Atod

Projet Framework

Plan

Atod

Projet Framework

Présentation du Framework

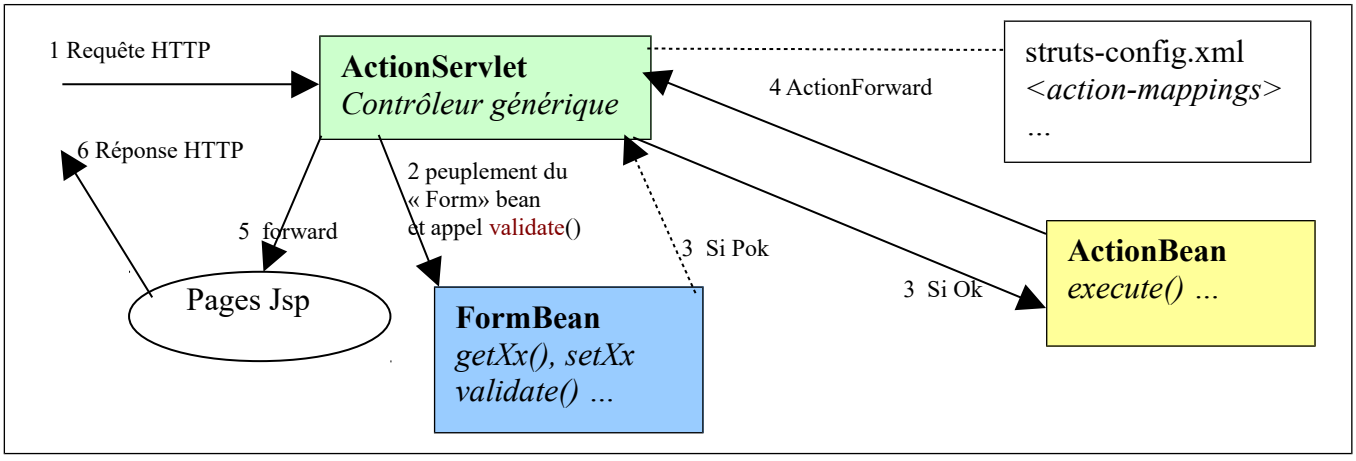
1 – Fonctionnement :

Développer un framework qui aide le programmeur dans la mise en place d'un projet web.
(Ce framework s'inspire de struts).

En vigueur depuis les années 70 (SMALLTALK), l'architecture MVC (Modèle-Vue-Contrôleur), est très utilisée pour la partie "présentation" **:**
Model = modèle objet (structure de données & traitements)
View = Vue de présentation (IHM - affichage)
Control = contrôles des enchaînements des opérations (navigation, ...)

La version "MVC 2" correspond à une variante améliorée de "*MVC pour le Web*" où il n'y a qu'un seul contrôleur (servlet) générique supervisant toute les navigations entre les pages
(Avantage : un seul point central à sécuriser et à paramétrer dans web.xml).

Collaboration MVC2 au sein d'une application Web développée en Java:
→ La Servlet joue le rôle de contrôleur : elle reçoit une requête, lui applique un éventuel contrôle de saisie (validation quelquefois déléguée ou automatisée).
→ Le contrôleur demande à un JavaBean d'effectuer les traitements. Le JavaBean effectue (ou bien *délègue*) les traitements (accès Jdbc, *EJB*,…) et récupère les résultats qu'il mémorise dans ses attributs (*ou dans d'éventuel(s) JavaBean(s) annexes de données).*
Ce(s) JavaBean(s) joue(nt) le rôle de Modèle.
→ Le contrôleur après avoir reçu un statut (ok/ko) ou une exception effectue une redirection (forward) vers une page JSP pour l'affichage du résultat ou vers une page JSP d'erreur.
→ La page JSP (jouant le rôle de Vue) *récupère les données nécessaires à l'affichage auprès du JavaBean* et génère (met en forme) la page HTML à renvoyer.



A l'arrivée d'une requète cliente, il y a

- instantiation d'une classe form correspondant à cette requete qui va recuperer les données du formulaire et les valider
- instantiation d'une classe action correspondant à cette requete qui va faire le traitement et renvoyer une url de suite

Projet Framework

Présentation du Framework

2 – Configuration :

Un fichier xml permettra de paramétrer notre framework et sera de cette forme :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<monframework-config>
	<actions>
		<action>
			<action-name>controller.ActionTestWs1</action-name>
			<url-pattern>/testWs1Action.perform</url-pattern>
			<form-name>testWs1Form</form-name>
		</action>
		<action>
			<action-name>controller.ActionTestWs2</action-name>
			<url-pattern>/testWs2Action.perform</url-pattern>
			<form-name>testWs2Form</form-name>
		</action>
	</actions>
	<forms>
		<form>
			<form-class>form.TestWs1Form</form-class>
			<form-name>testWs1Form</form-name>
		</form>
		<form>
			<form-class>form.TestWs2Form</form-class>
			<form-name>testWs2Form</form-name>
		</form>
	</forms>
</monframework-config>
```

<url-pattern> correspond à l'url entrée dans la jsp,
<action-name> correspond au nom de l'action à instancier lors de la réception de l'url pattern correspondante
<form-name> correspond au nom interne au fichier xml
<form-class> correspond au nom de la classe form

Atod

Projet Framework

Les étapes du projet

1 – Etape 1 : mise en place du modèle MVC2:

Le framework permettra d'implémenter un modèle MVC2 :

- une seule servlet en guise de contrôleur - ActionServlet
- des classes ActionX qui contiennent le corps du code à exécuter.
- Les classes ActionX implémentent l'interface Action.
- L'interface Action contient une méthode 'execute' qui retourne une String et prend en argument HttpServletRequest request et HttpServletResponse response
- Une factory permettant d'identifier la classe ActionX à instancier en fonction du path d'entrée dans la servlet.
- Cette classe implémente le design pattern factory.

2 – Etape 2 : fichier de configuration:

Exportation des données vers un fichier xml.
Parsing de ce fichier pour alimenter la Map de la fabrique du noyau MVC2

- Dans la map; la clef est une String, la value est une classe

Implémenter les balises `<monframework-config>` `<actions>` `<action>` `<action-name>` et `<url-pattern>`

- La lecture du fichier xml se fera via l'utilisation du DOM (aide en annexe)
- Le fichier xml ne doit pas être renseigné par un chemin en absolu, pour que le framework garde de sa souplesse. (aide en annexe)

3 – Etape 3 : mise en place des formbeans :

Cette étape implémente la récupération et la validation des données saisies par un formulaire HTML.
A chaque action correspond un form.
Le nom de la classe form correspondante au formulaire sera renseignée dans le fichier xml par la balise : <form-name>
Les classes form implementent l'interface ActionForm qui comprend la méthode boolean validateForm().
Cette méthode vérifie la validité des données saisies dans le formulaire.

Introspection (aide en annexe)
Les classes formX sont peuplées par les données saisies dans le formulaire.
Une classe MyBeanPopulate, comprenant une méthode :

populateBean(Object theObejct, Map params)

qui permet de peupler un objet quelconque à partir d'une Map

4 – Etape 4 : peuplement des formbeans pour les actions:

Repeuplement de formulaire après vérification, le formulaire est renvoyé au client qui doit ressaisir les données erronées.

5 – Etape 5 : les exceptions :

Faire une classe Exeption pour gérer les exceptions du projet
Faire une java doc du projet

Atod

Projet Framework

Annexe 1 : Java XML

Utilisation du DOM pour la lecture d'un document XML :

Document XML :

```
<personnes>
	<personne>
	<nom>toto</nom>
	<prenom>titi</prenom>
	</personne>
	<personne>
	<nom>toto1</nom>
	<prenom>titi1</prenom>
	</personne>
</personnes>
```

Lecture de ce fichier via le DOM java :

```
import java.io.IOException; //pour les erreurs de lecture du fichier

import javax.xml.parsers.DocumentBuilder; //pour l'ouverture et la construction du DOM
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public abstract class test {
	public static void main(String[] args) throws ParserConfigurationException,
			SAXException, IOException {
		// TODO Auto-generated method stub
		String fichier = "c:/personnes.xml";
		/*1*/
		DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
		DocumentBuilder builder = factory.newDocumentBuilder();
		/*2*/
		Document document = builder.parse(fichier);
		/*3*/
		Element racine = document.getDocumentElement();
		NodeList bases = racine.getElementsByTagName("personne");

		/*4*/// parcours des elements
		for (int i = 0; i < bases.getLength(); i++) {
			Node base = bases.item(i);
			// elements enfants
			NodeList elements = base.getChildNodes();
			for (int j = 0; j < elements.getLength(); j++) {
				Node enfant = elements.item(j);
				if(enfant.getNodeName().equals("nom"))
					System.out.println(enfant.getTextContent());
				if(enfant.getNodeName().equals("prenom"))
					System.out.println(enfant.getTextContent());
			}
		}
	}
}
```

Atod

Projet Framework

Annexe 1 : Java XML

```
/*1*/
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
```

Ces deux lignes permettent la création en mémoire d'un document XML.

```
/*2*/
Document document = builder.parse(fichier);
```

La méthode « parse » du builder permet de mettre en memoire un Objet de type « Document », contenant le fichier XML.

```
/*3*/
Element racine = document.getDocumentElement();
NodeList bases = racine.getElementsByTagName("personne");
```

La méthode « `getDocumentElement()` », permet de se positionner dans le document et de récupérer sa racine.
La méthode « `getElementsByTagName("personne")` », permet de récupérer à partir de la racine du document les enfants dont le nœud s'appelle « personne ».

```
/*4*/// parcours des elements
for (int i = 0; i < bases.getLength(); i++) {
	Node base = bases.item(i);
```

Cette boucle permet de parcourir tous les éléments retourné par La méthode « `getElementsByTagName("personne")` », et de pouvoir manipuler ensuite chacun des Nœud de type « personne ».

```
NodeList elements = base.getChildNodes();
for (int j = 0; j < elements.getLength(); j++) {
	Node enfant = elements.item(j);
	if(enfant.getNodeName().equals("nom"))
		System.out.println(enfant.getTextContent());
	if(enfant.getNodeName().equals("prenom"))
		System.out.println(enfant.getTextContent());
}
```

La méthode « `getChildNodes()` » permet de retourner la les enfants de l'element « personne » que l'on est en train de parcourir.
La boucle, nous permet de parcourir chacun de ses enfants, puis d'afficher dans la console leur valeur, à l'aide de la méthode « `getTextContent()` ».

Atod

Projet Framework

Annexe 2 : Introspection

Découvrir dynamiquement les informations propres à une classe ou à un objet.

La classe « Class »

Les méhodes :

- objet.getClass() : la classe de l'objet ou objet.class
- Class.forName() : charge une classe dynamique avec le nom complet, paquetage/classe
- objet.newInstance() : créé une nouvelle instance (utile pour les classes chargées dynamiquement et pour lesquelles on ne peut pas utiliser new
- objet.getName() : retourne le nom de la classe

L'API java.lang.reflect

Permet d'accéder aux classes, à leurs champs, méthodes :

- – les attributs : la classe Field
- – les méthodes : la classe Method
- – les constructeurs : la classe Constructor
- ces 3 classes étendent la classe AccessibleObject

```
import java.lang.reflect;
Object o;                                                    // un objet qcq
Class c = o.getClass();                                      // obtenir le type de classe
while ( c.isArray()) c = c.getComponentType(); // as-t-on à faire à un tableau?
if ( !c.isPrimitive()) {                                     // un type primitif
for ( Class s = c ; s != null; s = s.getSuperClass() ) // recherche des ancêtres
System.out.println(s.getName() + " extends " );
Object newobj = null;
try { newobj = c.newInstance(); }   // créer une nouvelle instance de c
catch ( Exception e ) {}
try {                                  // vérifier si la classe à une méthode « setText »
Method m = c.getMethod( " setText" , new Class[] { String.class } );
          //  ( " setText" : nom            new Class[] { String.class } :type des arguments

m.invoke(newobj, new Object[] { "MyLabel" });
}
catch ( Exception e ) { }
```

Projet Framework

Annexe 2 : Introspection

Appels (ou invocations) de méthodes dynamiquement

```
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

public class DynCall {

   public static void main(String[] args) {
      try {
         Class c;
         DynCall o = new DynCall();
         c = o.getClass();
         // Récupère un objet Method représentant la méthode "display" sans argument
         Method display = c.getMethod("display", null);

         // Ces 2 appels sont équivalents, et font l'appel de la méthode sur  l'objet o
         display.invoke(o, null);
         display.invoke(o, new Object[] {});

         // Récupère un objet Method représentant la méthode "setName" ayant
         // 1 argument String
         Method setName = c.getMethod("setName",
               new Class[] { String.class });

         // Appel de la méthode avec un argument String
         setName.invoke(o, new Object[] { "John" });

         display.invoke(o, null);

      } catch (Exception e) { }
   }
```

Atod

Projet Framework

Annexe 3 : Utiliser des ressources

Les ressources peuvent être des fichiers, image, txt, classes. Comment retrouver leur référence ?
ClassLoader référence toutes les classes chargées en mémoire.
Thread.currentThread().getContextClassLoader() permet de récupérer le ClassLoader courant.

Vous récupérez l'Url de l'objet référencé par la méthode getResource(String ressource). Une connexion à une url se fait par la méthode openConexion(). Un canal de lecture sur une connexion d'obtient par getInputStream.
La méthode getResourceAsStream qui fourni un InputStream directement