthodes
 - un modifieur pour affecter une valeur à l'attribut
 - un accesseur pour obtenir la valeur de l'attribut correspondant

L'instanciation d'un objet de type *ActionForm* est implicite. L'information est donnée dans *struts-config.xml*



Struts : **ActionForm**

- Hormis le but de stocker les propriétés des formulaires, les objets de type *ActionForm* s'occupe aussi de l'aspect sémantique des données
- La méthode *validate* s'occupe de vérifier la validité des attributs de l'objet Bean
- *ActionErrors validate(ActionMapping, HttpServletRequest)*
 - le paramètre *ActionMapping* est un objet image de la configuration de l'action en cours stockée dans *struts-config.xml*
 - le paramètre *HttpServletRequest* est la requête du client transmise par la Servlet de contrôle
 - le retour *ActionErrors* permet de retourner des messages erreurs au client
- La classe *ActionForm* dispose également d'autres méthodes
 - *ActionServlet getServlet()* : retourne la Servlet qui gère le contrôle
 - *reset(ActionMapping, HttpServletRequest)* : initialise les propriétés

Struts : ActionForm

- Un objet de type *ActionMapping* permet d'extraire les informations contenu dans le fichier `struts-config.xml`
- Il possède des méthodes associées

```
...  
<action  
  path="/main" name="formPersonne" scope="session" validate="true"  
  input="/erreurs.do" parameter="/vues/main.jsp"  
  type="org.apache.struts.actions.ForwardAction " />  
...
```

- *String getType()* : pour accéder au contenu de l'attribut *type*
- *String getInput()* : pour accéder au contenu de l'attribut *input*

- Un objet *ActionErrors* permet d'ajouter des erreurs et l'ajout se fait par la méthode
 - *add(String, ActionMessage)* : où le premier paramètre correspond à la clé et le second au message d'erreur

Struts : ActionForm

➤ Épisode 3 : stocker les informations du formulaire

```
public class FormulaireBean extends ActionForm {  
    private String nom = null;  
    private String age = null;  
  
    public String getNom() {  
        return nom;  
    }  
  
    public void setNom(String nom){  
        this.nom = nom;  
    }  
  
    public void setAge(String age) {  
        this.age = age;  
    }  
  
    public String getAge() {  
        return age;  
    }  
    ...  
}
```

Les deux attributs
modélisant les propriétés
du Bean

Les modifieurs et
accesseurs pour traiter et
modifier les propriétés

La classe du
framework
Struts qui gère
les Beans
associés aux
formulaires

Struts : ActionForm

➤ Épisode 3 (suite) : stocker et valider les informations du formulaire

Au début erreurs
est vide donc pas
d'erreur

```
public class FormulaireBean extends ActionForm {
    ... // Lié à la modélisation des propriétés
    public ActionErrors validate(ActionMapping arg0, HttpServletRequest arg1) {
        ActionErrors erreurs = new ActionErrors();
        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 {
            try {
                int mon_age_int = Integer.parseInt(age);
                if (mon_age_int < 0) {
                    erreurs.add("ageincorrect",
                        new ActionMessage("formulaire.age.incorrect"));
                }
            } catch (Exception e) {
                erreurs.add("ageincorrect",
                    new ActionMessage("formulaire.age.incorrect", age));
            }
        }
        return erreurs;
    }
}
```

Ajout des erreurs
selon « l'arrivage »

Depuis la nouvelle version 1.2, il faut utiliser
ActionMessage et non *ActionError* (désapprouvée)



Struts : ActionForm et ActionErrors

- Les messages d'erreurs stockés dans un objet *ActionErrors* et retourner par la méthode *validate* sont transmis au contrôleur
- Si l'attribut *validate* vaut « true » et que l'objet *ActionErrors* n'est pas *null* le contrôleur redirige vers la vue de l'attribut *input*

```
...  
<action  
  path="/main" name="formPersonne" scope="session" validate="true"  
  input="/erreurs.do" parameter="/vues/main.jsp"  
  type="org.apache.struts.actions.ForwardAction " />  
...
```

- Les erreurs sont affichées dans la vue JSP au moyen de la balise personnalisée *<errors>* de la bibliothèque Struts-HTML

```
<%@ taglib uri="htmlstruts" prefix="html" %>  
<html:errors/>
```

La balise *<errors>* n'affiche pas les messages mais des identifiants présents dans un fichier ressource qui doit être référencé dans *struts-config.xml*

Struts : ActionForm et ActionErrors

- Pour déclarer un fichier ressource dans le fichier configuration struts-config.xml utiliser la balise `<message-resources>`
 - *String parameter* : nom du fichier ressource
 - *boolean null* : *true* affiche *null*, *false* affiche *???key???*
 - *String key* : à utiliser quand il y a plusieurs fichiers ressources

De préférence à la fin du fichier struts-config.xml
- Le fichier ressource doit porter comme extension « .properties »
 - Exemple de fichier : toto.properties
- Le fichier ressource doit être placé obligatoirement dans un sous-répertoire de /WEB-INF/classes. Exemples :
 - /WEB-INF/classes/toto.properties
 - /WEB-INF/classes/

Les fichiers ressources sont obligatoires quand vous utilisez la balise `<errors>` dans une page JSP
- Pour choisir le fichier ressource utilisez l'attribut *bundle* dans la balise `<errors>` en indiquant le nom de la clé

Struts : ActionForm et ActionErrors

➤ Épisode 4 : gérer les erreurs sémantiques du formulaire

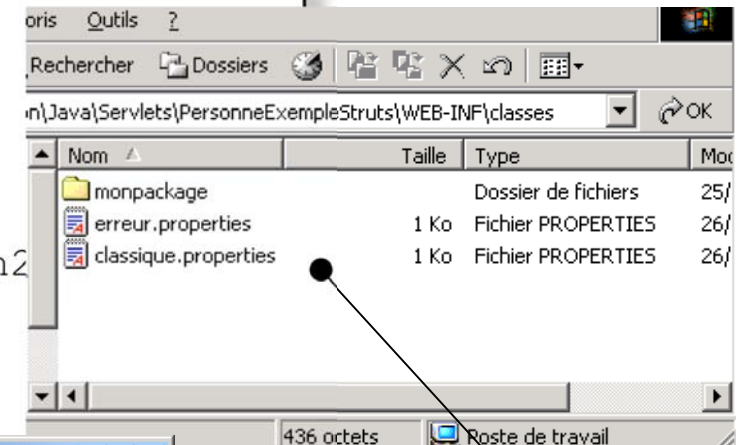
```
...  
<message-resources parameter="erreur" null="false" key="erreur" />  
<message-resources parameter="classique" null="false" key="classique" />  
</struts-config>
```

Fichier

« erreur.properties »

```
formulaire.nom.vide=<li>Vous devez indiquer un nom</li>  
formulaire.age.vide=<li>Vous devez indiquer un age</li>  
formulaire.age.incorrect=<li>L'age [{0}] est incorrect</li>  
errors.header=<ul>  
errors.footer=</ul>
```

```
<%@ taglib uri="/WEB-INF/tlds/struts-  
<html>  
  <head>  
    <title>Personne</title>  
  </head>  
  <body>  
    <h2>Les erreurs suivantes se sont produites</h2>  
    <html:errors bundle="erreur" />  
    <html:link page="/formulaire.do">  
      Retour au formulaire  
    </html:link>  
  </body>
```



Nom	Taille	Type	Mod
monpackage		Dossier de fichiers	25/
erreur.properties	1 Ko	Fichier PROPERTIES	26/
classique.properties	1 Ko	Fichier PROPERTIES	26/



Emplacement des
fichiers properties



Struts : Action

- Nous avons pour l'instant utilisé simplement la classe *ForwardAction* qui ne permet que de traiter des re-directions sans de réels effets de bord

```
...  
<action  
  path="/main" name="formPersonne" scope="session" validate="true"  
  input="/erreurs.do" parameter="/vues/main.jsp"  
  type="org.apache.struts.actions.ForwardAction " />  
...
```

- De manière à pouvoir réaliser des actions plus complexes (modification du modèle, création de nouveaux Bean, ...) nous dérivons explicitement la classe *Action*
- Cette classe possède la méthode *execute* appelée par le constructeur de l'application web si aucune erreur ne s'est produite

Struts : Action

- *ActionForward* execute(*ActionMapping*, *ActionForm*, *HttpServletRequest*, *HttpServletResponse*)
 - le paramètre *ActionMapping* est un objet image de la configuration de l'action en cours stockée dans struts-config.xml
 - le paramètre *ActionForm* correspond au Bean qui stocke l'information du formulaire
 - le paramètre *HttpServletRequest* est la référence de la requête
 - le paramètre *HttpServletResponse* est la référence de la réponse
 - le retour *ActionForward* est un objet pour identifier la destination prochaine que le contrôleur choisira
- Il faut modifier également struts-config.xml en ajoutant au corps de la balise *<action>* la balise *<forward>*
 - *String name* : étiquette pour la re-direction
 - *String path* : chemin de re-direction

Struts : Action

➤ Épisode 5 : améliorer le traitement des actions du contrôleur

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

Grâce au paramètre
ActionForm on a accès
au contenu du Bean

L'objet requête de la Servlet
est modifié en ajoutant deux
attributs issus du Bean

On indique ici que la
prochaine re-direction se fera
dans « response »

```
...  
<action  
    path="/main" name="formPersonne" scope="session" validate="true"  
    input="/erreurs.do" parameter="/vues/main.jsp"  
    type="monpackage.FormulaireAction">  
        <forward name="response" path="/reponse.do" />  
</action>  
...
```

Ajout dans le corps
de cette action de
la balise
<forward>

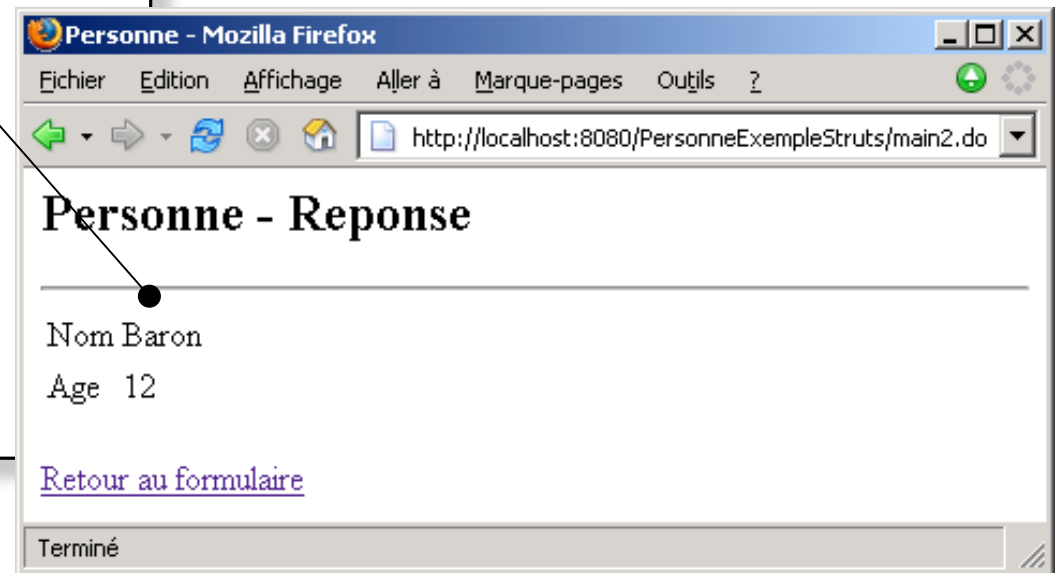
L'étiquette « response » indique
une nouvelle page cible

Struts : Action

➤ Épisode 5 (suite) : réponse positive

```
<%@ taglib uri="htmlstruts" prefix="html" %>
<html>
  <head>
    <title>Personne</title>
  </head>
  <body>
    <h2>Personne - Reponse</h2><hr>
    <table>
      <tr>
        <td>Nom</td><td>${nom}</td>
      </tr>
      <tr>
        <td>Age</td><td>${age}</td>
      </tr>
    </table>
    <br>
    <html:link page="/formulaire.do">
      Retour au formulaire
    </html:link>
  </body>
</html>
```

Utilisation des EL dans la page JSP
puisque deux attributs ont été définis
dans la classe *FormulaireAction*
(scope = request)



La balise `<link>` permet de retourner
facilement un lien hypertexte

Struts : 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>`

Struts : DynaActionForm

- La classe *DynaActionForm* possède la même méthode *validate* que *ActionForm* cependant l'accès aux attributs se fait par un objet *Map*



**Ne pas oublier de
« Caster » l'objet en retour**

- 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

- *String type* : son type Java

Deux propriétés ont été définies dans le formulaire « nomFormulaire » traité par le Bean dynamique « DynaForm »

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


Struts : DynaActionForm

► Épisode 6 : utilisation d'un Bean dynamique

Le 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;
    }
}
```

Chaque propriété est extraite par un identifiant

Le reste du code est identique au code fourni par l'*ActionForm* précédente

```
<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>
    ...
  </action-mappings>
</struts-config>
```

Déclaration des deux propriétés

Le fichier <struts-config.xml>

Struts : DynaActionForm

➤ Épisode 6 (suite) : utilisation d'un 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");  
    }  
}
```

Il faut modifier
en conséquence
l'action associée

Cette solution offre
l'avantage de décrire la
structure du Bean par une
description XML



Struts : Validator : des constats ...

- En utilisant la solution *DynaActionForm* nous sommes encore obligés d'écrire le code Java correspondant aux contraintes d'intégrité
- La validation des données se fait uniquement côté serveur
- Deux types de validation
 - validation de contrôle de surface (présence ou pas de données, données numériques contenant ou pas de lettre?)
 - validation sémantique (numéro de carte bleue valide?)
- Technologies couramment employées pour la validation
 - validation de contrôle de surface par JavaScript côté client
 - validation sémantique côté Serveur

Struts : Validator : ... et une solution

➤ Constats

- La vérification côté client est longue, explicitement non performante car longue à déboguer et nécessite l'apprentissage du langage JavaScript
- La vérification de surface n'a pas lieu d'être réalisé sur le serveur
- La solution par un Bean *ActionForm* et par la méthode *validate* est répétitive car aucune aide à la vérification

➤ Solution : utiliser le plug-in *Validator*

- Permet de décrire des contraintes d'intégrité directement dans un fichier XML
- Permet de valider les données côté client sans écrire la moindre ligne de JavaScript

Struts : Validator : Déploiement

- Le plug-in *Validator* doit être ajouté dans l'application web
 - Ajouter au répertoire de librairie de votre application *commons-validator.jar* et *jakarta-oro.jar*
 - Ajouter le fichier *validator-rules.xml* définissant les contraintes d'intégrité standard dans le répertoire */WEB-INF*
- Ajouter la balise `<plug-in>` dans le fichier *struts-config.xml* qui sert à charger la classe externe *Validator* à Struts
 - *String classname* : indique le nom de la classe à instancier
 - La balise `<set-property>` est utiliser pour initialiser la classe
 - *String property* : le nom de la propriété à initialiser
 - *String value* : la valeur de la propriété
- Pour notre utilisation `<value>` doit renseigner deux informations
 - le fichier *validator-rules.xml* (ne pas modifier)
 - le fichier définissant les contraintes d'intégrité des différents formulaires construit par le concepteur

Struts : Validator : Déploiement

- Préciser dans la balise `<form-bean>` que l'*ActionForm* est définie par *Validator* : `org.apache.struts.validator.DynaValidatorForm`
- Préciser également les paramètres présents dans le formulaire comme pour *DynaActionForm* dans la balise `<form-property>`
 - *String name* : le nom du champ
 - *String type* : son type Java

Dans le fichier `struts-config.xml` seulement deux modifications

Pour dire que les paramètres sont gérés dynamiquement et par *Validator*

```
<form-beans>
  <form-bean name="nomFormulaire" type="org.apache.struts.validator.DynaValidatorForm" >
    <form-property name="toto" type="java.lang.String" />
    <form-property name="tutu" type="java.lang.String" />
  </form-bean>
  ...
<plug-in className="org.apache.struts.validator.ValidatorPlugIn" >
  <set-property property="pathnames" value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml" />
</plug-in>
</struts-config>
```

Ce nom de fichier pour la validation explicite des contraintes n'est pas fixé

Struts : Validator : Écriture des contraintes d'intégrité

- Les contraintes d'intégrité sont définies explicitement dans un fichier xml (ici *validation.xml*)
- Le fichier décrit en deux parties des règles d'intégrité qui se trouvent dans la balise *<form-validation>*
 - *<global>* : informations à portée globale, valable dans tous les formulaires
 - *<formset>* : définit l'ensemble des formulaires pour lesquels il y a des contraintes d'intégrité

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE form-validation PUBLIC
    "-//Apache Software Foundation//DTD Commons ... Configuration 1.1.3//EN"
    "http://jakarta.apache.org/commons/dtds/validator_1_1_3.dtd">
```

<form-validation>

<global>

...

</global>

<formset>

...

</formset>

</form-validation>

Informations globales à la
description

Description de l'ensemble des
formulaires

Struts : Validator : Écriture des contraintes d'intégrité

- Dans la balise `<global>` on peut définir des constantes utilisées dans les autres descriptions des formulaires
 - `<constant-name>` : nom de la constante
 - `<constant-value>` : contient la valeur de la constante

Description
d'une constante

```
...  
<global>  
  ● <constant-name>...</constant-name>  
    <constant-value>...</constant-value> ●  
</global>  
...
```

Valeur d'une
constante

- Les contraintes d'intégrité de chaque paramètre du formulaire sont définies par des expressions régulières (peut-être déclarées dans la balise globale comme constante)

Struts : Validator : Expression Régulière

- Rappel : une expression régulière permet de trouver plusieurs mots ou phrases qui sont proches (hydro => hydrocarbure, hydrofuge, ...)
- On ne vérifie ici pas le contenu des paramètres (effectué dans la classe *Action*) mais la forme (chiffre avec des lettres, caractères interdits, ...)
- Elles sont basées sur des caractères spécifiques ayant chacun une importance
- Nous trouvons trois types de caractères
 - méta-caractères : ^, ., ?, *, + et le \$
 - groupe de caractères : \d, \D, \w, \W, \s, \S, \b, \B et \nnn
 - autres méta-caractères : {m,n}, |, [...]



Les expressions régulières sont importantes et sont souvent utilisées dans les langages de Scripts (Perl, shell bash, ...)

Struts : Validator : Expression Régulière : Méta-caractères

- Le point « . » : représente n'importe quel caractère
 - 123.5 => 123.5, 12345, 123t5, 123 5, ...
- Le point d'interrogation « ? » : le caractère précédent « ? » est optionnel
 - 12?45 => 145, 1245

Le caractère « \ » est utilisé comme échappement.
Pour chercher « . » on utilisera « \. », ...
- L'étoile « * » : le caractère précédent « * » peut être répété 0 ou plusieurs fois
 - 12*45 => 145, 122245
- Le plus « + » : le caractère précédent « + » peut être répété 1 ou plusieurs fois
 - 12+34 => 12234, 12222234
- Le dollar « \$ » et le chapeau « ^ » : caractère en fin et début de ligne
 - ^toto\$ => ligne finissant par toto et commençant par toto

Struts : Validator : Expression Régulière : Groupe de caractères

- \d : tout caractère numérique : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- \D : tout caractère sauf numérique
- \w : une lettre, une lettre accentuée, un chiffre ou « _ »
- \W : tout caractère sauf une lettre, accentuée, un chiffre ou « _ »
- \s : espace, tabulation, saut de ligne, ou tout autre caractère non imprimable
- \S : tout caractère sauf ceux définis par \s
- \b : espace, ponctuation, le début du texte du texte, fin du texte
- \B : le contraire de \b

Struts : Validator : Expression Régulière : autre méta-caractères

- Accolades {n,m} : les accolades agissent comme l'étoile, l'itération est comprise entre n et m
- Alternative | : $a \mid b \Rightarrow a \text{ or } b$
- Occurrence [abc] : 1 lettre au choix parmi trois
- Occurrence [a-z], [g-j], [a-zA-Z] : lettre comprise entre ...
- Occurrence [0-9], [6-9] : chiffre compris entre ...
- Occurrence [abc-], [a-z-] : toute lettre comprise entre ... et « - »
- Occurrence [^abc], [^a-d] : toute lettre sauf ...

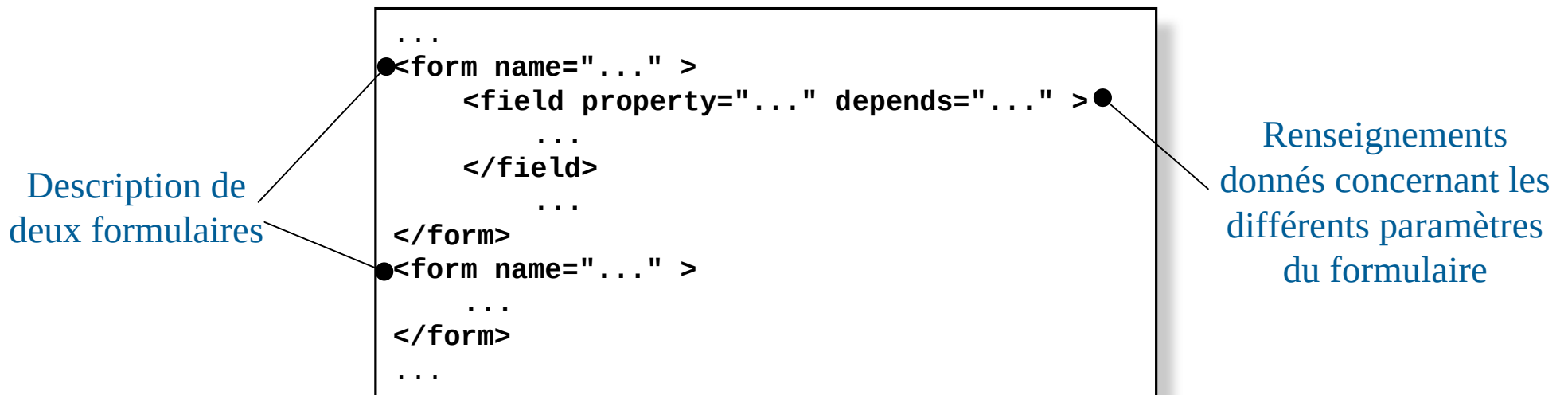
Le caractère « ^ » associé à une occurrence désigne un complément

Struts : Validator : Expression Régulière : exemples

- $a\backslash.*z \Rightarrow az, a.z, a..z, a.....z, \dots$
- $a\backslash+\backslash+z \Rightarrow a+z, a+++++z, a++++++z, \dots$
- $a\backslash.?e \Rightarrow ae, a.e$ et rien d'autre
- $a\backslash Dz \Rightarrow aaZ, a\%z, abz$ mais ne trouvera pas $a2z, a5z$
- $\backslash.\backslash d\backslash d4 \Rightarrow 1.4544, 1.8884, 1.3254, \dots$
- $\backslash D+ :$ toute chaîne de caractère non nulle sans caractères
- $a\{1,3\} \Rightarrow a, aa, aaa$ et rien d'autre
- $a.\{2,2\}z \Rightarrow abrz, avbz, a23z$
- $a|b \Rightarrow a, b$
- $(a|b)+ \Rightarrow a, b, ab, ba, abab$
- $[\backslash(\backslash\backslash)]abc \Rightarrow [abc, \backslash abc,]abc$
- $gphy[4-8] \Rightarrow gphy4, gphy5, \dots, gphy8$

Struts : Validator : Écriture des contraintes d'intégrité

- Pour chaque formulaire décrit dans la balise `<form>` il faut préciser les contraintes d'intégrité des paramètres
- Le nom du formulaire est indiqué dans l'attribut *name* de `<form>`
- Une balise `<form>` contient autant de balises `<field>` que de paramètre du formulaire
 - *String property* : nom du champ du formulaire pour lequel on définit des contraintes d'intégrité
 - *String depends* : liste des contraintes d'intégrité à vérifier



Struts : Validator : Écriture des contraintes d'intégrité

- L'attribut *depends* peut prendre les valeurs suivantes
 - *required* : champ ne doit pas être vide
 - *mask* : le champ doit correspondre à une ExpReg définie par la variable *mask*
 - *integer* : champ doit être un entier
 - *byte*, *long*, *float*, *double*, *email*, *date*, *range*, ...
- Possibilité de mettre plusieurs valeurs dans l'attribut *depends*
- Les contraintes sont vérifiées dans l'ordre de l'attribut *depends*. Si une contrainte n'est pas vérifiée, les suivantes ne le sont pas

```
...  
    <field property="prop1" depends="required, mask" >  
        ...  
    </field>  
...
```

La propriété « prop1 » est obligatoire
et doit correspondre à une ExpReg

Struts : Validator : Écriture des contraintes d'intégrité

- L'expression régulière associée à la variable *mask* est renseignée par la sous balise `<var>` de la balise `<field>`
- La balise `<var>` possède les sous balises suivantes
 - `<var-name>` : nom de la variable à modifier
 - `<var-value>` : valeur de la variable

Modification de la variable *mask*

```
...  
    <field property="prop1" depends="required, mask" >  
        ● <var-name>mask</var-name>  
          <var-value>^\s*\d+\s*$</var-value>  
    </field>  
...
```

Utilisation d'une expression régulière qui exprime que celle des chiffres précédés et suivis d'espace sont autorisé

Struts : Validator : Écriture des contraintes d'intégrité

- Chaque contrainte est liée à un message d'erreur défini par une clé et dont le contenu est initialisé dans les fichiers properties
 - *errors.required* : message lié à la contrainte *required*
 - *errors.invalid* : message lié à la contrainte *mask*
 - *errors.email* : message liée à la contrainte *email*
- Possibilité d'ajouter des arguments au moyen de la balise `<argi>`
 - *i* est un variant allant de 0 à 3
 - *String key* : message à retourner

L'argument *arg0* sera utilisé dans le retour des messages

```
<field property="prop1" depends="required, mask" >  
  <arg0 key="mypropriete" />  
</field>
```

- Les messages d'erreurs sont déjà définis (en anglais) dans le fichier *validator-rules.xml*

Fichier *validator-rules.xml* par défaut

```
errors.required={0} is required.  
errors.invalid={0} is invalid.  
errors.email={0} is an invalid e-mail adress
```

Argument utilisé dans le message

Struts : Validator : Écriture des contraintes d'intégrité

➤ Épisode 7 : utilisation d'un Bean *Validator* dynamique

```
...
<form-beans>
  <form-bean name="formPeronne" type="org.apache.struts.validator.DynaValidatorForm" >
    <form-property name="nom" type="java.lang.String" />
    <form-property name="age" type="java.lang.String" />
  </form-bean>
...

<action
  path="/main" name="formPersonne" scope="session" validate="true" input="/erreurs.do"
  parameter="/vues/main.html" type="monpackage.FormulaireAction">
  ...
</action>

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

Utilisation de la classe *DynaValidatorForm* de Struts

Validation des données du formulaire côté serveur activée

```
...
<html:form action="/main" >
<table>
  <tr><td>Nom</td><td><html:text property="nom" size="20" /></td></tr>
  <tr><td>Age</td><td><html:text property="age" size="3"/></td></tr>
</table>
<table>
  <tr>...</tr>
</table>
</html:form>
...
```

Le formulaire JSP ne change pas

Struts : Validator : Écriture des contraintes d'intégrité

➤ Épisode 7 (suite) : utilisation d'un Bean *Validator* dynamique

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE form-validation PUBLIC
    "-//Apache Software Foundation//DTD Commons ... Configuration 1.1.3//EN"
    "http://jakarta.apache.org/commons/dtds/validator_1_1_3.dtd">

<form-validation>
<global>
    <constant>
        <constant-name>entierpositif</constant-name>
        <constant-value>^\s*\d+\s*$</constant-value>
    </constant>
</global>
<formset>
    <form name="formPersonne3">
        <field property="nom" depends="required">
            <arg0 key="personne.nom"/>
        </field>
        <field property="age" depends="required,mask">
            <arg0 key="personne.age"/>
            <var>
                <var-name>mask</var-name>
                <var-value>${entierpositif}</var-value>
            </var>
        </field>
    </form>
</formset>
</form-validation>
```

Définition d'une contrainte d'intégrité dans une constante

Le paramètre « nom » est obligatoire

Le paramètre « age » est obligatoire et doit respecter l'expression régulière définie par *entierpositif*

Fichier de validation.xml

Struts : Validator : Écriture des contraintes d'intégrité

➤ Épisode 7 (suite bis) : utilisation d'un Bean *Validator* dynamique

```
# Pour les messages d'erreurs d'avant Validator
formulaire.nom.vide=<li>Vous devez indiquer un nom</li>
formulaire.age.vide=<li>Vous devez indiquer un age</li>
formulaire.age.incorrect=<li>L'age [{0}] est incorrect</li>
errors.header=<ul>
errors.footer=</ul>
```

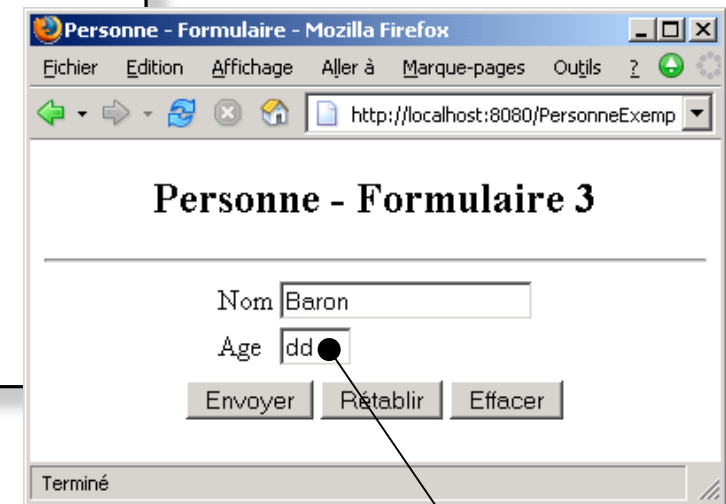
Fichier de properties_fr

```
# Pour la validation des données avec Validator
personne.nom=nom
personne.age=age
```

{0} est remplacé par age.

Défini dans arg0 de validator.xml

```
errors.invalid={0} est invalide.
errors.required={0} est obligatoire.
```



L'âge est invalide car il contient du texte



Le message « errors.invalid » est utilisé dans le fichier properties.

Struts : Validator : Écriture des contraintes d'intégrité

- Nous avons vu pour l'instant que la vérification des données se faisait essentiellement côté serveur
- Struts et le plug-in *Validator* offre la possibilité de renforcer la vérification en amont côté client en générant automatiquement le JavaScript
- Le JavaScript généré correspond à la description donnée dans le fichier de contraintes d'intégrité
- Deux choses doivent être renseignées dans la JSP du formulaire
 - Dans la balise `<form>` il faut initialiser l'attribut `onsubmit = "return validateFormPersonne(this)"` si le nom du formulaire est « FormPersonne »
 - Ajouter la balise `<javascript>` avec l'attribut `formName="formPersonne"`

Deux vérifications de surface des données sont réalisées. Une côté client et une côté serveur



Struts : Validator : Écriture des contraintes d'intégrité

➤ Épisode 8 : génération automatique de code JavaScript

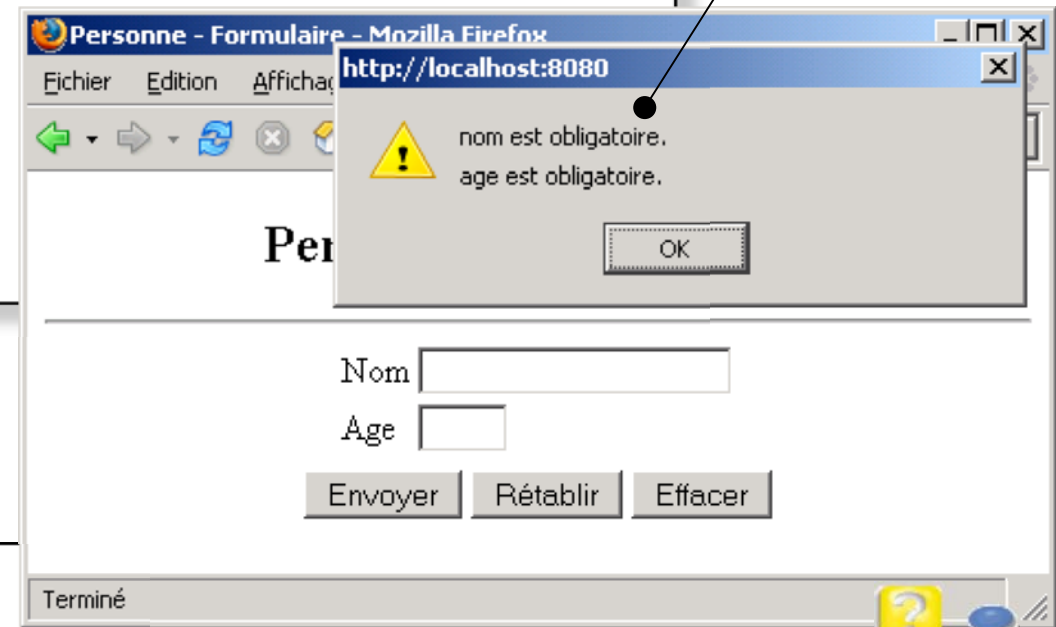
```
...  
<html:form action="/main" onsubmit="return validateFormPersonne(this);" >  
<table>  
  <tr><td>Nom</td><td><html:text property="nom" size="20" /></td></tr>  
  <tr><td>Age</td><td><html:text property="age" size="3" /></td></tr>  
</table>  
<table>  
  <tr>...</tr>  
</table>  
</html:form>  
<html:javascript formName="formPersonne" />  
...
```

Fichier JSP du formulaire

Code source de la page
HTML généré par Tomcat

```
...  
<!-- Begin  
function validateFormPersonne4(form) {  
  if (bCancel)  
    return true;  
  else  
    var formValidationResult;  
    formValidationResult = validateRequired(form) &&  
    validateMask(form);  
    return (formValidationResult == 1);  
  }  
  ...
```

Boîte de dialogue
gérée par
JavaScript



Le code JavaScript a été
généré automatiquement par
le framework Struts

Struts : changement de Locale

- Struts gère en interne l'internationalisation puisqu'il accède à des fichiers *properties* pour les messages à afficher
- Par défaut la « locale » est configurée suivant la localisation du client (un client dans une région française utilisera les *properties fr*)
- On peut cependant changer la « locale » d'une partie partielle ou complète d'une application web
- Le changement de « locale » ne peut être effectué que dans une classe *Action* du framework Struts
- La classe *Action* possède la méthode *setLocale*
setLocale(HttpServletRequest,Locale)
 - *HttpServletRequest* : permet d'accéder à la session de l'utilisateur
 - *Locale* : objet relatif à la nouvelle « locale »

Struts : changement de Locale

➤ Exemple : choisir la « locale » d'une application web

Le fichier JSP
associé

La page d'accueil
en Français

```
<%@ taglib uri="htmlstruts" prefix="html" %>
<%@ taglib uri="beanstruts" prefix="bean" %>

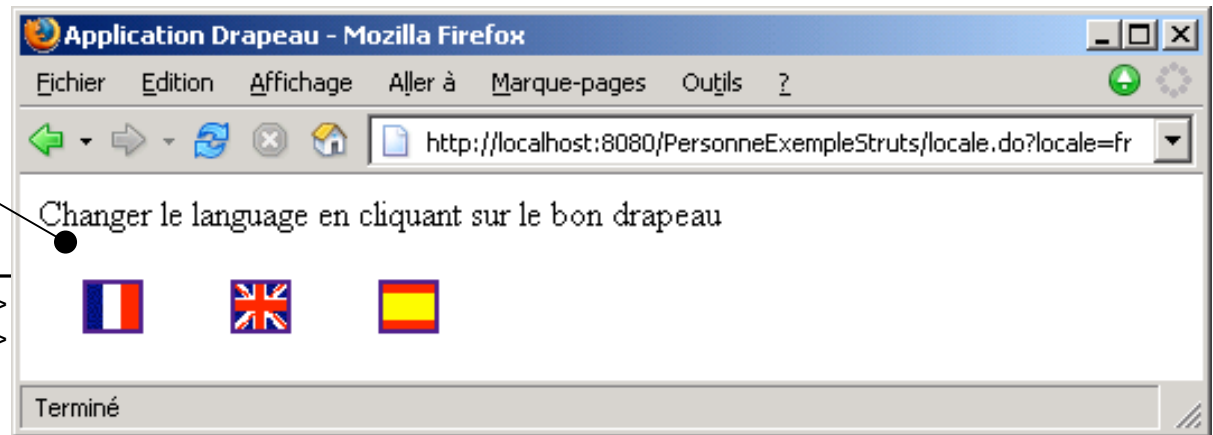
<html>
<head>
<title><bean:message key="welcome.title"/></title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body>
<p><bean:message key="welcome.message"/></p>

<table width="200" border="0">
  <tr>
    <th scope="col"><html:link page="/locale.do?locale=fr" ><html:img
      page="/vues/french.gif" width="23" height="20" /></html:link></th>
    <th scope="col"><html:link page="/locale.do?locale=en" ><html:img
      page="/vues/english.gif" width="23" height="20" /></html:link></th>
    <th scope="col"><html:link page="/locale.do?locale=sp" ><html:img
      page="/vues/spanish.gif" width="23" height="20" /></html:link></th>
  </tr>
</table>
</body>
</html>
```

Solution pour afficher une
clé d'une ressource

Les liens hypertextes
font appel à une action
qui modifie la « locale »



Struts : changement de Locale

➤ Exemple (suite) : choisir la « locale » d'une application web

```
...  
<action  
  path="/localechoice" parameter="/vues/localechoice.jsp"  
  type="org.apache.struts.actions.ForwardAction" />  
<action  
  path="/locale" type="monpackage.ChoiceLocale">  
  <forward name="success" path="/localechoice.do" />  
</action>  
</action-mappings>  
  
</struts-config>
```

L'action associée à la page JSP est d'effectuer une simple redirection

Quand l'utilisateur clique sur le lien il appelle une action spécifique

```
public class ChoiceLocale extends Action {  
  public ActionForward execute(ActionMapping mapping, ActionForm form,  
    HttpServletRequest req, HttpServletResponse res) throws Exception {  
    String mon_objet = req.getParameter("locale");  
    Locale ma_locale;  
    if (mon_objet != null) {  
      ma_locale = new Locale(mon_objet);  
    } else {  
      ma_locale = new Locale("fr", "FR");  
    }  
    this.setLocale( req, ma_locale);  
    return mapping.findForward("success");  
  }  
}
```

Modification de la « locale » à l'aide du paramètre de la requête

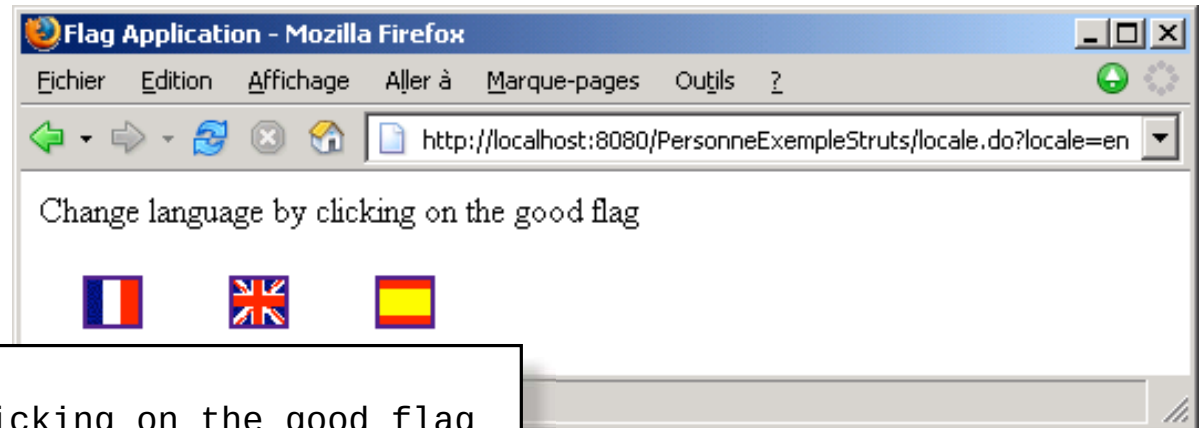
La « locale » est modifiée au niveau session

Struts : changement de Locale

➤ Exemple (suite bis) : choisir la « locale » d'une application web

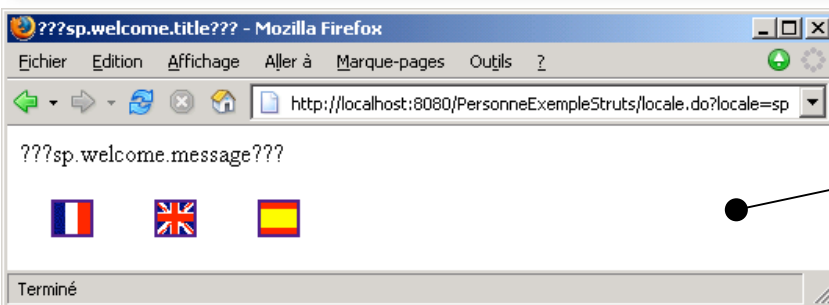
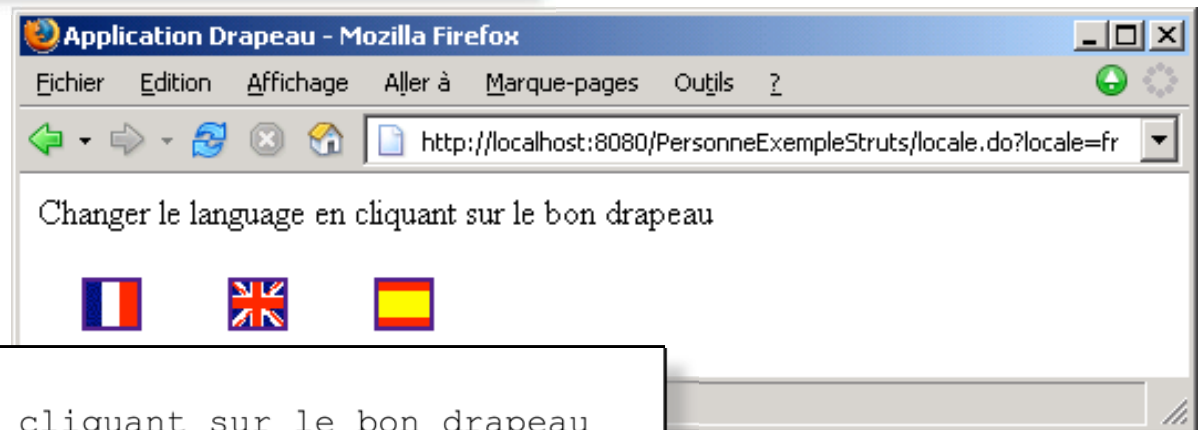
Fichier properties
« english »

```
welcome.title=Flag Application  
welcome.message=Change language by clicking on the good flag
```



Fichier properties
« french »

```
welcome.title=Application Drapeau  
welcome.message=Changer le langage en cliquant sur le bon drapeau
```



Pas de fichier properties pour le
langage « spanish » problème pour
trouver la clé associée

Struts : fichier struts-config.xml multiples

- Pour l'instant nous nous sommes limités à utiliser un seul fichier struts-config.xml pour décrire l'application web
- La séparation en plusieurs fichiers permet
 - une meilleur visibilité des fichiers de configuration
 - une décomposition moins monolithique de l'application web globale
- Déclarer dans web.xml de l'intégralité des fichiers struts-config.xml

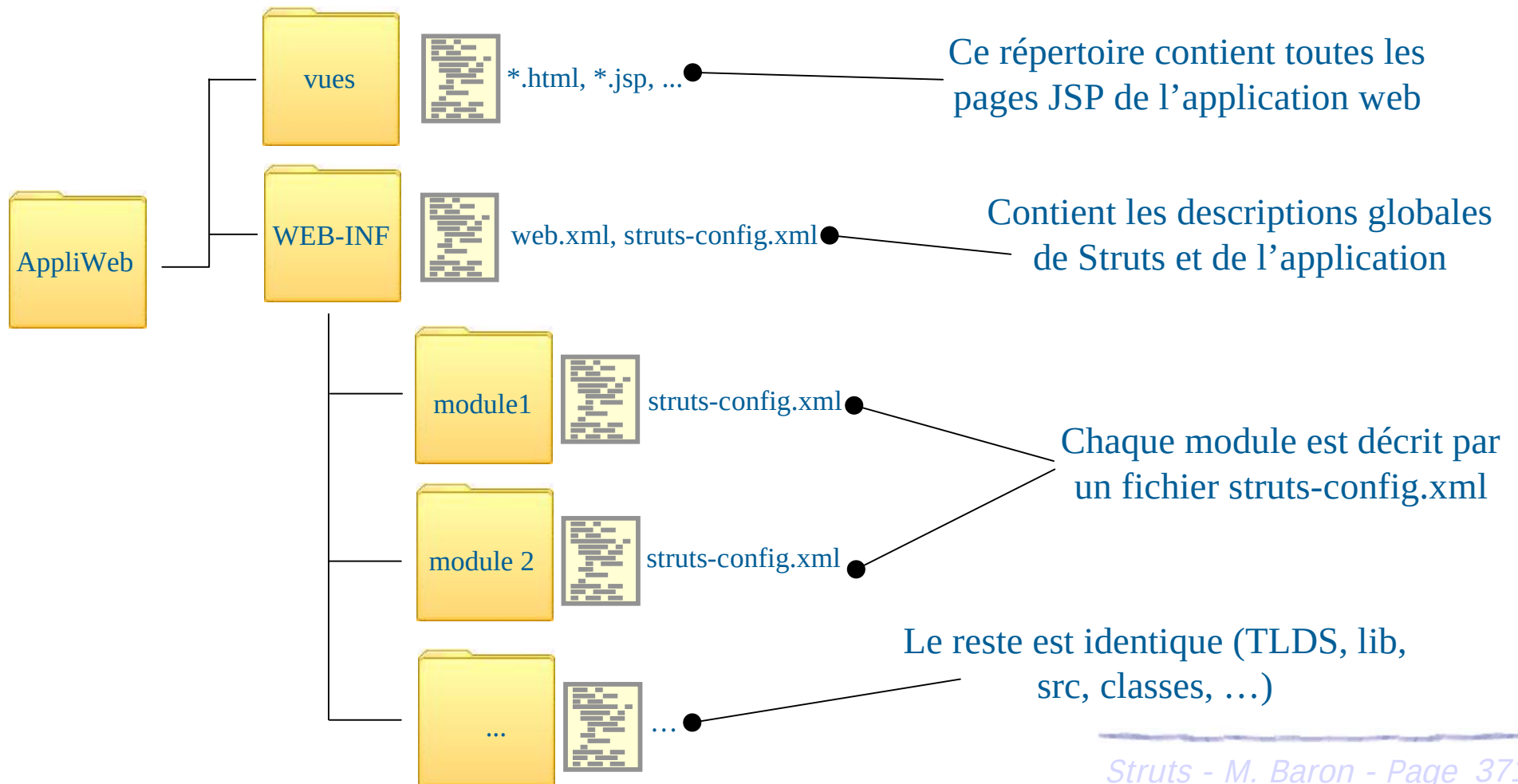
Un fichier principal
relatif à l'application
web globale

Pour chaque
struts-config.xml
supplémentaire ajouter
une balise <init-param>

```
...  
    <servlet>  
        <servlet-name>action</servlet-name>  
        <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>  
        <init-param>  
            ● <param-name>config</param-name>  
              <param-value>/WEB-INF/struts-config.xml</param-value>  
        </init-param>  
        <init-param>  
            ● <param-name>config/module1</param-name>  
              <param-value>/WEB-INF/module1/struts-config.xml</param-value>  
        </init-param>  
        ...  
    <servlet-mapping>  
        <servlet-name>action</servlet-name>  
        <url-pattern>*.do</url-pattern>  
    </servlet-mapping>  
...
```

Struts : fichier struts-config.xml multiples

- La décomposition physique des fichiers de votre application web peut suivre la logique suivante
- Pour accéder à un élément utilisation du nom du module associé dans l'URL *localhost:8080/appliweb/module1/toto.do*

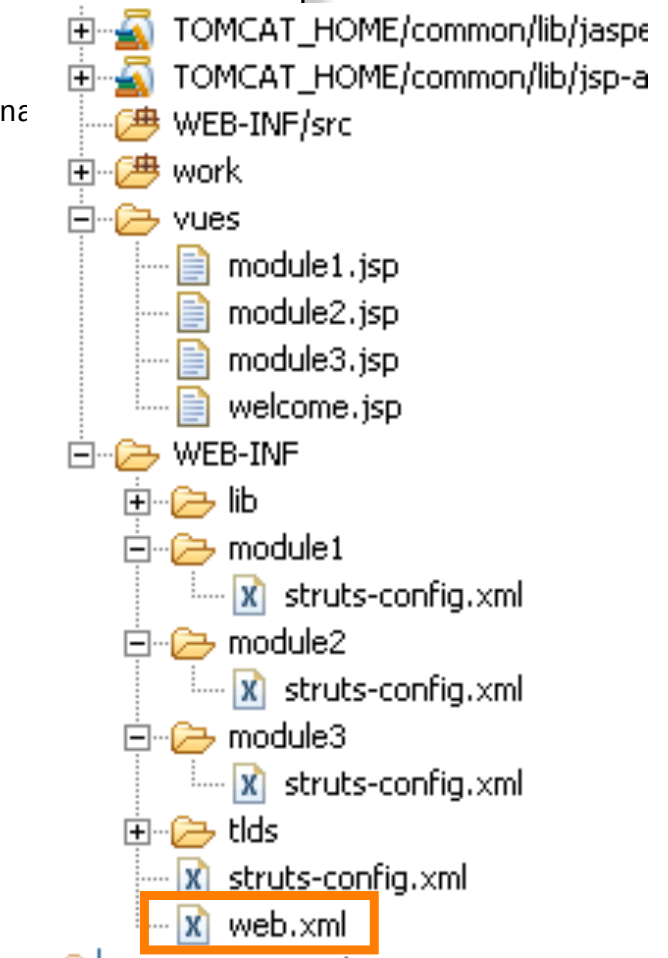


Struts : fichier struts-config.xml multiples

➤ Exemple : une petite application qui fait coucou plusieurs fois

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <display-name>Application gérant plusieurs fichiers struts-config</display-name>

  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>config/module1</param-name>
      <param-value>/WEB-INF/module1/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>config/module2</param-name>
      <param-value>/WEB-INF/module2/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>config/module3</param-name>
      <param-value>/WEB-INF/module3/struts-config.xml</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```



Le fichier web.xml

Struts : fichier struts-config.xml multiples

► Exemple (suite) : une petite application qui fait coucou plusieurs fois

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
    "http://struts.apache.org/dtds/struts-config_1_2.dtd">

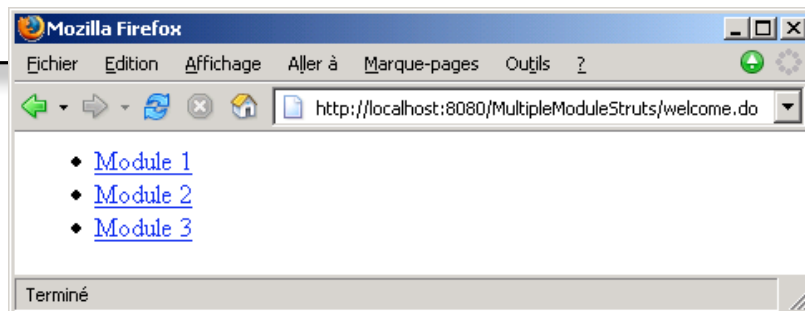
<struts-config>
    <action-mappings>
        <action path="/welcome" parameter="/vues/welcome.jsp"
            type="org.apache.struts.actions.ForwardAction" />
    </action-mappings>
</struts-config>
```

Le fichier struts-config.xml

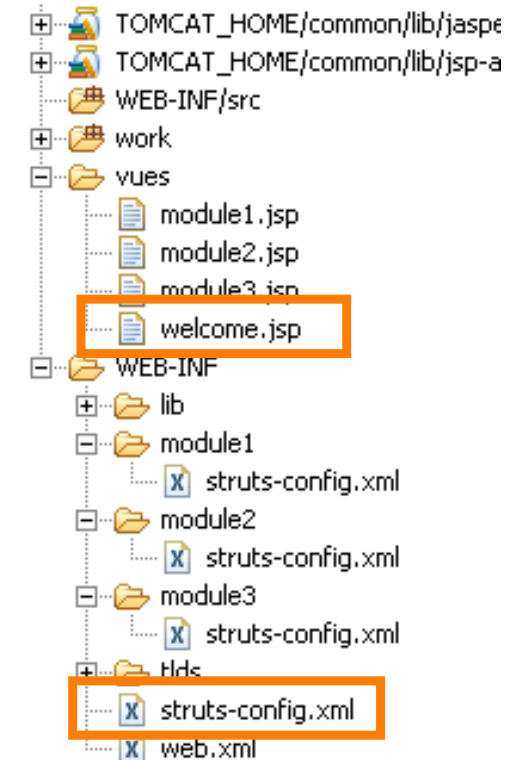
Permet d'identifier
le module visé

```
<%@ taglib uri="htmlstruts" prefix="html" %>

<ul>
<li><html:link module="/module1" action="/module1">Module 1</html:link></li>
    <li><html:link module="/module2" action="/module2">Module 2</html:link></li>
    <li><html:link module="/module3" action="/module3">Module 3</html:link></li>
</ul>
```



Le fichier
welcome.jsp



Struts : fichier struts-config.xml multiples

► Exemple (suite) : une petite application qui fait coucou plusieurs fois

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
    "http://struts.apache.org/dtds/struts-config_1_2.dtd">

<struts-config>
    <action-mappings>
        <action path="/module1" parameter="/vues/module1.jsp"
            type="org.apache.struts.actions.ForwardAction" />
    </action-mappings>
</struts-config>
```

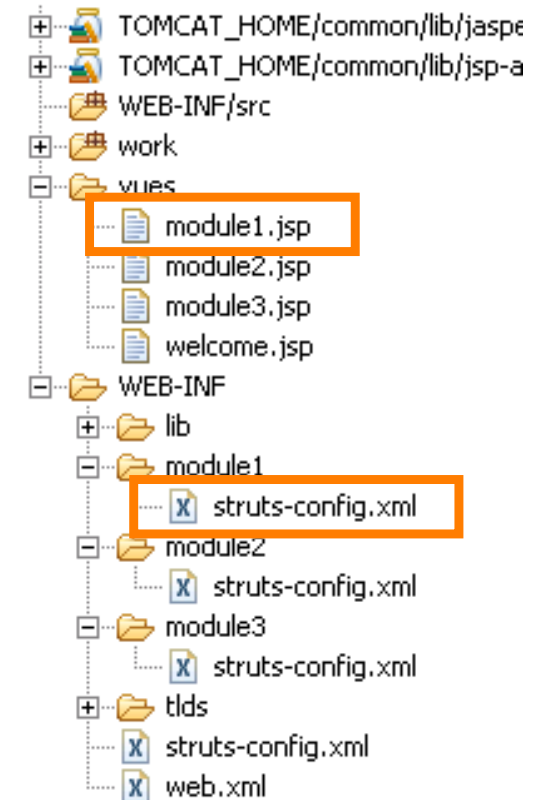
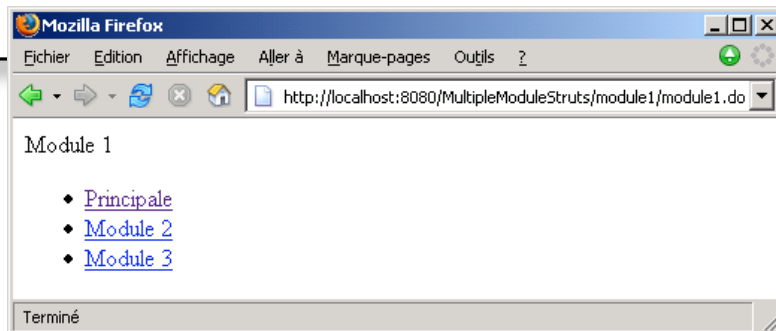
Le fichier struts-config.xml du module 1

```
<%@ taglib uri="htmlstruts" prefix="html" %>
```

Module 1

```
<ul>
<li><html:link module="" action="/welcome.do">Principale</html:link></li>
<li><html:link module="/module2" action="/module2">Module 2</html:link></li>
<li><html:link module="/module3" action="/module3">Module 3</html:link></li>
</ul>
```

Pas de module



Le fichier module1.jsp

Struts et JSTL : coopération pour l'internationalisation

- Nous avons étudié dans la partie des balises personnalisées, la bibliothèque JSTL et son module I18n
- La bibliothèque JSTL est simple d'utilisation et permet d'ajouter des fichiers properties sans déclaration dans Struts
- La coopération de JSTL et Struts pour l'affichage d'information dans les pages JSP nécessite la correspondance entre les variables
- Exemple pour l'internationalisation et la variable « locale »
 - Struts : `org.apache.struts.action.LOCALE`
 - JSTL : `javax.servlet.jsp.jstl.fmt.locale.session`

Struts et JSTL ont la même
Locale « fr, FR »

```
session.setAttribute("org.apache.struts.action.LOCALE", new Locale("fr", "FR"));  
session.setAttribute("javax.servlet.jsp.jstl.fmt.locale.session", new Locale("fr", "FR"));
```

- Solution : modifier dans une classe *Action* les deux variables

Struts et JSTL : coopération pour l'internationalisation

► Exemple : coopérer JSTL et Struts

Remplace les balises « Bean »
par celles de la JSTL

```
<%@ taglib uri="htmlstruts" prefix="html" %>
<%@ taglib uri="fmtjstl" prefix="fmt" %>

<html>
<head>
<fmt:setBundle basename="erreur" />
<title><fmt:message key="welcome.title" /></title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body>
<p><fmt:message key="welcome.message" /></p><br>

<table width="200" border="0">
  <tr>
    <th scope="col"><html:link page="/locale.do?locale=fr" ><html:img
      page="/vues/french.gif" width="23" height="20" /></html:link></th>
    <th scope="col"><html:link page="/locale.do?locale=en" ><html:img
      page="/vues/english.gif" width="23" height="20" /></html:link></th>
    <th scope="col"><html:link page="/locale.do?locale=sp" ><html:img
      page="/vues/spanish.gif" width="23" height="20" /></html:link></th>
  </tr>
</table>
</body>
</html>
```

La balise « setBundle » permet
d'indiquer la ressource
properties



Reprise de l'exemple avec
les drapeaux.

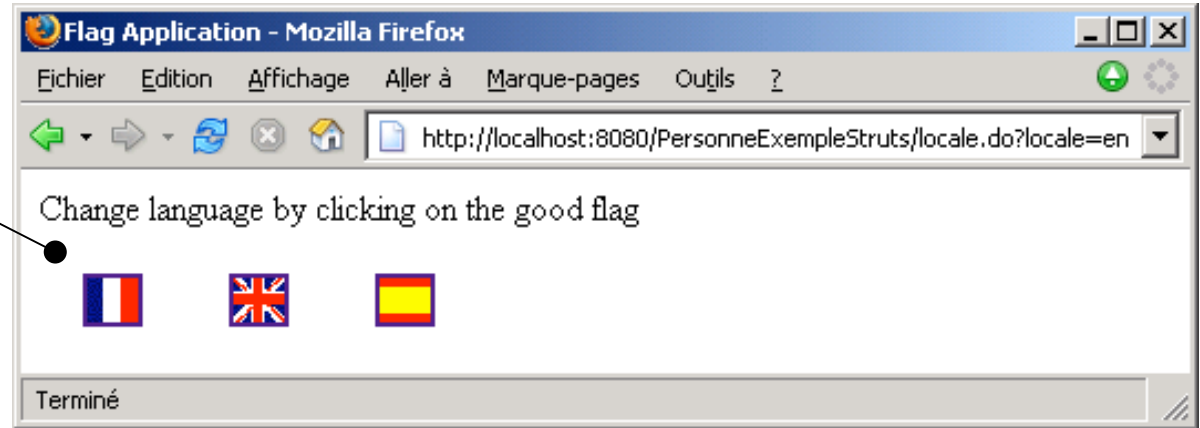
Affichage des clés en
utilisant la balise « fmt »

Struts et JSTL : coopération pour l'internationalisation

► Exemple (suite) : coopérer JSTL et Struts

Résultat si on choisit la langue anglaise

Classe de type *Action*



```
public class ChoiceLocale extends Action {
    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest req, HttpServletResponse res) throws Exception {
        String mon_objet = req.getParameter("locale");
        Locale ma_locale;
        if (mon_objet != null) {
            ma_locale = new Locale(mon_objet);
        } else {
            ma_locale = new Locale("fr", "FR");
        }
        this.setLocale( req, ma_locale);
        req.getSession().setAttribute("javax.servlet.jsp.jstl.fmt.locale.session",
            ma_locale);
        return mapping.findForward("success");
    }
}
```

La classe *Action* est complétée de manière à modifier la variable liée à l'internationalisation de JSTL