6

Intégration de Struts

Struts est certainement le framework de présentation le plus réputé dans le monde J2EE. Cette notoriété le fait souvent demander dans les projets afin de respecter l'état de l'art.

En termes techniques, Struts est un framework MVC relativement élémentaire. Il gère des formulaires HTML et les actions des utilisateurs, mais ne va pas beaucoup plus loin. Cette simplicité fait aussi sa force : facile à prendre en main, il permet d'effectuer efficacement les tâches pour lesquelles il est conçu.

L'intérêt principal de Struts réside aussi dans le très grand nombre de développeurs qui connaissent son fonctionnement. Utiliser Struts sur un projet, c'est donc s'assurer de ne pas avoir de problèmes de ressources humaines.

Ce framework est cependant vieillissant et se trouve aujourd'hui sous le feu des critiques. Nous verrons dans le cours de ce chapitre quels sont ses défauts et dans quelles situations Struts n'est plus la meilleure solution disponible.

Une fois l'architecture de Struts et ses principales classes explicitées, nous verrons de quelle manière Spring intègre Struts dans une application Java/J2EE, et à quelles fins. Cette intégration pouvant être réalisée de trois manières, nous étudierons chacune d'elles et dégagerons leurs forces et faiblesses respectives.

En fin de chapitre, nous détaillerons au travers de l'étude de cas Tudu Lists la configuration de Struts et de Spring, ainsi que le développement d'actions et de formulaires spécifiques.

Fonctionnement de Struts

Cette section se penche sur l'architecture interne de Struts et présente sa configuration et ses classes principales.

Struts représentant une implémentation classique du framework MVC, il est important de commencer par rappeler brièvement les caractéristiques de ce modèle.

Le pattern MVC (Model View Controller)

Le pattern MVC est communément utilisé dans les applications Java/J2EE pour réaliser la couche de présentation des données aussi bien dans les applications Web que pour les clients lourds. Lorsqu'il est utilisé dans le cadre de J2EE, il s'appuie généralement sur l'API servlet et des technologies telles que JSP/JSTL.

Il existe deux types de patterns MVC, le pattern MVC dit de type 1, qui possède un contrôleur par action, et le pattern MVC dit de type 2, plus récent, qui possède un contrôleur unique. Nous nous concentrerons sur ce dernier, puisque c'est celui sur lequel s'appuie Struts.

La figure 6.1 illustre les différentes entités du type 2 du pattern MVC ainsi que leurs interactions lors du traitement d'une requête.

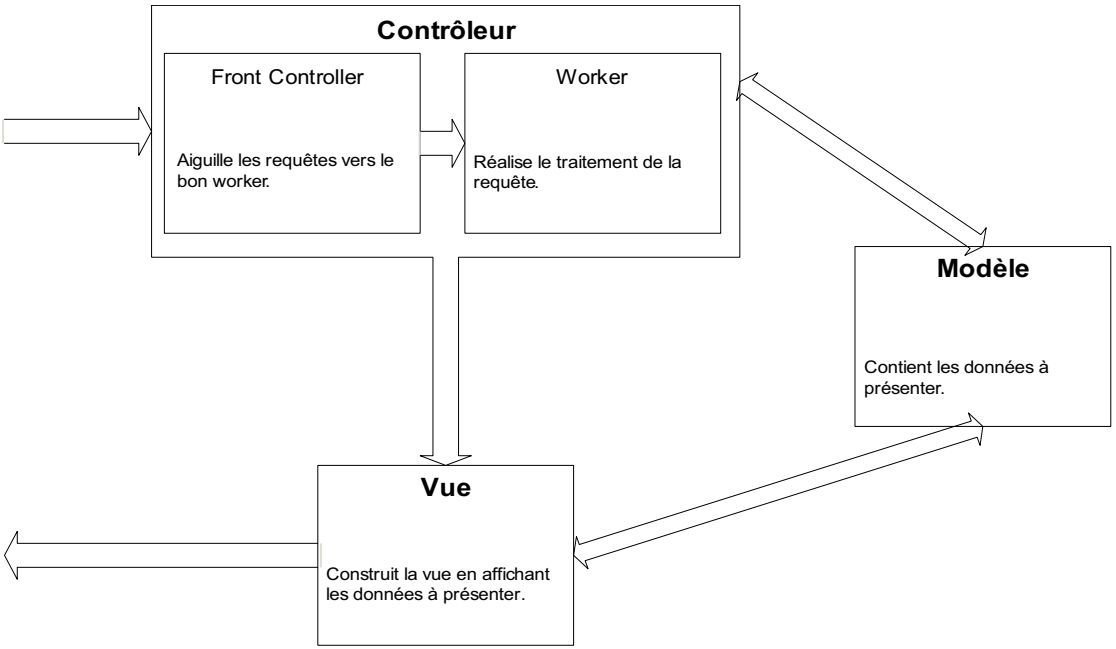

Figure 6.1
Composants du pattern MVC type 2

Les caractéristiques des composants de ce pattern sont les suivantes :

- **Modèle.** Permet de mettre à disposition les informations utilisées par la suite lors des traitements de présentation. Cette entité est indépendante des API techniques et est constituée uniquement de Beans Java.
- **Vue.** Permet la présentation des données du modèle. Il existe plusieurs technologies de présentation, parmi lesquelles JSP/JSTL et XML, pouvant générer différents types de formats.
- **Contrôleur.** Gère les interactions avec le client tout en déclenchant les traitements appropriés. Cette entité interagit directement avec les composants de la couche service métier et a pour responsabilité la récupération des données mises à disposition dans le modèle. Lors de la mise en œuvre du type 2 de ce pattern, cette partie se compose d'un point d'entrée unique pour toute l'application et de plusieurs entités de traitement. Ce point d'entrée unique traite la requête et dirige les traitements vers l'entité Worker appropriée. Pour cette raison, l'entité Worker est habituellement appelée contrôleur. Le Front Controller, ou « contrôleur façade », est intégré au framework MVC, et seuls les workers sont spécifiques à l'application.

Un framework MVC implémente le contrôleur façade, les mécanismes de gestion du modèle ainsi que ceux de sélection et de construction de la vue. L'utilisateur d'un tel framework a en charge le développement et la configuration des workers.

Architecture et concepts de Struts

Comme expliqué précédemment, Struts est une implémentation classique du framework MVC de type 2, dont les caractéristiques sont les suivantes :

- **Modèle.** Constitué de JavaBeans standards. Dans notre cas, ils proviennent des couches inférieures de l'application (couche de service ou couche de domaine).
- **Vue.** Classiquement constituée de pages JSP. Cette partie peut aussi être prise en charge par d'autres technologies de présentation, telles que Velocity *(http://jakarta.apache.org/velocity/)* ou Cocoon *(http://cocoon.apache.org/),* tous deux de la fondaction Apache Jakarta.
- **Contrôleur.** Constitué de la servlet Struts principale, appelée `ActionServlet`.

Dans son implémentation, Struts utilise deux notions principales, qui sont représentées par les classes `org.apache.struts.action.Action`, appelées actions, et `org.apache.struts.action.ActionForm`, appelées `FormBeans`, ou formulaires :

- **Action.** Représente une action d'un utilisateur, telle que valider une sélection, mettre un produit dans un panier électronique, payer en ligne, etc. Cette classe proche du contrôleur sert de liant entre le formulaire envoyé par l'utilisateur, l'exécution de traitements dans les couches métier de l'application et l'affichage d'une page Web résultant de l'opération. Elle représente le worker introduit à la section précédente.

- **Formulaire.** Représente un formulaire HTML, tel qu'il a été rempli par l'utilisateur. Typiquement, ce formulaire a été codé en HTML avec des balises classiques de type `<input type='...'/>`. Struts reçoit et interprète en réalité les paramètres passés en GET ou en POST d'une requête HTTP. Il est donc possible de forger des URL, en JavaScript, par exemple, que Struts comprendra. Voici un exemple de paire clé/valeur passée en paramètre d'une Action Struts : *http://localhost:8080/example/test.do?clef=valeur.*

L'ensemble formé par les actions, formulaires et JSP est relié *via* le fichier de configuration de Struts, **struts-config.xml,** lequel est généralement stocké dans le répertoire **WEB-INF** de l'application Web. Dans le cas de Tudu Lists, ce fichier se trouve à l'emplacement **WEB-INF/struts-config.xml.**

Configuration de Struts

Struts se fonde sur une servlet très évoluée pour assurer sa fonction de contrôleur générique. Comme pour toute servlet, il faut la configurer dans le fichier **web.xml :**

```
( ... )
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>
    org.apache.struts.action.ActionServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
( ... )
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
( ... )
```

L'usage veut que l'on fasse correspondre les actions Struts à l'URL ***.do,** comme dans l'exemple ci-dessus. Cette extension en **.do** provient du verbe anglais *to do* (faire). Les actions Struts ont donc toutes une URL se terminant en **.do,** par exemple, *http://localhost/example/test.do,* de façon à être renvoyées à `ActionServlet`, qui les traite. Comme il s'agit d'une convention de nommage arbitraire, pour Tudu Lists nous avons préféré suffixer les actions par ***.action,** que nous trouvons plus parlant.

`ActionServlet`, quant à elle, est configurée *via* le fichier **struts-config.xml,** que nous avons présenté précédemment. Voici un exemple de ce fichier :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
➥Configuration 1.1//EN" "http://jakarta.apache.org/struts/dtds/struts-
➥config_1_1.dtd">
<struts-config>
  <form-beans>
    <form-bean
```

```
      name="exampleForm"
      type="tudu.web.form.ExampleForm"/>
  </form-beans>
  <global-exceptions>
    <exception  handler="tudu.web.TuduExceptionHandler"
                key="error.general"
                path="/WEB-INF/jsp/error.jsp"
                type="java.lang.Throwable"/>
  </global-exceptions>
  <global-forwards>
    <forward name="error" path="/WEB-INF/jsp/error.jsp"/>
  </global-forwards>
  <action-mappings>
    <action path="/test/example"
            name="exampleForm"
            type="tudu.web.action.ExampleAction"
            validate="true">

      <forward name="success"
               path="/WEB-INF/jsp/example.jsp"/>
    </action>
  </action-mappings>
  <message-resources parameter="messages"/>
</struts-config>
```

Ce fichier XML comporte les grandes parties suivantes :

- `<form-beans>`. Contient la liste des formulaires Struts. Nous donnons un nom à chaque formulaire, comme `exampleForm`, ainsi que le nom de la classe Java à laquelle il correspond. Il existe un type particulier de formulaire, dit `DynaBean` ou `DynaForm`, entièrement décrit en XML. Tudu Lists est majoritairement constitué de `DynaForms`, mais nous conseillons dans un premier temps l'utilisation de JavaBeans standards codés en Java. Presque aussi rapides à écrire, ils sont moins sujets à erreur au début.
- `<global-exceptions>`. En fonction de leur type, les exceptions lancées par les actions sont renvoyées par Struts à des classes spécifiques, qui doivent être spécifiquement implémentées. Cette partie est facultative.
- `<global-forwards>`. Donne des noms génériques à des pages JSP. Ces noms peuvent ensuite être utilisés depuis n'importe quelle action Struts. Dans notre exemple, nous pouvons renvoyer la page nommée `erreur` depuis toutes les actions, ce qui peut se révéler très utile. Cette partie est également facultative.
- `<action-mappings>`. Cette partie est la plus importante et la plus complexe. Elle permet de lier une URL à une action. Dans notre exemple, nous utilisons l'URL */test/example.action,* en supposant que le suffixe Struts tel que décrit plus haut est **.action.** Toute requête envoyée à cette URL est traitée par la classe `tudu.web.action.ExampleAction`, si elle est validée par le formulaire `exampleForm`.
- `<message-resources>`. Définit un fichier de propriétés, dans notre cas `messages.properties`. Situé à la racine du répertoire **JavaSource** de notre projet Eclipse, ce fichier

contient les messages utilisés dans les pages JSP et les actions. Ce fichier est recherché à la racine du classpath (typiquement dans **WEB-INF/classes**).

Ce fichier de configuration relativement complexe peut être édité graphiquement. Struts étant un standard du marché, de nombreux IDE supportent son utilisation. C'est le cas notamment de JDeveloper d'Oracle, un IDE gratuit qui propose une configuration graphique assez intuitive.

Si vous utilisez Eclipse, ou si vous n'utilisez pas d'IDE, nous vous conseillons l'utilisation de Struts Console, un utilitaire gratuit très bien conçu disponible à l'adresse *http://www.jamesholmes.com/struts/console/index.html*.

Actions et formulaires

Les actions et les formulaires sont les classes de base utilisées dans Struts.

Les actions Struts héritent toutes de `org.apache.struts.action.Action`. Cette classe représente une action standard, mais Struts est fourni avec un certain nombre d'actions plus évoluées, par exemple `org.apache.struts.actions.DispatchAction`, lesquelles restent cependant proches dans leur fonctionnement.

Voici un exemple d'action simple, provenant de Tudu Lists :

```
package tudu.web;

( ... )

/**
 * The Log out action.
 */
public class LogoutAction extends Action {

    private final Log log = LogFactory.getLog(LogoutAction.class);

    public final ActionForward execute(ActionMapping mapping,
        ActionForm form,
            HttpServletRequest request,
            HttpServletResponse response) {

        log.debug("Execute action");
        request.getSession().invalidate();
        Cookie terminate = new Cookie(TokenBasedRememberMeServices.ACEGI_SECURITY-
        ➥_HASHED_REMEMBER_ME_COOKIE_KEY, null);
        terminate.setMaxAge(0);
        response.addCookie(terminate);
        return mapping.findForward("logout");
    }
}
```

Comme toutes les actions Struts, `LogoutAction` implémente la méthode `execute()`, laquelle prend quatre arguments :

- `mapping` : correspond à la description faite précédemment dans le fichier **struts-config.xml** et permet de retrouver les vues telles quelles y ont été décrites dans les balises `<forward>`. Si nous reprenons la configuration décrite précédemment, nous pouvons rediriger l'utilisateur vers la page d'erreur en faisant un `mapping.findForward("error")`.
- `form` : correspond au formulaire Struts, tel qu'il a été décrit dans le fichier de configuration. Dans cet exemple très simple, il n'y a pas de formulaire associé, ce qui est une configuration correcte.
- `request`/`response` : correspondent aux objets classiques `HttpServletRequest` et `HttpServletResponse`, tels qu'ils existent pour les servlets.

L'objectif de l'action est d'effectuer un traitement, généralement réalisé par la couche métier, avec un Bean Spring ou un EJB, par exemple, puis de renvoyer l'utilisateur vers une nouvelle page Web. Cette page est représentée par l'objet `ActionForward`, que retourne l'action.

Les formulaires Struts héritent tous de la classe `org.apache.struts.action.ActionForm` et sont des JavaBeans standards.

Il s'agit de formulaires HTML créés dans une page Web. Lors de la validation d'un formulaire, Struts transforme la requête HTTP envoyée en cet objet Java, que nos pouvons ensuite plus facilement manipuler. Chaque paramètre de la requête HTTP est renseigné dans l'attribut du formulaire qui porte son nom.

Par exemple, pour la requête HTTP suivante :

```
http://localhost/Tudu/exampleAction.do?listId=001&name=test
```

le formulaire `ExampleForm` a ses attributs `listId` et `name` renseignés (méthodes `setListId` et `setName`), et nous pouvons accéder à ces valeurs *via* les méthodes `getListId` et `getName`.

Il faut cependant être conscient que ce formulaire est l'exacte représentation de ce qu'a envoyé l'utilisateur *via* son navigateur et que les données qu'il contient ne sont donc pas forcément valides. C'est pourquoi les formulaires Struts possèdent une méthode `validate`, qui est automatiquement appelée si, dans le fichier **struts-config.xml,** l'attribut `validate="true"` est mis à une action donnée, ce qui est le cas dans l'exemple de configuration ci-dessus.

Voici un exemple de formulaire inspiré de Tudu Lists :

```
package tudu.web.form;

( ... )

public class ExampleForm extends ActionForm {

    private String listId;
```

```
    private String name;

    public ActionErrors validate(ActionMapping mapping,    HttpServletRequest request) {

        ActionErrors errors = new ActionErrors();
        if (this.listId == null || this.listId.equals("")) {
            ActionMessage message = new ActionMessage(
                    "errors.required",  "Todo List ID");

            errors.add(ActionMessages.GLOBAL_MESSAGE, message);
        }
     if (this.name == null || this.name.equals("")) {
            ActionMessage message = new ActionMessage(
                    "errors.required",  "name");

            errors.add(ActionMessages.GLOBAL_MESSAGE, message);
        }
        return errors;
    }

    public String getListId() {
        return listId;
    }

    public void setListId(String listId) {
        this.listId = listId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Dans cet exemple, la méthode `validate` force l'utilisateur à renseigner les champs `listId` et `name` de son formulaire. Sans cela, la requête HTTP envoyée n'atteindrait même pas l'action.

Les bibliothèques de tags

Afin d'aider à l'affichage des pages JSP, Struts propose en standard les cinq bibliothèques de tags *(tag libraries)* suivantes :

- **Bean.** Sert à afficher et manipuler des JavaBeans dans une JSP. Développée avant l'existence des bibliothèques de tags standards, ou JSTL (JavaServer Pages Standard Tag Library), cette bibliothèque présente aujourd'hui peu d'intérêt. Rappelons que

depuis J2EE 1.4, nous pouvons utiliser les JavaBeans directement dans les Pages JSP, de la manière suivante :

```
${todo.todoList.name}
```

- **HTML.** D'excellente qualité, cette bibliothèque est l'une des raisons du succès de Struts. Très proche dans sa syntaxe des éléments de formulaire HTML, elle est intuitive et permet de réaliser facilement des formulaires Struts. Bien entendu, les formulaires HTML peuvent toujours être réalisés en HTML classique, Struts interprétant uniquement la requête HTTP envoyée. Cette bibliothèque n'est donc pas obligatoire mais fournit une aide intéressante.
- **Logic.** Sert à réaliser des boucles et des branchements conditionnels. Comme la bibliothèque Bean, elle n'est plus d'actualité depuis l'apparition de JSTL.
- **Nested.** Reprend l'ensemble des bibliothèques Struts, en leur donnant la capacité de s'imbriquer les unes dans les autres. Cela simplifie considérablement l'utilisation d'arbres d'objets complexes.
- **Tiles.** Permet l'inclusion et le paramétrage de fragments Tiles *(voir ci-après).*

En résumé, une grande partie des bibliothèques de tags de Struts ont été détrônées par JSTL. Reste principalement la bibliothèque HTML, qui permet de construire plus facilement des formulaires utilisés avec Struts. En voici un exemple, issu d'une simplification de la page **user_info.jsp** de Tudu Lists :

```
<h3><fmt:message key="user.info.title"/></h3>
<html:form action="/secure/myInfo" focus="firstName">
 <c:if test="${success eq 'true'}">
  <span class="success"><fmt:message key="form.success"/></span>
 </c:if>
 <html:errors/>
 <hr/>
  <fmt:message key="user.info.first.name"/> :
  <html:text property="firstName" size="15" maxlength="60"/>
 <br/>
  <fmt:message key="user.info.last.name"/> :
  <html:text property="lastName" size="15" maxlength="60"/>
 <br/>
  <fmt:message key="user.info.email"/> :
  <html:text property="email" size="30" maxlength="100"/>
 <hr/>
 <html:submit>
  <fmt:message key="form.submit"/>
 </html:submit>
 <html:submit><fmt:message key="form.cancel"/></html:submit>
</html:form>
```

La technologie Tiles

Tiles est une technologie de découpage de pages JSP qui s'appuie sur les `include` des JSP en les améliorant significativement. En particulier, Tiles permet d'effectuer les tâches suivantes :

- Créer aisément des modèles de pages réutilisables, avec un support de l'héritage entre différents modèles.
- Nommer les pages au lieu de donner leur chemin complet, dans l'esprit des tags `<forward>` vus précédemment dans la configuration de Struts.
- Réutiliser des composants de présentation, y compris des composants internationalisés (nous n'affichons pas le même composant suivant la langue de l'utilisateur).

Il y a donc de grandes chances pour que vous souhaitiez mettre Tiles en œuvre dans vos projets, d'autant plus que ce n'est guère compliqué.

Tiles fait partie intégrante du code source de Struts, bien qu'il s'agisse d'un composant optionnel, qui n'est pas activé par défaut. Pour l'activer, il suffit d'ajouter les lignes suivantes à la fin du fichier **struts-config.xml** (juste avant la balise de fermeture `</struts-config>`) :

```
<plug-in className=
"org.apache.struts.tiles.TilesPlugin">
  <set-property property=
  "definitions-config"
   value="/WEB-INF/tiles-defs.xml"/>
</plug-in>
```

Il suffit ensuite de créer le fichier **tiles-def.xml** dans le répertoire **WEB-INF.** Voici un exemple de ce fichier, dans lequel nous définissons un modèle simple (`layout`) et deux pages qui en héritent :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE tiles-definitions PUBLIC "-//Apache Software Foundation//DTD
➥Tiles Configuration 1.1//EN" "http://jakarta.apache.org/struts/dtds/
➥tiles-config_1_1.dtd">

<tiles-definitions>
 <definition name="layout" path="/WEB-INF/jsp/layout.jsp">
  <put name="title" value="Titre par defaut" />
  <put name="body" />
 </definition>
 <definition name="index" extends="layout">
  <put name="body" value="/WEB-INF/jsp/index.jsp" />
  <put name="title" value="Bienvenue" />
 </definition>
 <definition name="user" extends="layout">
  <put name="body" value="/WEB-INF/jsp/user.jsp" />
  <put name="title" value="Gestion des utilisateurs" />
 </definition>
</tiles-definitions>
```

Pour utiliser ces composants dans Struts, il faut modifier les `<forward>` dans **struts-config.xml.** Dans l'exemple précédent, pour rediriger vers le composant `index`, il faut mettre dans la configuration de l'action :

```
<forward name="success" path="index"/>
```

Nous utilisons comme chemin *(path)* le nom de la définition Tiles configurée dans le fichier **tiles-def.xml.** Bien entendu, nous pouvons toujours utiliser des pages JSP classiques à la place de Tiles. En effet, à chaque `forward`, Struts vérifie si une définition Tiles correspond au chemin demandé. S'il n'en trouve pas, il bascule en fonctionnement normal et peut alors rediriger vers une JSP classique.

Points faibles et problèmes liés à Struts

Struts a été le premier framework MVC à rencontrer un fort succès auprès de la communauté Java. Depuis sa création, fin 2000, ce framework a cependant relativement mal vieilli, et un certain nombre de problèmes sont apparus, notamment les suivants :

- Trop grande adhérence avec les classes `HttpServletRequest` et `HttpServletResponse`, qui sont passées en paramètre de la méthode `execute` des actions. Cela limite Struts à être utilisé avec des servlets et des pages JSP, si bien qu'il est impossible de réutiliser une couche Struts dans un client Swing.
- Utilisation de classes concrètes, et non d'interfaces, ce qui limite considérablement la possibilité de tester des actions Struts. En particulier, il est impossible d'utiliser des simulacres d'objets, ou mock objects, pour tester des actions Struts.
- Absence de mécanisme standard d'interception des actions. Une chaîne d'interception permettrait, par exemple, de gérer le logging, la sécurité ou l'ouverture des accès en base de données dans des objets spécialisés et transversaux aux actions.
- Erreurs de typage dans les formulaires, qui lancent des exceptions techniques critiques. Au final, les seuls attributs réellement utilisables pour les formulaires sont de type String. Cela ajoute un niveau de conversion avec les objets métier d'une complexité inutile.
- Prise en compte des seules données envoyées par les formulaires. Les données de référence, par exemple une liste à sélectionner, doivent être traitées séparément et mises en attribut de requête ou de session, de la même manière qu'avec les servlets. Cela lie encore plus le contrôleur à une couche de présentation codée en JSP.
- Nécessité, pour la bibliothèque de tags HTML, d'utiliser un éditeur de JSP supportant Struts. Sa manière de fonctionner l'empêche en effet d'être comprise par des éditeurs HTML couramment utilisés, tels que DreamWeaver.
- Séparation entre actions et formulaires peu justifiée, qui augmente le nombre de classes et de lignes à écrire.
- Absence de mécanisme d'accès à une couche métier. Dans la pratique, les actions Struts sont forcées d'aller elles-mêmes se connecter à la couche métier, par exemple en

utilisant JNDI dans le cadre d'une couche métier codée en EJB. Cela présente les inconvénients supplémentaires suivants :

– Gêne encore plus la testabilité des actions, puisqu'il est impossible d'exécuter un test JUnit si l'objet testé appelle un EJB dans ses méthodes.

– Lie les actions Struts à une implémentation de la couche métier et non à des interfaces.

– Limite l'utilisation de fonctionnalités avancées lors de l'accès à la couche métier, comme des proxy dynamiques ou de la POA pour gérer les transactions.

Ces désavantages, minimes au début de la vie de Struts, sont devenus aujourd'hui plus visibles : de nouveaux frameworks, mieux conçus, ont mis en lumière ces manquements. Citons parmi eux Spring MVC, que nous présentons au chapitre suivant, mais aussi Tapestry ou WebWork, avec lequel Struts est destiné à fusionner depuis fin 2005.

Struts et JSF (Java Server Faces)

JSF est l'API standard (JSR 127 du Java Community Process) d'affichage des composants dans des pages Web. Il ne s'agit pas d'un concurrent direct de Struts, puisqu'il se préoccupe uniquement de la partie présentation. Par ailleurs, rien n'empêche d'utiliser Struts en tant que contrôleur, au sens MVC, de l'application. Il n'y a donc rien de surprenant à la bonne intégration de ces deux technologies, qui sont de surcroît l'œuvre d'un même homme, Craig McClanahan.

JSF est davantage concurrent de la bibliothèque de tags HTML de Struts, mais rien n'empêche d'utiliser ces deux technologies conjointement.

Concernant l'avenir de Struts, il est certain que sa très grande popularité lui vaudra de rester encore longtemps d'actualité. L'un des objectifs de l'équipe qui l'a développé a toujours été d'assurer une bonne compatibilité entre les versions. C'est pourquoi elle a préféré une API stable à une évolution vers des concepts plus modernes. Dans la pratique, le succès de Struts lui a donné raison.

Cependant, Struts pourrait être détrôné à long terme par plusieurs autres technologies, notamment les suivantes :

- **Shale.** Nouvelle version de Struts, toujours développée par la fondation Apache et Craig McClanahan, résolvant la majorité des problèmes soulevés à l'encontre de Struts. Ne semble pas avoir encore provoqué d'engouement particulier.
- **Spring MVC.** Excellente implémentation du pattern MVC, entièrement intégrée à Spring.
- **AJAX.** Technique de codage d'une page Web très en vogue utilisant JavaScript pour charger à la demande des morceaux d'information insérés à chaud dans la page en cours. Tudu Lists en fait une utilisation intensive pour gérer les listes de todos et les todos eux-mêmes, et nous la présentons en détail au chapitre 9. Struts est mal adapté à ce type de programmation, qui possède ses propres frameworks, comme DWR, que nous présentons également au chapitre 9. Utilisé dans Tudu Lists, ce dernier permet d'utiliser directement des Beans Spring en JavaScript, ce qui court-circuite complètement Struts.

En résumé

Bien que toujours très populaire, Struts est un framework vieillissant. Parmi les nombreuses solutions de rechange plus évoluées techniquement disponibles sur le marché, citons Spring MVC, Tapestry ou WebWork.

Ces solutions ont en commun d'être fondées sur des conteneurs légers et de proposer des mécanismes d'interception. Nous verrons dans les sections qui suivent qu'en couplant Spring et Struts, il est possible d'obtenir les mêmes avantages.

Intégration de Struts à Spring

Bien que Spring possède son propre framework MVC, il s'intègre parfaitement avec Struts.

Nous verrons dans cette section de quelle manière cette intégration est réalisée et tenterons de dégager les bénéfices que nous pouvons en retirer, en particulier du point de vue des utilisateurs de Struts.

Intérêt de l'intégration de Struts à Spring

L'intérêt de l'intégration de Struts à Spring est d'ajouter à Struts un accès performant à la couche métier de l'application, Spring se chargeant de gérer la couche métier. Ce faisant, les actions Struts ne sont plus liées à des implémentations d'objets métier mais à leurs interfaces. Toutes les fonctionnalités de Spring sont disponibles lors de l'accès à cette couche métier, notamment l'utilisation de la POA pour gérer les transactions.

Spring vient ainsi combler une importante lacune de Struts, contribuant à moderniser ce framework.

Cette intégration est disponible sous trois formes différentes, chacune ayant ses avantages et ses inconvénients. Les sections qui suivent passent en revue chacune d'elles.

Configuration commune

Quel que soit le type d'intégration choisi, il faut tout d'abord ajouter un contexte Spring à Struts. Il s'agit en réalité d'un sous-contexte du contexte Spring général, qui se charge en tant que plug-in Struts.

Pour cela, il faut ajouter les lignes suivantes dans le fichier **struts-config.xml :**

```
<plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">

    <set-property  property="contextConfigLocation"
                   value="/WEB-INF/action-servlet.xml"/>
</plug-in>
```

Cette configuration provient de Tudu Lists, qui fournit un exemple d'intégration de Spring et de Struts. Habituellement, ce fichier de configuration Spring est nommé **action-servlet.xml.**

*Utilisation d'*ActionSupport

La manière la plus simple de connecter Struts à Spring consiste à utiliser la classe `org.springframework.web.struts.ActionSupport`. Il suffit pour cela de faire hériter ses actions de `ActionSupport` au lieu de `org.apache.struts.action.Action`.

L'action Struts en cours hérite alors de la méthode `getWebApplicationContext`, qui permet d'accéder au contexte d'application Spring configuré précédemment (en tant que plug-in Struts) :

```
package tudu.web;

( ... )

/** Backup a Todo List. */
public class BackupTodoListAction
    extends org.springframework.web.struts.ActionSupport {

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

        DynaActionForm todoListForm = (DynaActionForm) form;
        String listId = (String) todoListForm.get("listId");
        ApplicationContext ctx = getWebApplicationContext();
        TodoListsManager todoListsManager =
        (TodoListsManager) ctx.getBean("todoListsManager");
        TodoList todoList = todoListsManager.findTodoList(listId);
        Document doc = todoListsManager.backupTodoList(todoList);
        request.getSession()
        .setAttribute("todoListDocument", doc);
        return mapping.findForward("backup");
    }
}
```

Dans cet exemple, nous recherchons le Bean Spring `todoListsManager` dans le contexte Spring.

Spring propose des équivalents plus évolués des actions Struts, comme `DispatchAction` (`DispathActionSupport`). Le code reste proche de celui d'une action Struts classique et est simple à mettre en œuvre dans une application existante. Il ne permet toutefois pas de bénéficier pleinement des capacités de Spring au niveau de la couche MVC, comme l'inversion de contrôle pour injecter des Beans métier dans les actions. Pour cette raison, ce n'est pas la méthode que nous recommandons, en dépit de sa facilité d'utilisation attrayante.

Le DelegationRequestProcessor

Changer le `RequestProcessor` de Struts permet de découpler les actions Struts de Spring. Il n'y a donc plus d'import de classes Spring dans les actions, et les actions Struts deviennent de véritables Beans Spring, qui bénéficient de toute la puissance de ce framework (injection de dépendances, POA, etc.).

Pour changer le `RequestProcessor`, il faut ajouter un élément dans le fichier **struts-config.properties,** entre l'élément `<message-resources>` et l'élément `<plug-in>` :

```
<controller processorClass="org.springframework.web.struts.
   DelegatingRequestProcessor"/>
```

Une action Struts doit être définie en tant qu'action dans le fichier **struts-config.properties** et en tant que Bean Spring dans le fichier **action-servlet.xml.**

Dans **struts-config.properties :**

```
<action path="/secure/backupTodoList"
    name="todoListForm"
    type="tudu.web.BackupTodoListAction">

</action>
```

Dans **action-servlet.xml :**

```
<bean name="/secure/backupTodoList"
class="tudu.web.BackupTodoListAction">

   <property name="todoListsManager">
      <ref bean="todoListsManager" />
    </property>
</bean>
```

Notons que le nom du chemin *(path)* Struts est identique à celui du Bean Spring.

L'action vue précédemment bénéficie maintenant de l'inversion de contrôle et peut être réécrite ainsi :

```
package tudu.web;

( ... )

/** Backup a Todo List. */
public class BackupTodoListAction
    extends org.apache.struts.action.Action {

    private TodoListsManager todoListsManager = null;

    public final void setTodoListsManager(
    TodoListsManager todoListsManager) {

        this.todoListsManager = todoListsManager;
    }
```

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

        DynaActionForm todoListForm = (DynaActionForm) form;
        String listId = (String) todoListForm.get("listId");
        TodoList todoList = todoListsManager.findTodoList(listId);
        Document doc = todoListsManager.backupTodoList(todoList);
        request.getSession()
        .setAttribute("todoListDocument", doc);
        return mapping.findForward("backup");
    }
}
```

La seule faiblesse de cette méthode est d'utiliser un `RequestProcessor` spécifique. Cela pose des problèmes aux applications qui utilisent leur propre `RequestProcessor`, une technique couramment utilisée pour étendre Struts.

Ainsi, pour l'utilisation de Tiles, qui utilise lui aussi son propre `RequestProcessor`, une classe `DelegatingTilesRequestProcessor` est fournie par Spring. Mais cet exemple met en lumière la nécessité de coder des `RequestProcessor` très spécifiques afin de lier Struts et Spring.

La délégation d'actions

La délégation d'actions est la méthode que nous conseillons et que nous utilisons avec Tudu Lists.

Proche de la méthode précédente, elle permet de gérer les actions Struts depuis Spring comme des JavaBeans, mais sans impact sur la configuration de Struts. L'idée est d'utiliser une classe spécifique, `org.springframework.web.struts.DelegatingActionProxy`, afin de déléguer la gestion de l'action Struts à Spring.

Dans la pratique, nous continuons à coder des actions Struts normalement (ces classes peuvent même hériter d'actions complexes, comme les `DispatchAction`). La seule différence est que, dans **struts-config.xml,** nous donnons `org.springframework.web.struts.DelegatingActionProxy` comme étant la classe de l'action.

Par exemple, dans Tudu Lists, l'action `"/register"` est configurée de la façon suivante :

```
<actionpath="/register"
    name="registerForm"
    type="org.springframework.web.struts.DelegatingActionProxy"
    parameter="method"
    validate="false"
    input="/WEB-INF/jsp/register.jsp">

<forward name="register" path="/WEB-INF/jsp/register.jsp"/>
<forward name="success" path="/WEB-INF/jsp/register_ok.jsp"/>
     <forward name="cancel" path="/welcome.action" redirect="true"/>
</action>
```

Dans le fichier de configuration Spring **action-servlet.xml,** cette action est configurée de la façon suivante :

```
<bean name="/register" class="tudu.web.RegisterAction">
    <property name="userManager">
        <ref bean="userManager" />
    </property>
</bean>
```

Spring réalise le lien avec l'action Struts *via* le nom du Bean, identique au chemin de l'action Struts `"/register"`.

De même que pour la méthode précédente, `RegisterAction` est à la fois une action Struts complète, qui hérite de `DispatchAction`, et un Bean Spring à part entière, bénéficiant de l'inversion de contrôle.

En résumé

Il est relativement facile d'intégrer Struts et Spring et de combiner ainsi les avantages de chaque framework :

- Struts apporte une couche MVC simple et bien connue des développeurs.
- Spring apporte un accès simple et non intrusif à une couche métier performante.

Nous avons vu qu'il était possible d'avoir des classes étant à la fois des actions Struts et des Beans Spring. Ces « nouvelles actions Struts » ont des capacités bien supérieures aux actions classiques, réduisant ainsi les faiblesses de Struts.

Grâce à la délégation d'actions, ces nouvelles classes bénéficient des avantages suivants :

- Elles ne lient plus la couche de présentation à une implémentation donnée de la couche métier.
- Elles sont plus facilement testables de manière unitaire du point de vue de l'accès à la couche de service. Elles dépendent toutefois toujours des API servlet et Struts et restent donc difficiles à tester.
- Elles disposent d'un mécanisme d'interception. Ces classes étant des Beans Spring, nous pouvons utiliser la POA pour intercepter leurs méthodes. Cela peut permettre d'ajouter des mécanismes transversaux de sécurité, de monitoring, de logging, de cache, etc.

Tudu Lists : intégration de Struts

Reprenant notre étude de cas Tudu Lists, nous allons mettre en pratique l'intégration de Spring et de Struts en utilisant la délégation d'actions.

Pour Tudu Lists, nous avons fait le choix d'une couche de présentation MVC avec Struts, enrichie de l'intégration avec Spring. Malgré ses défauts, nous avons préféré utiliser

Struts en standard avec Tudu Lists en raison de sa très large diffusion et parce que nous en connaissons bien les rouages.

Pour cette application particulière, nous n'avons pas besoin d'une couche de présentation très élaborée, capable, par exemple, de gérer des workflows complexes. Si cela avait été le cas, nous aurions opté pour l'utilisation conjointe de Struts et d'une technologie plus adaptée, par exemple Spring Web Flow.

Concernant la partie vue de l'application, nous avons choisi d'utiliser des pages JSP dans le cas général, en utilisant au maximum la JSTL (JavaServer Pages Standard Tag Library). Nous utilisons aussi les nouvelles notations apportées par J2EE 1.4, de type `${javabean.attribut}`, ce qui réduit et clarifie considérablement le code. Dans certains cas très particuliers, nous avons utilisé des servlets : c'est le cas de la génération de flux RSS. Ce flux étant généré par Rome, un framework Open Source spécialisé dans la gestion de ce type de flux, l'utilisation d'une JSP n'a pas d'intérêt.

Tudu Lists utilise aussi la technologie AJAX (Asynchronous Javascript And XML), qui permet de communiquer en XML avec le serveur sans recharger la page Web en cours. Une partie de la couche de présentation a donc été réalisée avec DWR, un framework permettant d'utiliser des Beans Spring côté serveur directement en JavaScript côté client. Cette technique est détaillée au chapitre 9, dédié à AJAX.

Les fichiers de configuration

La configuration de la partie présentation de Tudu Lists est répartie dans les fichiers suivants :

- **WEB-INF/web.xml.** Fichier standard de configuration des applications Web J2EE servant ici à configurer la servlet `ActionServlet` de Struts.
- **WEB-INF/struts-config.xml.** Fichier de configuration de Struts que nous avons étudié précédemment. Tudu Lists utilisant la délégation d'actions, l'ensemble des actions décrites dans ce fichier est de type `org.springframework.web.struts.DelegatingActionProxy`. Afin de limiter le nombre de formulaires Struts, nous utilisons des `DynaForm`. Uniquement décrits en XML dans le fichier **struts-config.xml,** ces formulaires n'ont pas d'implémentation Java.
- **WEB-INF/validation.xml.** Fichier de configuration du Validator de Struts. Ce dernier est un plug-in Struts aidant à la validation de formulaires en fournissant des implémentations Java et JavaScript des principales règles de validation trouvées dans une application Web (champ obligatoire, tailles minimale et maximale, champ e-mail, etc.).
- **WEB-INF/validator-rules.xml.** Fichier de configuration interne du Validator, n'ayant normalement pas à être modifié.
- **WEB-INF/action-servlet.xml.** Fichier contenant une configuration Spring standard, dans laquelle les actions Struts sont définies en tant que Beans Spring. Cela permet de configurer l'injection de dépendances dont elles bénéficient.

- **JavaSource/messages.properties.** Fichier stockant les messages utilisés dans l'application. Il s'agit d'un fichier de propriétés, aussi appelé *resource bundle,* permettant de gérer plusieurs langues.

Exemple d'action Struts avec injection de dépendances

Pour cet exemple, nous allons utiliser `tudu.web.MyInfoAction`, une action qui permet à l'utilisateur de gérer ses informations personnelles.

Lors de l'utilisation de cette action, le parcours de l'utilisateur est le suivant :

1. L'utilisateur exécute l'action `MyInfoAction` sans lui envoyer de paramètre.
2. Les informations concernant l'utilisateur en cours sont recherchées en base de données.
3. La page **WEB-INF/jsp/user_info.jsp** est affichée. Elle contient les informations récupérées précédemment, prêtes à être modifiées.
4. L'utilisateur modifie ces informations. Il peut ensuite soit annuler ses changements (retour à l'étape 1), soit les valider et passer à l'étape 5.
5. Si les informations envoyées ne sont pas correctes, l'utilisateur est renvoyé à l'étape 3.
6. Les informations sont sauvegardées en base de données, et nous revenons à l'étape 1.

La classe `MyInfoAction` est un groupement de trois actions différentes : l'affichage de la page, l'annulation du formulaire et sa validation.

Dans les premières versions de Struts, il fallait trois classes différentes pour représenter ces trois actions, ce qui obligeait à écrire beaucoup de code. C'est désormais simplifié grâce à la classe `org.apache.struts.actions.DispatchAction`, dont `MyInfoAction` hérite *via* `tudu.web.TuduDispatchAction`.

Ainsi, `MyInfoAction` est plus un groupe d'actions qu'une action, ce qui a pour effet de réduire le nombre de classes et la taille du fichier de configuration.

La signature de ses méthodes est la suivante :

```
package tudu.web;

( ... )

public class MyInfoAction extends TuduDispatchAction {

    public final void setUserManager(UserManager userManager) {
        ( ... )
    }

    public final ActionForward display(
    ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response) {
```

```
    ( ... )
    }

    public final ActionForward update(
    ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response) {

        ( ... )
    }
}
```

La première méthode, `setUserManager`, est utilisée pour l'injection de dépendances avec Spring.

Les deux autres méthodes, `display` et `update`, correspondent aux actions Struts dont nous avons précédemment parlé, qui servent à afficher et à mettre à jour les informations utilisateur. Il existe également une méthode `cancel`, dont `MyInfoAction` hérite, qui correspond à l'annulation de l'action. En réalité, cette méthode ne fait que rediriger vers la méthode `display`. En effet, une annulation effectue un nouvel affichage de la page initiale, avec les données d'origine.

Ces actions sont configurées *via* le fichier **struts-config.xml :**

```
<form-bean name="userForm"
        type="org.apache.struts.validator.DynaValidatorForm">

<form-property name="password" type="java.lang.String"/>
<form-property name="verifyPassword" type="java.lang.String"/><form-property
➥name="firstName" type="java.lang.String"/>
<form-property name="lastName" type="java.lang.String"/>
<form-property name="email" type="java.lang.String"/>
</form-bean>

( ... )
<action
    path="/secure/myInfo"
    name="userForm"
    type="org.springframework.web.struts.DelegatingActionProxy"
    parameter="method"
    validate="false"
    input="/WEB-INF/jsp/user_info.jsp">

<forward name="user.info" path="/WEB-INF/jsp/user_info.jsp"/>
</action>
```

`MyInfoAction` peut exécuter chacune des méthodes `display`, `update` et `cancel` en fonction d'un paramètre qui lui est envoyé. Ce paramètre, configuré plus haut en tant qu'attribut de l'action, est défini à `"method"`. Cela signifie qu'un paramètre HTTP `"method"` va être envoyé avec le formulaire et qu'en fonction de ce paramètre une méthode sera exécutée.

On retrouve ce paramètre dans la page JSP **WEB-INF/jsp/user_info.jsp** (le code suivant est volontairement simplifié afin de mettre en valeur l'essentiel) :

```
<html:form action="/secure/myInfo" focus="firstName">
<html:errors/>
<html:hidden property="method" value="cancel"/>

<fmt:message key="user.info.first.name"/>
<html:text property="firstName" size="15" maxlength="60"/>
<br/>
<fmt:message key="user.info.last.name"/>
<html:text property="lastName" size="15" maxlength="60"/>
<br/>
<fmt:message key="user.info.email"/>
<html:text property="email" size="30" maxlength="100"/>
<br/>
<fmt:message key="user.info.password"/>
<html:password property="password" size="15" maxlength="32"/>
<br/>
<html:submit
onclick="document.forms[0].elements['method'].value='update';">

<fmt:message key="form.submit"/>
</html:submit>
<html:submit><fmt:message key="form.cancel"/></html:submit>
</html:form>
```

L'utilisation du champ HTML `"method"`, qui est un champ caché, ainsi que celle de JavaScript dans l'événement `"onClick"` du bouton `"Submit"`, permettent d'envoyer avec le formulaire une variable supplémentaire.

L'envoi du formulaire permet ainsi d'envoyer les variables `"firstName"`, `"lastName"`, `"email"`, `"password"` et `"method"`. C'est cette dernière variable qui sera utilisée par Struts pour déterminer quelle méthode utiliser dans `MyInfoAction`.

Intégration de Spring et de Struts

Dans le fichier **struts-config.xml,** l'action est configurée en tant que `DelegatingActionProxy`. Nous utilisons donc la troisième forme d'intégration Struts-Spring présentée précédemment, à savoir la délégation d'actions.

Nous retrouvons par conséquent le JavaBean `MyInfoAction` configuré dans le fichier **WEB-INF/action-servlet.xml :**

```
<bean name="/secure/myInfo" class="tudu.web.MyInfoAction">
<property name="userManager">
    <ref bean="userManager" />
</property>
</bean>
```

Grâce à cette configuration, le JavaBean `userManager` est injecté dans `MyInfoAction`. Nous pouvons l'utiliser directement dans `MyInfoAction`, par exemple dans la méthode `display` :

```
public final ActionForward display(
    ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response) {

        log.debug("Execute display action");
        String login = request.getRemoteUser();
        User user = userManager.findUser(login);
        DynaActionForm userForm = (DynaActionForm) form;
        userForm.set("password", user.getPassword());
        userForm.set("verifyPassword", user.getPassword());
        userForm.set("firstName", user.getFirstName());
        userForm.set("lastName", user.getLastName());
        userForm.set("email", user.getEmail());
        return mapping.findForward("user.info");
    }
```

Dans cet exemple, nous avons donc bien une action Struts complète relativement évoluée, notamment grâce à l'utilisation de `DispatchAction`, qui s'intègre parfaitement dans Spring en tant que JavaBean standard et bénéficie ainsi de l'injection de dépendances.

Utilisation conjointe des DynaBeans et du Validator

Tudu Lists utilise deux fonctionnalités avancées de Struts que nous n'avons pas encore détaillées, les DynaBeans et le Validator. Utilisées conjointement, elles réduisent la complexité de l'application en supprimant la majeure partie du code nécessaire au codage des formulaires.

Coder des formulaires Struts devient rapidement répétitif puisque nous nous retrouvons avec un grand nombre de JavaBeans qui n'ont pas grande utilité. De plus, la validation de ces formulaires est fastidieuse, car il faut chaque fois coder la méthode `validate()` des formulaires, un travail le plus souvent long, même pour un résultat simple.

Voici un exemple de méthode `validate()` qui teste si un champ obligatoire a été renseigné :

```
public ActionErrors validate(
ActionMapping mapping, HttpServletRequest request) {

ActionErrors errors = new ActionErrors();
if (this.todoId == null || this.todoId.equals("")) {
    ActionMessage message = new ActionMessage(
                    "errors.required",  "Todo ID");

    errors.add(ActionMessages.GLOBAL_MESSAGE, message);
}
return errors;
}
```

Ce code est incomplet, car il ne représente qu'une validation côté serveur, alors que bien souvent une validation côté client en JavaScript est nécessaire afin de ne pas surcharger le serveur de requêtes contenant des informations potentiellement non valides.

Les formulaires Struts se révélant longs et répétitifs à coder, les DynaBeans et le Validator ont été conçus pour soulager le développeur.

Un DynaBean est un formulaire Struts décrit uniquement en XML dans le fichier **struts-config.xml.** Nous en avons déjà rencontré un dans le code décrivant le formulaire `user-Form` :

```
<form-bean name="userForm"
        type="org.apache.struts.validator.DynaValidatorForm">

<form-property name="password" type="java.lang.String"/>
<form-property name="verifyPassword" type="java.lang.String"/><form-property
name="firstName" type="java.lang.String"/>
<form-property name="lastName" type="java.lang.String"/>
<form-property name="email" type="java.lang.String"/>
</form-bean>
```

Malheureusement, une des limitations de Struts vient de ce que le type des propriétés des DynaBeans est presque obligatoirement `java.lang.String`. Struts renvoie des erreurs fatales en cas de non-correspondance entre une propriété envoyée par le navigateur client et son type.

Le Validator permet quant à lui de valider automatiquement le formulaire côté serveur comme côté client.

Cette validation est définie dans le fichier **WEB-INF/validation.xml :**

```
<form name="userForm">
<field property="password" depends="required">
    <arg0 key="user.info.password" />
</field>
<field property="verifyPassword" depends="validwhen">
    <arg0 key="user.info.password.not.matching" />
    <var>
        <var-name>test</var-name>
        <var-value>(*this* == password)</var-value>
    </var>
</field>
<field property="firstName" depends="required">
    <arg0 key="user.info.first.name" />
</field>
<field property="lastName" depends="required">
    <arg0 key="user.info.last.name" />
</field>
<field property="email" depends="email">
    <arg0 key="user.info.email" />
</field>
</form>
```

Dans cet exemple, nous pouvons définir des champs comme `"required"` (obligatoire), `"validwhen"` (valable en fonction d'un test) ou `"email"` (s'il s'agit d'un e-mail valide). Un certain nombre de validations standards, en particulier sur le type et la taille des champs, sont en outre configurées dans le fichier **WEB-INF/validator-rules.xml.** Il est bien entendu possible d'ajouter des validations supplémentaires, mais Struts fournit par défaut une bibliothèque de règles de validation assez complète.

Cette validation fonctionne automatiquement côté serveur et peut être ajoutée côté client en insérant simplement le tag suivant dans la page JSP de formulaire :

```
<html:javascript formName="userForm"/>
```

L'ajout de ce tag permet de générer à la volée le JavaScript nécessaire à la validation du formulaire passé en paramètre.

Dans Tudu Lists, nous utilisons cette double technique des DynaBeans et du Validator afin de réduire la taille du code à écrire et réaliser l'application le plus rapidement possible.

Cette solution n'est toutefois pas à recommander dans tous les cas, notamment pour les raisons suivantes :

- Le temps de codage des formulaires n'est pas toujours très long. Par exemple, coder un JavaBean avec Eclipse est très rapide, en particulier grâce au menu Source/Generate Getters and Setters.
- Les DynaBeans font perdre la garantie de la compilation et ne permettent plus d'utiliser les fonctionnalités de refactoring d'Eclipse pour renommer un champ.
- Très utile pour les validations simples, le Validator devient soit trop complexe pour les validations plus avancées (plus simples à réaliser en Java), soit complètement inefficace pour les validations métier (qui nécessitent de toute manière d'être exécutées au niveau de l'action).

Dans une application, la technique décrite précédemment n'est valable que pour les formulaires simples, pour lesquels elle apporte des gains de temps et de taille de code.

Dans le cas particulier de Tudu Lists, la majorité des formulaires est assez simple. Il s'agit soit de pages aux fonctionnalités élémentaires (comme l'action `MyInfoAction`), soit de pages plus complexes dans lesquelles les fonctionnalités avancées sont déportées dans la couche AJAX.

Création d'un intercepteur sur les actions Struts

Outre l'injection de dépendances, l'un des intérêts d'intégrer Spring et Struts est d'utiliser des intercepteurs Spring sur les actions Struts.

Pour cet exemple, nous allons utiliser l'un des intercepteurs fourni en standard avec Spring, `org.springframework.aop.interceptor.DebugInterceptor`.

Cet intercepteur est configuré dans le fichier **WEB-INF/action-servlet.xml :**

```
<bean id="debugInterceptor"
 class="org.springframework.aop.interceptor.DebugInterceptor">

  <property name="loggerName">
    <value>tudu.interceptor.debug</value>
  </property>
</bean>

<bean    name="proxyCreator"
 class="org.springframework.aop.framework
            .autoproxy.BeanNameAutoProxyCreator">

  <property name="beanNames" value="/*"/>
  <property name="interceptorNames">
    <list>
        <value>debugInterceptor</value>
    </list>
  </property>
</bean>
```

Concernant le Bean `debugInterceptor`, il s'agit de la configuration classique d'un intercepteur Spring. Le `"proxyCreator"` est configuré pour correspondre au nom `"/*"`, ce qui permet d'intercepter l'ensemble des appels aux actions Struts.

Cette configuration permet d'obtenir des messages de log particulièrement intéressants pour déboguer une application, comme ici :

```
DEBUG tudu.interceptor.debug  - Exiting invocation: method 'execute',
arguments [ActionConfig[path=/secure/admin/administration,name=adminis-
trationForm,parameter=method,scope=request,type=org.springframe-
work.web.struts.DelegatingActionProxy,
DynaActionForm[dynaClass=administrationForm,smtpHost=smtp.test.com,smtpFr
om=email@test.com,smtpPassword=password,smtpPort=25,smtpUser=user],
net.sf.acegisecurity.wrapper.ContextHolderAwareRequestWrapper@676316,
org.apache.catalina.connector.ResponseFacade@9a6c56]; target is of class
[tudu.web.AdministrationAction]; count=4
```

Nous retrouvons dans ce message de log l'URL de l'action appelée, ainsi que l'ensemble des paramètres envoyés, la classe exécutée et le nombre d'appels à cette URL.

Ces messages apparaissant en mode debug, il faut, pour les observer, configurer le fichier **log4j.properties** (dans le répertoire **JavaSource**) de la manière suivante :

```
log4j.logger.tudu.interceptor.debug=DEBUG
```

Des intercepteurs peuvent être ajoutés ou enlevés très facilement. Ils servent généralement à monitorer l'application.

Points forts et points faibles de la solution

Cette solution permet de combiner les forces de Struts et de Spring :

- Elle est la plus rapide et la moins coûteuse à mettre en œuvre, car tout développeur maîtrisant Struts est presque immédiatement opérationnel.
- Du fait de l'injection de dépendances et des intercepteurs, l'utilisation de Struts est considérablement améliorée par Spring.

Cependant, elle ne résout pas tous les problèmes liés à Struts :

- Une grande partie des problèmes provenant de Struts restent présents, notamment l'utilisation de classes et non d'interfaces, la dépendance directe à l'API des servlets, etc.
- La configuration nécessaire est complexe et nécessite une bonne connaissance du fichier **struts-config.xml** et de la configuration de Spring.
- Les objets ainsi créés sont de mauvaise qualité du point de vue de Spring. Ils dépendent à la fois de l'API servlet et de l'API Struts, les rendant difficiles à tester.

Conclusion

Après une rapide présentation de Struts, ce chapitre a détaillé les nombreux problèmes liés à ce framework populaire mais vieillissant.

L'intégration de Spring et de Struts permet de bénéficier des avantages de chacune de ces deux technologies, notamment la vaste communauté d'utilisateurs de Struts et l'injection de dépendances et la POA proposées par Spring. La combinaison de ces deux technologies permet de réaliser une couche MVC de bonne qualité, sans nécessiter la maîtrise des nouveaux et complexes frameworks MVC à la mode.

Nous avons étudié les trois méthodes classiques d'intégration de Spring et de Struts, en constatant que la méthode de délégation d'action était généralement la plus intéressante. Au prix d'une configuration un peu lourde, il est possible d'avoir des objets qui soient à la fois des actions Struts et des Beans Spring, bénéficiant de la sorte des deux technologies.

Ce choix reste un compromis puisqu'il ne résout pas tous les problèmes liés à Struts et ne permet pas de créer des Beans Spring de bonne qualité, ces derniers dépendant des API servlet et Struts et restant difficilement testables unitairement. Il permet néanmoins de combiner intelligemment les forces de ces deux frameworks et d'obtenir un résultat rapide sans prendre trop de risques.

Il s'agit donc d'un bon compromis pour les équipes possédant déjà des compétences Struts et ne désirant pas passer à un autre framework MVC dans l'immédiat.