8

Développement Web avec le framework JBoss Seam

Ce chapitre complète l'outillage Web Tools avec Seam, un framework « Web 2.0 Ready », qui facilite le développement JEE, en particulier pour les applications fondées sur les technologies JSF et EJB3.

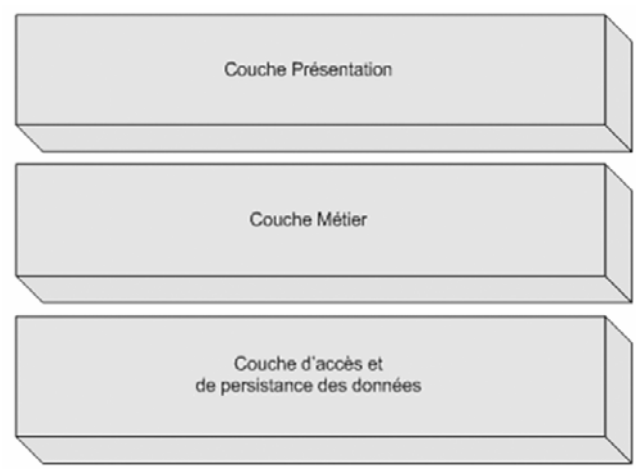
Après un bref rappel de la genèse des frameworks JEE et des fondamentaux du framework Seam, vous mettrez en œuvre ce dernier sur l'étude de cas.

Les frameworks J2EE

Depuis la fin des années 1990, les best practices de développement Java/J2EE, ont imposé, bien avant l'apparition des frameworks tels que Struts ou Spring, une approche de développement en couches : couche présentation, couche métier et couche d'accès aux données et de persistance, comme l'illustre la figure 8.1.

Figure 8.1

Architecture J2EE/ JEE standard



Cette approche en couches s'appuyait sur les API de base des servlets. Une servlet traite les requêtes Web en entrée et affiche le résultat sous forme de balises HTML, qui permettent l'affichage de pages au sein d'un navigateur. L'inconvénient de cette approche était que les balises HTML chargées de la présentation étaient mêlées à du code Java, ce qui impactait fortement la maintenabilité de l'ensemble et ne permettait pas une répartition efficace des rôles au sein de l'équipe projet (designer de pages Web et développeurs Java).

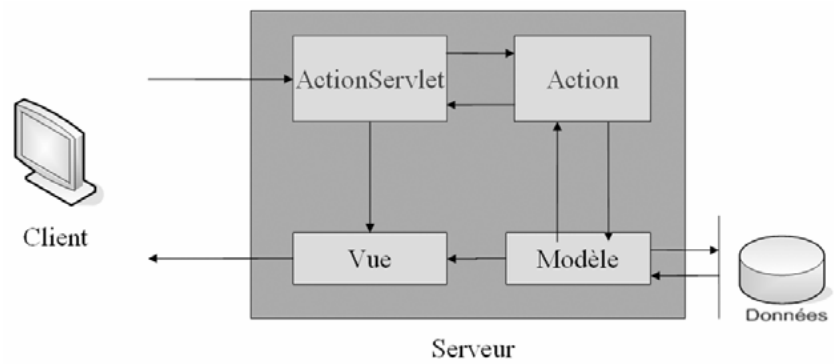
Est ensuite venue la technologie JSP (JavaServer Pages), qui permet aux développeurs d'embarquer des éléments dynamiques au sein de pages HTML. Ces éléments dynamiques utilisent des balises spécifiques (directive d'import, de code Java à exécuter, de résultats à afficher, etc.), ainsi que des variables implicites (request, response, session, etc.).

Bien que mieux structurée que les servlets, cette approche de développement présentait plusieurs inconvénients, le principal d'entre eux étant le manque de lisibilité du code du fait du mélange du HTML et du Java, d'où une confusion entre la présentation et la logique applicative.

Le framework Open Source Struts de la fondation Apache est ensuite apparu au début des années 2001. Dédié à la conception d'applications Web à base de servlets, de JSP et de JavaBeans, ce framework fonctionne schématiquement comme illustré à la figure 8.2.

Figure 8.2

Architecture de Struts



Conformément au modèle appelé MVC2, Struts s'appuie sur une classe principale, la classe org.apache.struts.action.ActionServlet, qui assure le rôle de contrôleur. Celle-ci est associée à toutes les requêtes HTTP se terminant par .do *via* la configuration du fichier de mapping associé (balise <servlet-mapping>). En plus de cette classe, un fichier de configuration struts-config.xml contient une liste d'URL dédiées chacune à une action. Pour transférer les données du formulaire à l'action, Struts utilise une instance dérivée d'ActionForm. Le lien entre cette instance, l'URL demandée et l'action est aussi défini dans le fichier de configuration Struts.

Côté modèle, Struts n'impose rien, et aucune classe n'est fournie, laissant une totale liberté de choix des technologies à utiliser pour l'implémentation du modèle (ce qui est aussi un inconvénient dans la mesure ou aucun standard n'est proposé pour organiser cette partie du développement).

Peu de temps après sont apparues les premières implémentations Sun des spécifications JSF (JavaServer Faces) — JSR-000127. Conçues pour le développement rapide d'applications Web, ces spécifications proposent une interface graphique riche associée à une logique de programmation événementielle. Pour cela, la norme définit un cycle normalisé de

traitement des requêtes et fournit une abstraction complète du protocole HTTP. En découle une gestion transparente de l'état des composants.

En comparaison de JSP, de l'API servlet et de Struts, qui oblige les développeurs à produire beaucoup de code pour gérer les aspects présentation, interaction et ergonomie, les composants JSF prennent en charge automatiquement ces opérations.

Limitations de Struts et JSF

Le framework Struts est aujourd'hui déprécié du fait des nouveaux besoins des applications Web 2.0 complexes. Les raisons à cela sont nombreuses. Citons en vrac l'absence de formulaires de type stateful permettant la conservation des états et une gestion du contexte évolué, le fait qu'une seule JSP soit associée à un unique form bean (bean associé à un formulaire), qu'une seule classe Action puisse mapper un form bean pour chaque configuration, sauf à passer par des solutions de contournement difficilement maintenables, etc.

Ces limitations ont favorisé l'émergence de JSF, qui propose une approche plus orientée composant, se rapprochant d'une approche davantage RAD et à la .Net. Par contre, toute la « plomberie » nécessaire à l'invocation des méthodes et des services métier avec EJB3 ainsi que la gestion des sessions reste à la charge du développeur, ce qui complexifie grandement le développement d'applications JEE.

Web 2.0

Le concept de Web 2.0 est vu comme la promesse d'une nouvelle version de l'Internet. Selon la définition de l'éditeur Tom O'Reilly, le Web 2.0 repose sur un ensemble de modèles de conception. Ces systèmes architecturaux plus intelligents permettent d'utiliser des modèles légers qui rendent possible la syndication des données et des services. « Le Web 2.0, dit-il, c'est le moment ou les gens réalisent que ce n'est pas le logiciel qui fait le Web mais les services. »

JSF (JavaServer Faces)

Un bref rappel des fondamentaux de la technologie JSF n'est pas superflu ici. Le lecteur débutant dans cette technologie est invité à examiner les nombreux ouvrages sur le sujet, notamment ceux cités en annexe.

JSF est devenu un standard du JCP pour le développement d'applications Web, en particulier pour le développement de la couche présentation. Cet état de fait conforte le constat que Struts est déprécié pour le développement d'applications Web 2.0 performantes.

JSF fournit les caractéristiques remarquables suivantes :

- Gestion simplifiée de la navigation interpage.
- Modèle de composants graphiques standards (widgets).
- Gestion des erreurs et de la validation des formulaires.
- Gestion événementielle évoluée, qui se rapproche de Visual Basic.
- Gestion évoluée des composants JavaBeans (avec la notion importante de backing beans, spécialement utilisés dans JSF pour encapsuler certaines parties des composants qui constituent une page et faire le lien avec la couche métier). Les backing beans

sont utiles pour afficher les données provenant de la couche métier ou pour la saisie des données de l'utilisateur.

- Support de l'internationalisation.

Toutes ces caractéristiques permettent au développeur de se concentrer sur son application et de se décharger de la gestion rébarbative des pages et des composants graphiques qui la composent.

Mise en œuvre de JSF avec Eclipse Web Tools

Dans cette section, vous allez créer et exécuter une application JSF simple de type HelloWord destinée à illustrer les composants d'une application Web fondée sur cette technologie et à évaluer la richesse des outils de support au développement JSF offerts par le projet Web Tools.

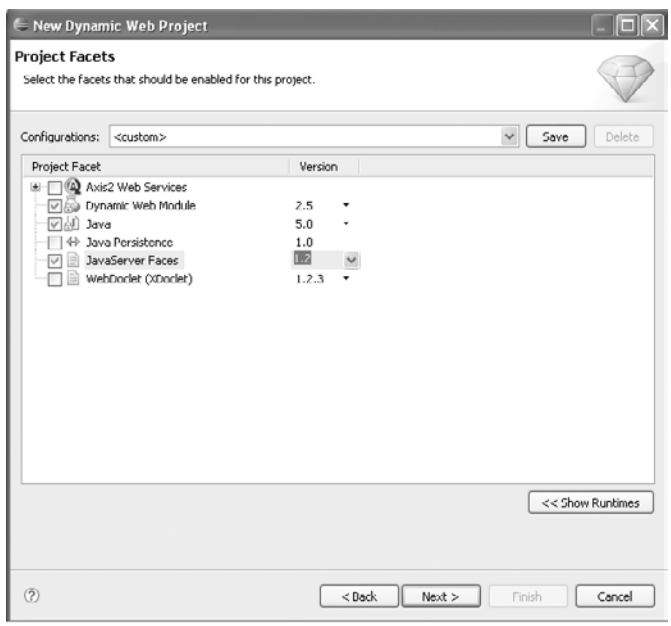
Le serveur JBoss 4.2 dispose d'un support natif de l'implémentation de référence de Sun (JSF 1.2). Cela vous épargnera les étapes d'intégration et de configuration des bibliothèques JSF dans le projet.

Nous supposons la cible définie dans la configuration de l'IDE (menu Preferences).

1. Créez un nouveau projet Web de type dynamique (options d'Eclipse File, New, Other, Web et Dynamic Web Project).
2. Appelez-le HelloJSF, puis cliquez sur Next (voir figure 8.3).

Figure 8.3

Configuration du projet avec les fonctionnalités Facet de Web Tools



Vous pouvez constatez que Web Tools propose les implémentations de référence de JSF.

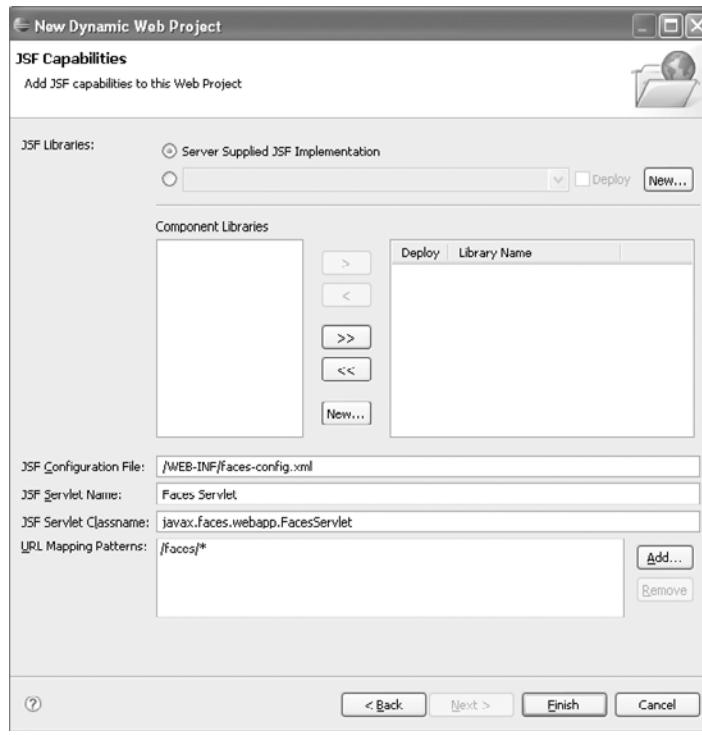
3. Cochez cette option, et sélectionnez la version 1.2, puis cliquez sur Next.
4. Les champs indiquant le contexte root de l'application et les répertoires Web (WebContent) et source (src) sont proposés. Cliquez sur Next.

La boîte de dialogue suivante de l'assistant de création de projet Web dynamique propose de configurer les bibliothèques JSF ainsi que l'emplacement des différents fichiers de configuration JSF du projet.

5. Sélectionnez lServer Supplied JSF Implementation, et cliquez sur Finish (voir figure 8.4).
6. Cliquez sur I Agree dans la boîte de dialogue de copyright concernant la version des bibliothèques JSF utilisées.

Figure 8.4

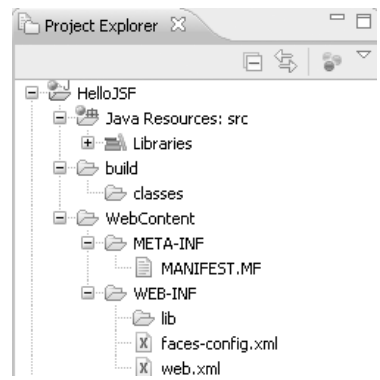
Configuration du projet dynamique avec JSF



L'arborescence « typique » d'un projet Web/JSF est générée comme illustré à la figure 8.5.

Figure 8.5

Arborescence du projet Web/JSF



7. Créez votre première page JSF en sélectionnant l'assistant de création de page JSP puis en donnant à la page le nom hello.jsp. Cliquez sur Finish.
8. Ajoutez les taglibs JSF dans la page créée en profitant des fonctionnalités de complétion automatique de code offertes par Web Tools (voir figure 8.6).

Figure 8.6

Construction de la page JSF

Les deux lignes (requises) contenant la balise @taglib sont des directives qui font respectivement référence à l'emplacement des balises JSF qui définissent les éléments de base de JSF ainsi qu'aux éléments HTML de la page.

9. Ajoutez juste au-dessus de la balise <f:view> les lignes suivantes :

```
<h:outputLabel> rendered="true" value="Nom" </h:outputLabel>
  <h:inputText tabindex="#{loginBean.nom}"></h:inputText>
</f:view>
```

Ces lignes vont permettre la construction d'un formulaire destiné à transmettre la valeur saisie à un backing bean (loginBean) d'affichage.

10. Créez la classe com.chap08.jsf.hello suivante à l'aide de l'assistant de création de classe Eclipse :

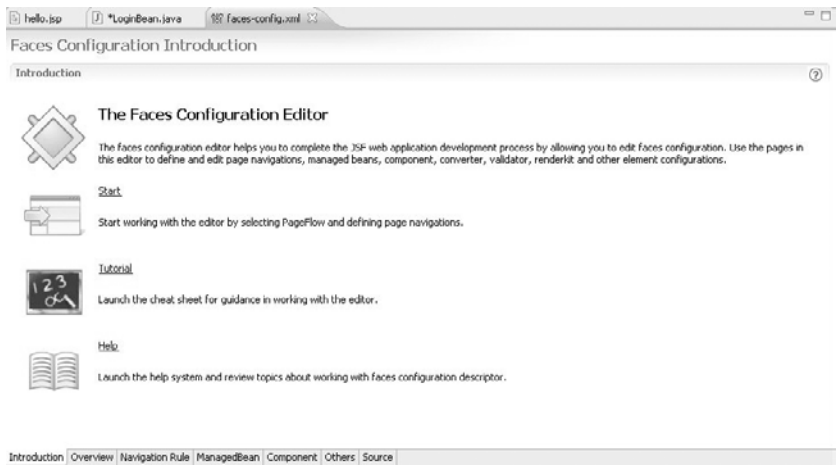
```
package com.chap08.jsf.hello;

public class LoginBean {
String nom;

public LoginBean () {
    super();
}

public String getNom() {
    return nom;
}
```

```
public void setNom(String nom) {
    this.nom = nom;
}
}
```

11. Dans l'explorateur de projet, double-cliquez sur le fichier de configuration JSF. L'éditeur JSF intégré s'affiche comme illustré à la figure 8.7.

Figure 8.7

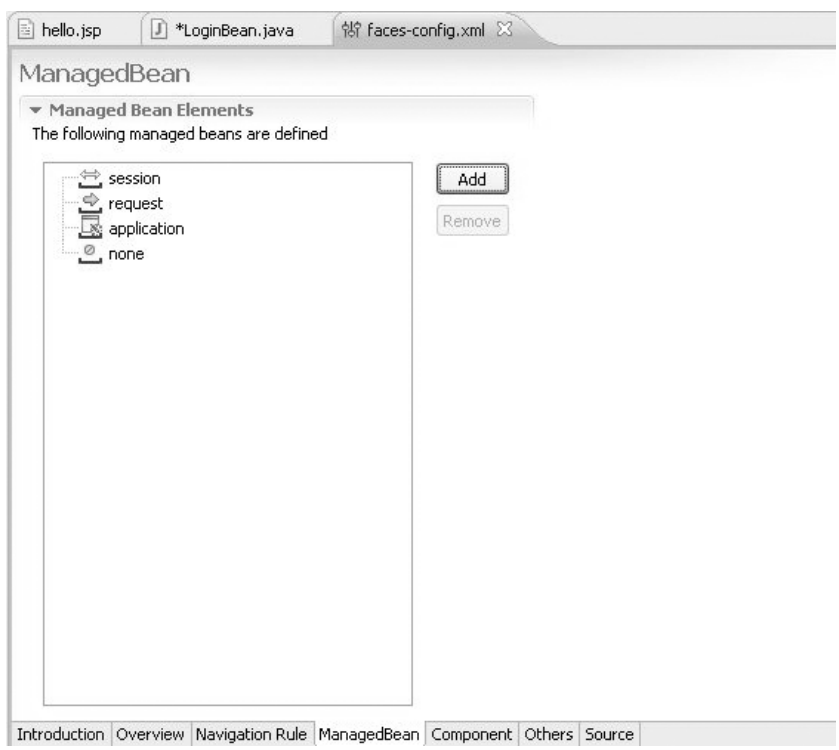
Assistant Faces Configuration Editor



12. Sélectionnez l'onglet ManagedBean, puis cliquez sur Add pour lancer l'assistant de création ManagedBean illustré à la figure 8.8.

Figure 8.8

Configuration du ManagedBean (1/2)



13. Sélectionnez la classe com.chap08.jsf.hello.loginBean et cliquez sur Next.
14. La boîte de dialogue de dialogue de configuration propose de définir le scope du bean (les scopes possibles sont application, session, request ou none). Sélectionnez session, et cliquez sur Finish.
15. L'écran suivant propose l'état de la configuration générale du bean ainsi enregistré. Cliquez sur Ctrl+S pour sauvegarder la configuration (voir figure 8.9).

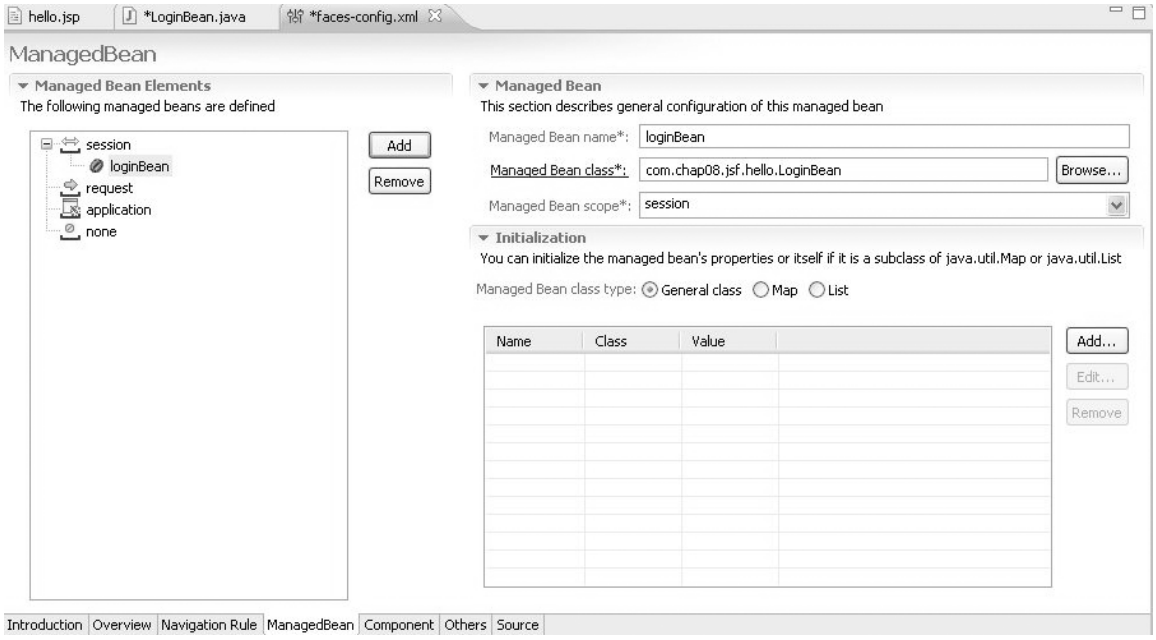


Figure 8.9

Configuration du ManagedBean (2/2)

La référence de ce bean est définie dans la configuration du fichier faces-config.xml avec l'attribut session correctement positionné :

```
<managed-bean>
    <managed-bean-name>
    loginBean</managed-bean-name>
    <managed-bean-class>
    com.chap08.jsf.hello.LoginBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

16. Basculez sur la page hello.jsp.
17. Faites un clic droit sur la page, et sélectionnez Validate dans le menu contextuel.
18. Ajoutez la balise JSF `<h :inputMessage value= »#{} »></h :inputMessage>`, comme illustré à la figure 8.10.
19. Sélectionnez loginBean dans la liste déroulante.
20. Entrez le nom d'une propriété, par exemple message, et sauvegardez la configuration.

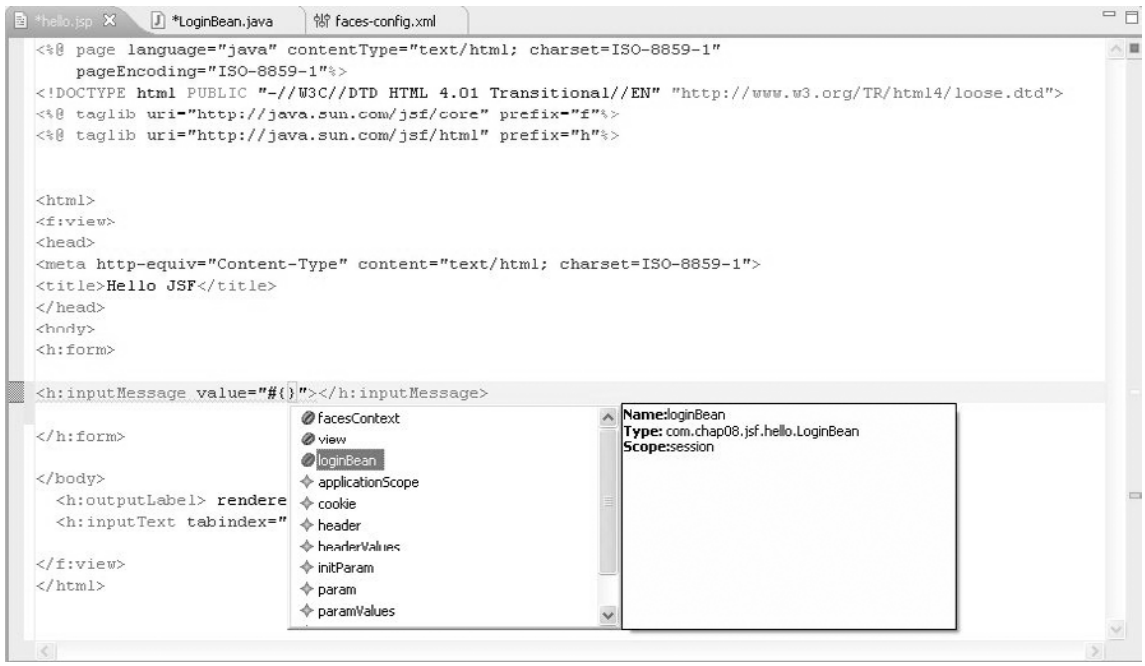


Figure 8.10

Configuration du bean dans la page JSF

21. Ajoutez la référence à une ressource de type message *via* le code suivant, juste après la balise de fin </title> :

```
<f :loadBundle basemane= "com.chap08.jsf.hello.message" var="msg"/>
```

22. Ajoutez la référence au fichier properties correspondant en créant, dans le package com.chap08.jsf.hello sous le dossier Java Resources:src, un fichier messages.properties contenant les lignes suivantes (destinées à externaliser les libellés utilisés par les pages JSP, ce qui est une bonne habitude à prendre) :

```
nom=Nom
login=Login
welcome=Bienvenue sur JSF
fin= !
```

Lors de la compilation, ce fichier sera copié dans le répertoire classes du projet et découvert par le runtime Eclipse lors de l'exécution.

23. Complétez votre page hello.jsp pour qu'elle intègre au final les lignes d'affichage de la page avec les champs Login ainsi que la référence aux libellés saisis :

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
➥"http://www.w3.org/TR/html4/loose.dtd">
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<html>
```

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Hello JSF</title>
<f:loadBundle basename="com.chap08.jsf.hello.messages.messages" var="msg"/>
</head>
<body>

<f:view>
  <h:form>

   <h:messages layout="table"></h:messages>
   <h:panelGrid columns="2">
      <h:outputLabel rendered="true" value="#{msg.nom}"></h:outputLabel>
      <h:inputText value="#{loginBean.nom}" tabindex="0"></h:inputText>
  </h:panelGrid>

  <h:commandButton action="hello" value="#{msg.login}"></h:commandButton>
</h:form>

</f:view>
</body>
</html>
```

Comme vous pouvez le voir dans ce dernier extrait, la balise outputLabel affiche un message « Nom » à partir du fichier properties créé lors des étapes précédentes.

La ligne contenant la balise commandButton permet de soumettre le formulaire HTML en lui passant l'attribut action positionné à « hello », qui va « matcher » ce mot-clé dans les sous-balises de la balise principale navigation-outcome.

Configuration de la cinématique des pages JSF avec l'éditeur de configuration Web Tools

Vous allez commencer par créer une deuxième page JSP destinée à afficher un message de bienvenue ainsi qu'à récupérer dans la session le login de la personne. Vous utiliserez pour cela l'éditeur JSF Faces Configuration Resource Editor, qui vous aidera grandement dans la gestion de la cinématique des pages.

1. À l'aide de l'assistant de création de page JSF, créez la page welcome.jsp suivante :

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
➥"http://www.w3.org/TR/html4/loose.dtd">
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>

<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Bienvenue</title>
<f:loadBundle basename="com.chap08.jsf.hello.messages" var="msg"/>
```

```
</head>
<body>
<f:view>

 < h3>
    <h:outputLabel value="#{msg.welcome}#{loginBean.nom}"></h:outputLabel>
     <h:outputLabel value="#msg.fin}" />
  </h3>

</f:view>

</body>
</html>
```

2. Créez une troisième page JSP, la page index.jsp suivante associée à la balise welcome-file du fichier de configuration web.xml afin de permettre de router l'application vers la page hello.jsp :

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
  <jsp:forward page="/hello.jsf" />
</body>
</html>
```

Vous allez maintenant configurer la cinématique des pages JSF avec l'éditeur de configuration Web Tools.

3. Double-cliquez sur le fichier faces-config.xml pour lancer l'éditeur de configuration des pages JSF et sélectionnez l'onglet Navigation Rule.
4. Cliquez sur le menu Palette pour faire apparaître l'éditeur graphique JSF Web Tools illustré à la figure 8.11.

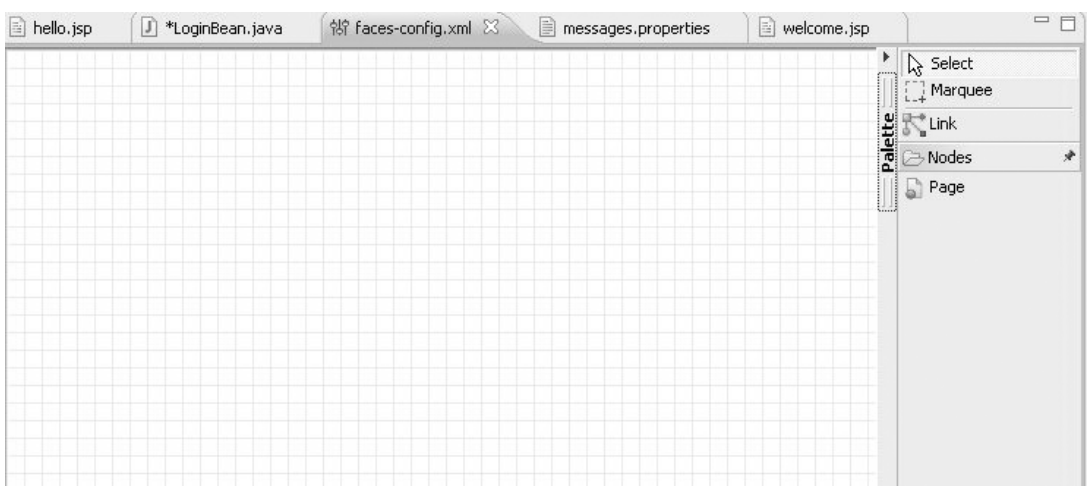


Figure 8.11

Configuration de la cinématique JSF avec l'assistant Web Tools (1/2)

5. Faites un glisser-dépose des pages hello.jsp et welcome.jsp dans l'éditeur JSF, et reliez-les à l'aide de l'option Link de la palette.
6. Sélectionnez la ligne et configurez-la *via* l'onglet Properties en renseignant les propriétés suivantes (voir figure 8.12) :
   – From Outcome : hello.
   – Redirect : false.

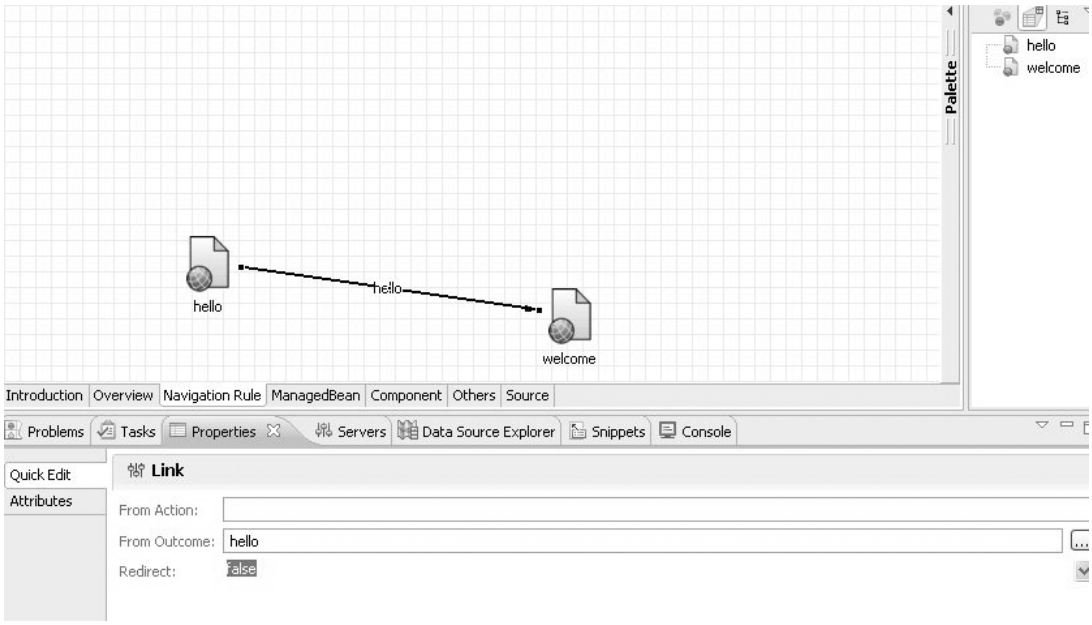


Figure 8.12

Configuration de la cinématique JSF avec l'assistant Web Tools (2/2)

Vous pouvez voir l'impact de cette définition de la cinématique du projet helloJSF dans le fichier de configuration JSFfaces-config.xml :

```
<navigation-rule>
    <display-name>hello</display-name>
    <from-view-id>/hello.jsp</from-view-id>
    <navigation-case>
        <from-outcome>hello</from-outcome>
        <to-view-id>/welcome.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```

Ce bloc XML définit les règles de navigation au sein des pages du projet. La balise <from-view-id> définit la page initiale avant l'invocation de la requête vers le bean managé ; dans votre cas, il s'agit de la page hello.jsp.

Le listener associé au bean ne dispose que d'une option par l'intermédiaire de la chaîne hello retournée : l'invocation de la page welcome.jsp définie dans le fichier de configuration JSF ci-dessus.

Déploiement de la mini-application helloword sur JBoss

Il vous reste un dernier ajustement à apporter à votre fichier descripteur de déploiement web.xml généré :

1. Remplacez le contenu de la ligne contenant la balise <url-pattern> :

```
<url-pattern>/faces/*</url-pattern>
```

Par le contenu suivant :

```
<url-pattern>*.jsf </url-pattern>
```

`*.jsf` est le modèle d'URL utilisé au lieu de `faces/*` pour signaler que la page redirigée dans la page index.jsp doit être prise en charge par la servlet JSF au sein du conteneur Web de JBoss.

2. Faites une publication complète de votre projet sur le serveur *via* l'option du menu contextuel du serveur Publish, et lancez votre serveur. Cela a pour effet de vous placer automatiquement sur la page hello.jsp, comme l'illustre la figure 8.13.

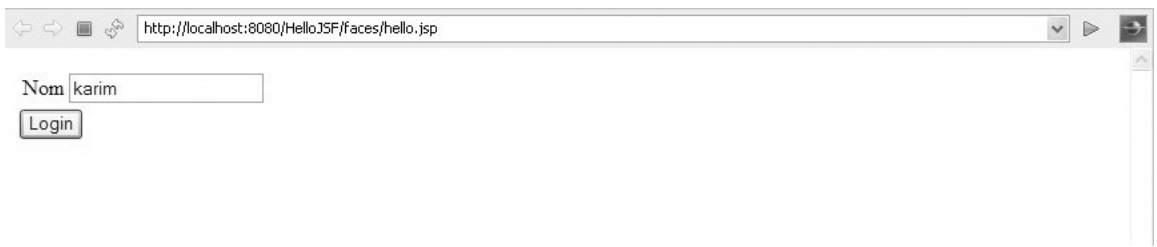


Figure 8.13

Déploiement de helloword avec l'assistant de déploiement Web Tools (1/2)

3. Cliquez sur le bouton Login. Vous devez voir invoquée la page de bienvenue illustrée à la figure 8.14.

Figure 8.14

Déploiement de helloword avec l'assistant de déploiement Web Tools (2/2)

Cette section construite autour d'un exemple simple vous a permis d'apprécier la richesse de la panoplie Web Tools.

JBoss Seam

Après un rappel de la genèse du projet JBoss Seam et de ses caractéristiques fondamentales, vous verrez comment configurer Seam au sein de vos projets JEE fondés sur l'IDE Eclipse Europa et Web Tools. Les bibliothèques associées à la version du framework utilisée ici figurent sur la page Web dédiée à l'ouvrage (répertoire lib).

Conçu par les équipes R&D des laboratoires JBoss et par l'un des concepteurs du framework ORM Hibernate, Gavin King, Seam est un framework pour les applications Web 2.0 qui permet d'unifier et d'intégrer les technologies suivantes :

- SOA (Service Oriented Architecture), telles qu'AJAX (Asynchronous JavaScript and XML) ;
- JSF 1.2 (avec Seam 2.0) ;
- EJB 3.0 ;
- portlets Java ;
- gestion des processus métier (jBPM) et des workflows.

Conçu avant tout pour éliminer la complexité de l'architecture et de l'interface de programmation (API), Seam permet aux développeurs d'associer des applications Web complexes à des objets Java simples (POJO) annotés, des composants graphiques Web et une petite quantité de code XML(voir plus loin la définition des objets POJO).

Pour ce faire, Seam applique le modèle de programmation par annotations avec configuration par exception d'EJB3 à toute la pile d'applications Web. Cela a pour effet de combler l'espace entre EJB 3.0 et JSF sur la plate-forme JEE5 (Java Enterprise Edition 5.0). Cette nouvelle version est un modèle d'unification étroitement intégrée, qui active les applications avec état, sans état, transactionnelles ou fondées sur des processus, telles que les workflows et les flux de pages.

La simplicité de Seam facilite l'intégration avec les ESB (Enterprise Service Bus) et bientôt JBI (Java Business Integration), une norme édictée dans la JSR-208 définissant une approche orientée composant permettant le routage des composants *via* des messages.

En résumé, JBoss Seam est un framework clé en main permettant d'affronter les nouveaux standards du Web 2.0 d'aujourd'hui et de demain.

Caractéristiques du framework Seam

Seam présente de nombreux avantages et un certain nombre de fonctionnalités innovantes, parmi lesquelles :

- **Développement fondé sur les POJO et le pattern DI (Dependency Injection).** Seam est un conteneur léger (lightweight framework) qui promeut l'usage des objets POJO (Plain Old Java Objects) comme composants service de l'infrastructure sous-jacente. Il n'est pas nécessaire d'utiliser des interfaces ou des classes abstraites à hériter pour masquer les composants de votre application. L'approche Seam se fonde sur le design pattern DI, ou injection de dépendances, qui permet de déléguer au conteneur toutes les tâches fastidieuses de gestion du cycle de vie des composants et d'injecter des services ou d'autres objets au sein d'objets POJO grâce aux annotations : vous créez simplement vos objets POJO pour modéliser votre modèle métier ou implémenter le processus métier de votre application en utilisant les données.

- **Couche de traitement à distance fondée sur AJAX.** Seam Remoting permet aux beans session EJB3 d'être invoqués directement par le client de navigation Web *via* la technologie AJAX. Les beans session apparaissent au développeur comme de simples objets JavaScript et masquent la complexité de la sérialisation en XML et de l'API XMLHttpRequest.
- **Gestion déclarative de l'état des applications.** Jusqu'à présent, les applications Web J2EE implémentaient la gestion des états manuellement, une approche générant bogues et dysfonctionnements de la mémoire lorsque les applications ne parvenaient pas à nettoyer les attributs de session. Seam supprime ces désagréments. La gestion déclarative de l'état s'appuie sur le modèle de contexte enrichi défini par Seam.
- **Modèle de programmation contextuel.** Auparavant, une session HTTP était le seul moyen de gérer les états successifs d'une application Web. Seam fournit désormais plusieurs contextes, avec états de granularité différents, depuis le niveau conversation jusqu'au niveau processus métier, libérant ainsi les développeurs des contraintes des sessions HTTP. Seam ajoute trois contextes aux nombreux déjà existants : Conversation, Process et Application. Le premier permet de définir et de gérer simplement les conversations entre requêtes ; le deuxième gère les processus afin de définir des tâches et conversations entre eux de manière similaire à JBoss jBPM ; le troisième est un contexte disponible durant toute l'application.
- **Prise en charge des applications à base de processus.** Seam intègre en toute transparence la gestion des processus métier *via* JBoss jBPM, simplifiant ainsi radicalement l'implémentation des workflows complexes et des applications de flux de pages.
- **Intégration de portails.** Seam prend en charge les portails compatibles JSR-168, tels que JBoss Portal.

Pour toutes ces raisons, Seam représente un excellent choix pour le développement d'applications Web utilisant les technologies JSF et EJB 3.0, auxquelles il apporte une parfaite synergie. Les EJB stateless et stateful deviennent ainsi des composants de gestion événementielle complémentaires de JSF. De plus, les composants JSF peuvent lier leur état aux beans entité. La généralisation de techniques telles que l'inversion de contrôle (ou IOC) et les contextes Seam libère le développeur de la gestion manuelle des relations et du cycle de vie au sein des différents composants de l'application.

La figure 8.15 illustre l'architecture JEE finale en logique 3-tiers avec le framework JBoss Seam.

Seam permet « d'envelopper » les objets POJO, ou classes Java « simples », d'annotations Java liées à la couche présentation grâce à des intercepteurs Seam. Cela permet au développeur de se concentrer sur la logique métier, la présentation et la couche de données sans se soucier de la façon de les lier ensemble et de les faire interagir.

Précisons que le framework Seam fonctionne avec n'importe quel serveur d'applications supportant les EJB 3.0 (à l'heure ou nous écrivons ces lignes JBoss 4.2.0 GA avec le support EJB3 ou le SA compatible JEE5 Glassfish de Sun). Vous pouvez même utiliser Seam dans un conteneur de servlets comme Tomcat.

La figure 8.16 illustre les composants fondamentaux de l'architecture Seam : JSF pour la partie présentation, Hibernate (ou JDBC) pour la persistance et les composants JavaBeans pour la logique applicative.

Figure 8.15

Mise en œuvre de l'architecture 3-tiers pour les application JEE avec Seam

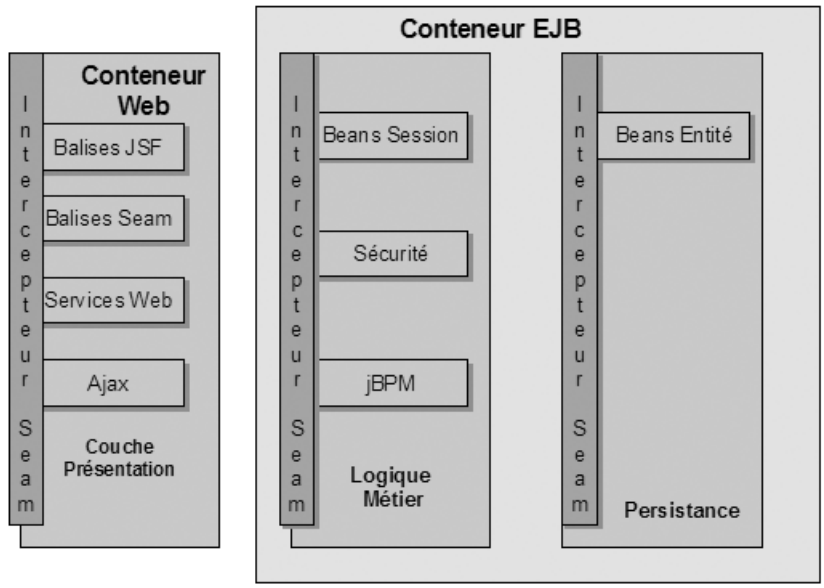
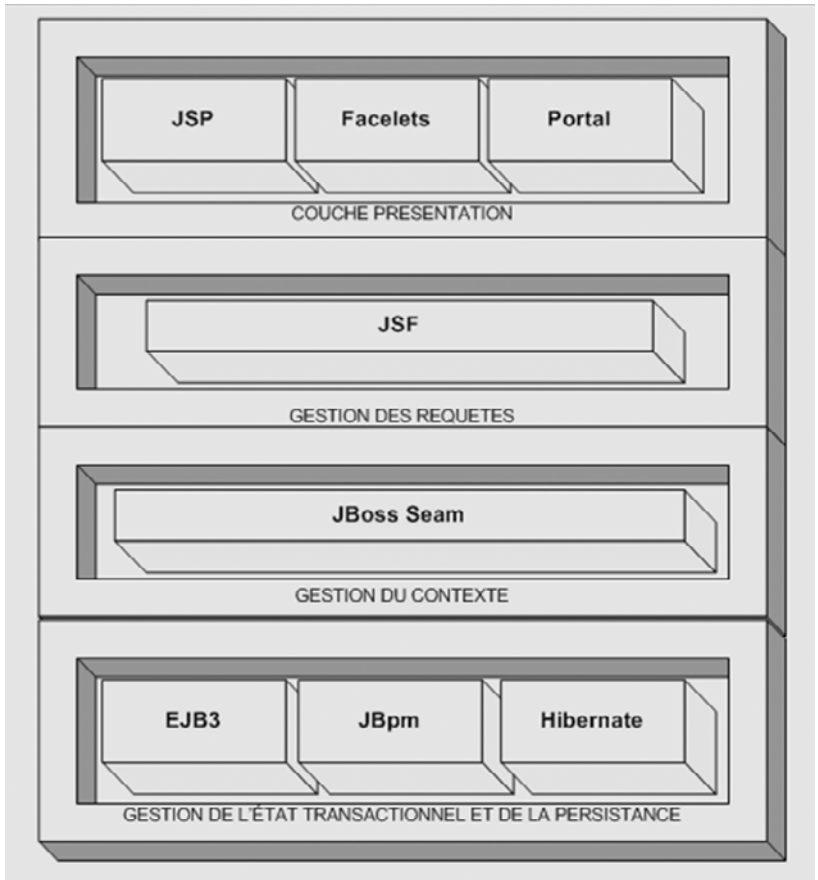


Figure 8.16

Composants de l'architecture Seam



En résumé, construire une application avec Seam est un processus relativement simple, qui ne nécessite que le codage des composants suivants :

- Composants métier, qui représentent le modèle métier. Ces objets peuvent être des beans entité utilisant l'API de persistance JPA (Java Persistence API) ou des objets POJO avec Hibernate, qui sont automatiquement mappés à des tables de la base de données.
- Pages Web JSF pour la présentation. Les champs de la page sont mappés au modèle de données *via* le langage JSF EL (Expression Lanuguage).
- Beans session EJB3 ou objets POJO annotés agissant à la façon d'un gestionnaire d'événements pour les pages Web JSF et mettant à jour le modèle en se fondant sur les données saisies par l'utilisateur.

POJO

POJO est l'acronyme de Plain Old Java Objects et fait référence à la simplicité d'utilisation des objets Java en comparaison de la lourdeur des composants EJB. Par définition, un objet POJO (que l'on peut traduire par objet Java), n'implémente pas d'interface spécifique à un framework particulier.

Nous aurons l'occasion de revenir sur ces concepts et de les mettre en œuvre au cours des chapitres suivants, qui traitent spécifiquement du développement EJB3 avec Seam et JBoss.

Mise en œuvre de Seam

Cette section s'efforce de préciser les étapes nécessaires à la mise en œuvre du framework Seam au sein de vos projets.

Vous devez disposer d'une version Java 5 ou ultérieure afin de pouvoir utiliser les annotations, ainsi que d'une implémentation de JSF pour la partie présentation. Cela suppose l'utilisation d'un serveur d'applications qui les supporte, comme c'est le cas des serveurs JBoss AS 4.2.x et 5.x *(bêta),* qui disposent d'une version embarquant l'implémentation de référence des spécifications JSF 1.2.

À la place d'EJB3, il est aussi possible d'utiliser le framework objet-relationnel Hibernate comme technologie de persistance.

La distribution de Seam utilisée dans cet ouvrage supportant la version du serveur JBoss 4.2 est la release 2.0, disponible sur la page Web dédiée à l'ouvrage. Nous supposons que vous l'avez téléchargée et décompressée dans un répertoire (par défaut jboss-seam-2.0.0.BETA1). À l'heure ou nous mettons sous presse, la release Seam 2.0.0 CR1 dite « stable » est disponible. Elle offre, entre autres nouvelles fonctions, une intégration poussé avec Hibernate et le support de JBoss EL (Expression Language) Enhancement, une extension du language d'expression unifié de JEE5 (voir le site officiel du projet pour plus de précisions http://labs.jboss.com/jbosseam/download/index.html).

Ant 1.6+ est également requis, tous les scripts générés par Seam s'appuyant sur cet outil de génération.

Rappels et précisions sur l'utilisation des facelets Seam

Le framework Seam offre une technologie de substitution à JSP avec le support du moteur de template Facelets. Les facelets offrent des avantages indéniables par rapport à JSP, notamment le fait de ne pas dépendre d'une balise JSP. Vous pouvez ainsi commencer à employer les nouveaux dispositifs de JSF 1.2 sans attendre la disponibilité d'une balise de JSP.

Fonctionnant avec JSF 1.1 et 1.2, les facelets facilitent la conception des pages en même temps que leur lisibilité, tout en réduisant la quantité de code nécessaire pour intégrer des composants dans la vue et en n'exigeant pas de balises Web.

Très légère, la technologie Facelet améliore les performances de JSF de 30 à 50 % en « bypassant » le moteur de JSP et en utilisant les balises xHTML directement comme composants « vue ».

Enfin, et c'est important pour le support de la technologie JSF, Facelets fournit un framework de templating pour JSF utilisant le modèle Seam d'injection de dépendances pour assembler les pages, au lieu de le faire manuellement, portion de page par portion de page à afficher (haut de page ou header, bas de page ou footer et barre de menus).

Seam pour des applications Web POJO

Seam a été conçu à l'origine pour se placer au-dessus de JEE5 afin de servir de pont entre les technologies JSF et EJB3. Toutefois, Seam n'oblige pas le développeur à fonder son architecture sur EJB3. Tout objet POJO avec une annotation @Name peut être managé par les services du conteneur. Rappelons qu'un objet POJO n'implémente pas d'interface spécifique d'un framework, à la différence d'un composant EJB.

Il est dès lors possible d'utiliser les transactions ou la persistance objet et de sécuriser votre application par le biais des frameworks Spring, Hibernate, etc., tout en ne faisant appel qu'à des objets Java classiques. L'astuce réside dans l'injection d'une dépendance au sein des paramètres du constructeur d'un objet (en particulier par l'utilisation des annotations @in et @out, par exemple, qui forment le cœur du modèle de programmation Seam).

Par exemple, vous pouvez développer l'objet POJO manager suivant au lieu d'un bean session EJB3 :

```
@Name("manager")
public class ManagerPojo {

  @In (required=false) @Out (required=false)
  private Employe employe;

  // @PersistenceContext (type=EXTENDED)
  @In (create=true)
private EntityManager em;
```

L'utilisation d'objets POJO pour remplacer les composants EJB3 ne présente toutefois pas que des avantages. Certes, les objets POJO sont beaucoup plus faciles à développer puisqu'ils ne requièrent pas d'annotations EJB3 ni d'interfaces spécifiques. De plus, si tous vos composants métier sont des objets POJO Seam, vous pouvez exécuter votre

application en dehors du conteneur du serveur d'applications EJB3 (tout conteneur Web compatible, comme Tomcat, est suffisant).

Toutefois, un objet POJO dispose de moins de fonctionnalités qu'un composant EJB3, car il ne peut pas compter sur les services du conteneur EJB3.

Citons notamment les limitations suivantes :

- L'injection @PersistenceContext ne fonctionne pas avec les objets POJO alors qu'elle est utilisée de manière transparente par le conteneur EJB3. Pour obtenir un objet EntityManager avec Seam POJO, il est nécessaire d'initialiser EntityManager dans le fichier de configuration Seam et d'utiliser l'annotation @in pour l'injecter dans l'objet POJO. Nous verrons ce point en détail dans les chapitres suivants.
- Il n'existe pas de support pour le niveau déclaratif des méthodes transactionnelles en POJO, qui reste du ressort du développeur, contrairement aux beans EJB3 (utilisation de l'annotation @TransactionAttribute). Il est donc nécessaire de configurer Seam pour démarrer une transaction base de données lorsque la requête est reçue jusqu'à l'affichage de la réponse.
- Pas de composants POJO de type MDB (message driven bean).
- Pas de support pour les méthodes asynchrones (@Asynchronous).
- Pas de support de la sécurité gérée par le conteneur.

En résumé, faire le choix de POJO complexifie la configuration à mettre en œuvre et réduit l'accès aux services offerts nativement par le conteneur.

Configuration de Seam avec les composants POJO

Vous allez construire une application POJO en utilisant les API de la spécification JPA (Java Persistence API), fondement de la norme EJB3, qui permet de gérer le cycle de vie d'un objet persistant. Précisons que cette API peut être utilisée en dehors d'un serveur d'applications, par exemple avec Tomcat en utilisant simplement Java SE.

JPA est un composant important de la spécification JEE5, et EJB3 en particulier, et constitue d'ailleurs un des apports majeurs de cette spécification qui va dans le sens d'une plus grande simplicité de mise en œuvre des EJB.

Votre application utilisera Hibernate JPA comme fournisseur JPA.

L'application permet de gérer une liste d'employés selon l'approche POJO en s'appuyant sur les API JPA. Comme vous le verrez, lorsque l'utilisateur clique sur le bouton du formulaire d'ajout d'employé, Seam crée un composant managé `employe`. Il invoque alors la méthode `updateEmploye()` sur le composant Seam managé.

Configuration du projet POJO

Pour déployer une application en dehors d'un conteneur EJB3, vous devez configurer Seam de manière qu'il puisse prendre en charge les services normalement gérés par le conteneur. Vous pouvez également utiliser le générateur Seam (seam-gen), qui permet, en sélectionnant un type de projet war, de construire la configuration d'un projet de type POJO au sein du serveur JBoss, ou simplement réutiliser un modèle existant en utilisant un modèle de fichier de construction build.xml comme celui que nous présentons ci-dessous.

Une structure de projet Seam/POJO avec JPA se présente selon la structure monApp-SeamPOJO suivante :

```
| + src
      | + Sources Java
|+  view
    |+  pages web (.xhtml), css et images
|+ resources
   |+ WEB-INF
        |+ web.xml
        | + components.xml
        | + faces-config.xml
        | + navigation.xml
        | + pages.xml
        |+ jboss-web.xml
    |+ META-INF
        |+ persistence.xml
    |+  seam.properties
|+ lib (libraires jar SEAM)
    |+ App bibliothèques spécifiques
|+ test
|+ build.xml
|+ build.properties
```

Cette structure projet Seam/POJO nécessite la personnalisation des composants suivants :

- Ajout des composants Seam et des classes nécessaires à l'application dans le répertoire src.
- Ajout des pages Web, des images et des autres ressources dans le répertoire view.
- Ajout des bibliothèques requises sous le répertoire lib (support des Facelets avec la bibliothèque jsf-facelets.jar et du framework Seam comme jboss-seam-ui.jar et jboss-seam-debug.jar).
- Ajout de bibliothèques spécifiques sous le répertoire lib pour le support, par exemple, de composants AJAX. Cela implique de modifier en conséquence le fichier build.xml si vous souhaitez packager vos jar en dehors du fichier app.jar (voir la tâche Ant pojar correspondante).
- Modification du fichier navigation.xml sous resources/WEB-INF pour définir les règles de navigation entre les pages.
- Modification du fichier resources/WEB-INF/pages.xml pour inclure les paramètres des pages de type restful.
- Modification du fichier resources/META-INF/persistence.xml pour spécifier les options personnalisées de persistance (type de framework JPA ou hibernate par exemple).
- Modification des paramètres de configuration de l'application (changement du nom du projet dans le fichier build.xml et modification du fichier resources/WEB-INF/jboss-web.xml pour le positionnement du contexte root de l'application).

Le modèle de fichier build.xml de votre application de gestion des employés avec support JPA est fourni pour examen ci-dessous (il s'appuie sur le fichier de propriété build.properties contenant la référence au serveur d'applications JBoss) :

```
<project name="MonPOJO" default="deploy" basedir=".">

 <description>Mon projet POJO</description>
 <property name="projname" value="jpa" />

 <property file="build.properties"/>
 <property name="jboss.deploy"
   location="${jboss.home}/server/default/deploy"/>

 <property name="lib" location="../lib" />
 <property name="testlib" location="../lib/embeddedejb3" />
 <property name="applib" location="lib" />
 <path id="lib.classpath">
   <fileset dir="${lib}" includes="*.jar"/>
   <fileset dir="${testlib}" includes="*.jar"/>
   <fileset dir="${applib}" includes="*.jar"/>
 </path>

 <property name="resources" location="resources" />

 <property name="src" location="src" />
 <property name="test" location="test" />
 <property name="view" location="view" />

 <property name="build.classes" location="build/classes" />
 <property name="build.jars" location="build/jars" />
 <property name="build.test" location="build/test" />
 <property name="build.testout" location="build/testout" />

 <target name="clean">
   <delete dir="build"/>
 </target>

 <target name="main" depends="compile,pojojar,war"/>

 <target name="compile">
   <mkdir dir="${build.classes}"/>
   <javac destdir="${build.classes}"
          classpathref="lib.classpath"
          debug="true">
     <src path="${src}"/>
   </javac>
 </target>

 <target name="test" depends="compile">

   <taskdef resource="testngtasks"
            classpathref="lib.classpath"/>

   <mkdir dir="${build.test}"/>

   <javac destdir="${build.test}"
          debug="true">
     <classpath>
       <path refid="lib.classpath"/>
       <pathelement location="${build.classes}"/>
     </classpath>
```

```
    <src path="${test}"/>
  </javac>

  <copy todir="${build.test}">
    <fileset dir="${build.classes}" includes="**/*.*"/>
    <fileset dir="${resources}" includes="**/*.*"/>
    <fileset dir="${testlib}/conf" includes="*.*"/>
  </copy>
  <!-- Overwrite the WEB-INF/components.xml -->
  <copy todir="${build.test}/WEB-INF" overwrite="true">
    <fileset dir="${test}" includes="components.xml"/>
  </copy>

  <testng outputdir="${build.testout}">
    <classpath refid="lib.classpath"/>
    <classpath path="${build.test}"/>
    <xmlfileset dir="${test}" includes="testng.xml"/>
  </testng>

</target>

<target name="pojojar" depends="compile">
  <mkdir dir="${build.jars}"/>

  <jar destfile="${build.jars}/app.jar">
    <fileset dir="${build.classes}">
      <include name="**/*.class"/>
    </fileset>
    <fileset dir="${resources}">
      <include name="seam.properties" />
    </fileset>
    <fileset dir="${applib}">
      <include name="*.jar" />
    </fileset>
    <metainf dir="${resources}/META-INF">
      <include name="persistence.xml" />
    </metainf>
  </jar>
</target>

<target name="war" depends="pojojar">
  <mkdir dir="${build.jars}"/>

  <war destfile="${build.jars}/${projname}.war"
       webxml="${resources}/WEB-INF/web.xml">
    <webinf dir="${resources}/WEB-INF">
      <include name="faces-config.xml" />
      <include name="components.xml" />
      <include name="navigation.xml" />
      <include name="pages.xml" />
      <include name="jboss-web.xml" />
    </webinf>
    <lib dir="${lib}">
      <include name="jboss-seam.jar" />
      <include name="jboss-seam-ui.jar" />
      <include name="jboss-seam-debug.jar" />
      <include name="jsf-facelets.jar" />
```

```
      <include name="hibernate3.jar" />
      <include name="hibernate-entitymanager.jar" />
      <include name="hibernate-annotations.jar" />
      <include name="ejb3-persistence.jar" />
    </lib>
    <lib dir="${build.jars}"
         includes="app.jar"/>
    <fileset dir="${view}"/>
  </war>
</target>

<target name="war421" depends="pojojar">
  <mkdir dir="${build.jars}"/>

  <war destfile="${build.jars}/${projname}.war"
       webxml="${resources}/WEB-INF/web.xml">
    <webinf dir="${resources}/WEB-INF">
      <include name="faces-config.xml" />
      <include name="components.xml" />
      <include name="navigation.xml" />
      <include name="pages.xml" />
      <include name="jboss-web.xml" />
    </webinf>
    <lib dir="${lib}">
      <include name="jboss-seam.jar" />
      <include name="jboss-seam-ui.jar" />
      <include name="jboss-seam-debug.jar" />
      <include name="jsf-facelets.jar" />
      <include name="jsf-api.jar" />
      <include name="jsf-impl.jar" />
      <include name="el-api.jar" />
      <include name="el-ri.jar" />
      <include name="hibernate3.jar" />
      <include name="hibernate-entitymanager.jar" />
      <include name="hibernate-annotations.jar" />
      <include name="ejb3-persistence.jar" />
    </lib>
    <lib dir="${build.jars}"
         includes="app.jar"/>
    <fileset dir="${view}"/>
  </war>
</target>

<target name="deploy">
  <copy file="${build.jars}/${projname}.war"
    todir="${jboss.deploy}"/>
</target>

<target name="undeploy">
  <delete
      file="${jboss.deploy}/${projname}.war"/>
</target>
</project>
```

La structure projet complète est disponible sur la page Web dédiée à l'ouvrage (projet MonPOJO du chapitre 08). Il vous suffit de l'importer dans Eclipse et de la lancer *via* l'outil Ant intégré dans l'IDE.

Mise en œuvre du projet Seam POJO

Comme indiqué précédemment, la différence principale entre un projet à base d'EJB3 et un projet POJO réside dans la manière dont l'objet manager est utilisé.

Conception du modèle métier

La mise en œuvre d'un projet Seam nécessite la conception des objets métier et de la couche de persistance associée. Selon l'approche POJO, il s'agit d'identifier les objets métier propres à l'application, soit ici l'entité `Employe` décrite dans l'extrait ci-dessous :

```
@Entity
@name ("employe")
@Table(name="employe")
public class Employe implements Serializable {

  private long id;
  private String name;
  private long salaire;
  private String adresse;
  private String comment;
@Id @GenerateValue
  public Employe () {
    nomprenom ="";
    salaire = 0;
    adresse ="";
    comment ="";
}
```

Examinez également le détail du code de la classe ManagerPojo.java, qui permet de faire le lien entre le bean métier `Employe` et la modélisation des processus métier (gestion des employés).

Cette classe présente un certain nombre de concepts Seam importants en relation avec les annotations @in, @DataModel, @DataModelSelection et @Factory, comme le montre l'extrait suivant :

```
@Name("manager")
public class ManagerPojo {

  @In (required=false) @Out (required=false)
  private Employe employe;

  // @PersistenceContext (type=EXTENDED)
  @In (create=true)
  private EntityManager em;

  // @RequestParameter
  Int pid;

  public String updateEmploye () {
    em.persist (employe);
    return "employes";
  }
```

```
@DataModel
private List <Employe> employes;

@DataModelSelection
private Employe selectedEmploye;

@Factory("employes")
public void findEmployes () {
  employes = em.createQuery("select emp from Employe emp")
                                .getResultList();
}

public void setPid (Long pid) {
  this.pid = pid;

  if (pid != null) {
    employe = (Employe) em.find(Employe.class, pid);
  } else {
    employe = new Employe ();
  }
}

public Int getPid () {
  return pid;
}

public String delete () {
  Employe toDelete = em.merge (selectedEmploye);
  em.remove( toDelete );
  findEmployes ();
  return null;
}

public String update () {
  return "employes";
}
```

Vous devez injecter un objet EntityManager en utilisant l'annotation Seam @In. @DataModel vous permet d'utiliser une liste d'éléments sélectionnables qui transforment le composant `employes` en objet DataModel JSF.

Notez dans l'extrait de la classe ManagerPojo l'utilisation de la méthode `persist()` pour effectuer la persistance de l'entité `employe`. Lors du retour de la méthode `persist()`, l'objet `employe` devient un objet entité « managé » dans le contexte de l'objet EntityManager.

L'annotation @DataModelSelection demande à Seam quel objet doit être injecté à partir de la liste et qui sera sélectionné dans la couche de présentation. Cet objet `selectedEmploye` peut être ensuite utilisé dans la méthode de suppression `delete()` et être appelé à partir de la page de suppression.

L'annotation @Factory entre ensuite en action pour spécifier le type d'initialisation du bean `listEmployes` et répondre aux besoins de la page JSF pour l'affichage des employés (page employes.xhtml).

Le processus Seam d'édition et de mise à jour des employés de la base se déroule de la manière suivante :

1. L'utilisateur déclenche une édition de l'employé en cours *via* l'invocation de l'URL `employe.seam ?pid=1 par exemple`.
2. La page employe.xhtml est traitée, et Seam instancie le composant `listEmployes` pour afficher les données dans la page.
3. Seam injecte la valeur du pid dans l'objet ManagerPojo et appelle la méthode fabrique ManagerPojo.findEmployes pour construire et « éjecter » le composant `listEmployes`.
4. La page est affichée avec le composant `employes`.

Le tiers présentation JSF

Vous pouvez passer à la gestion de la partie présentation, qui va être chargée de l'affichage de la liste des employés (page listEmployes.xhmtl) illustrée à la figure 8.17.

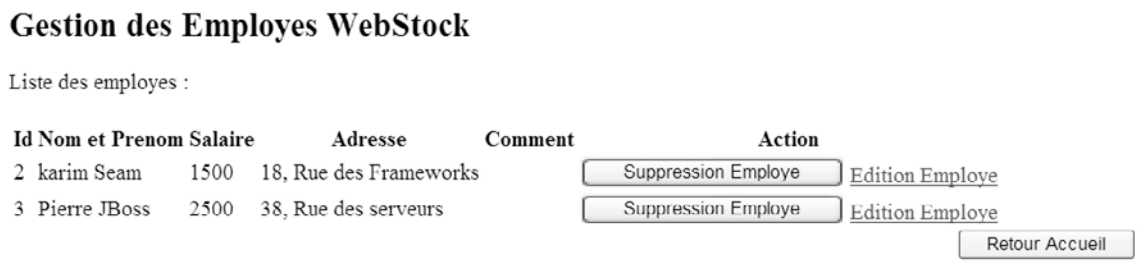


Figure 8.17

Page d'affichage de la liste des employés avec Seam et l'approche POJO

Comme vous pouvez le voir, chaque ligne du tableau contient les données de la base ainsi qu'un bouton et un hyperlien (colonne Action). Ce type de tableau est assez difficile à construire en pur JSF, car il n'existe pas de moyen naturel d'associer un identifiant d'enregistrement avec une gestion d'événements liée à un bouton d'action.

Le framework Seam permet d'implémenter ce genre de tableau activable sans grande difficulté, comme le montre l'extrait suivant de la page employes.xhtml :

```
<h:dataTable value="#{employes}" var="employe">
  <h:column>
    <f:facet name="header">Nom et Prenom</f:facet>
    #{employe.nomprenom}
  </h:column>
  <h:column>
    <f:facet name="header">Salaire</f:facet>
    #{fan.salaire}
  </h:column>
  <h:column>
    <f:facet name="header">Adresse</f:facet>
```

```
    #{ employe.adresse}
  </h:column>
  <h:column>
    <f:facet name="header">Comment</f:facet>
    #{ employe.commentaire}
  </h:column>
  <h:column>
    <f:facet name="header">Action</f:facet>
    <h:column>
    <h:commandButton value="Delete"
                     action="#{manager.delete}"/>
    </h:column>
    <h:column>
    <a href="employe.seam?pid=#{ employe.id}">Edition</a>
  </h:column>
  </h:column>


</h:dataTable>
```

Cette page affiche la liste des données de type `employe` en utilisant une balise d'affichage <h:dataTable> qui itère à partir d'un objet liste préparé par l'annotation @DataModel (voir classe ManagerPojo précédente). Chaque propriété de l'entité métier `Employe` est présentée sous forme de colonne dans un tableau. Le bouton Delete fait également partie du tableau et possède sa méthode de gestion d'événement associée (`#{manager.delete}`).

- Le tableau 8.1 récapitule les annotations associées à JSF pour la couche de présentation.

Tableau 8.1 Annotations associées à JSF

Annotation	Description
@DataModel ("nomVariable")	Représente la liste des éléments. Cette annotation permet l'utilisation d'objets `Collection` à partir de l'EJB dans la balise JSF <h:datatable> pour la partie présentation.
@DataModelSelection	Représente l'élément de la liste DataModel sélectionné.
@DataModelSelectionIndex	Injecte l'index de la ligne sélectionné de ListdataModel, MapDataModel ou ArrayDataModel. À utiliser avec l'annotation @DataModel.
@Factory	Permet l'instanciation de l'objet et le déclenchement de la méthode associée lorsque la requête initiée par la couche présentation requier t ce même objet.

Validation des données

La validation des données d'un formulaire est une fonction que tout framework se doit de supporter, et Seam ne déroge pas à la règle.

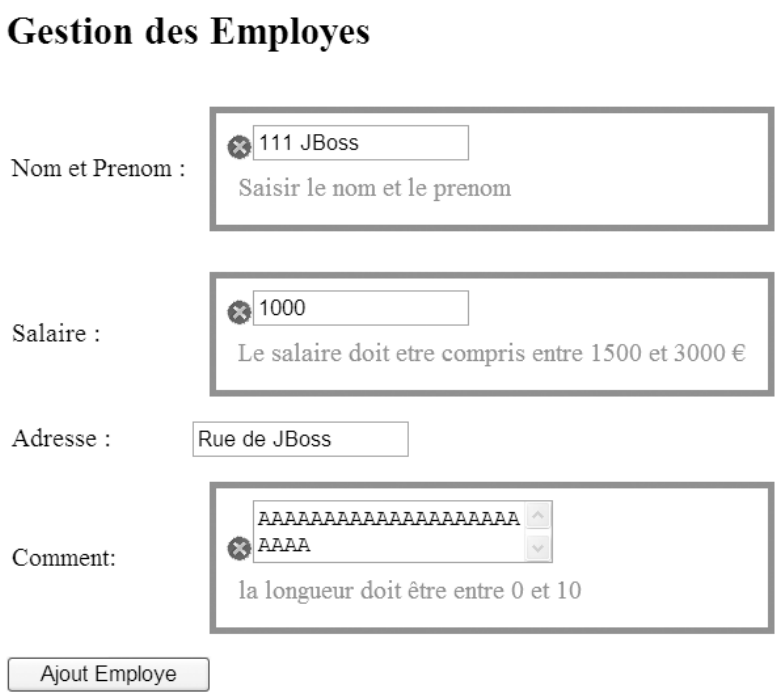
Dans votre application de gestion des employés, vous devez pouvoir valider les champs de formulaire suivants :

- Nom et Prénom : saisie uniquement en caractères alphanumériques.
- Salaire : le montant saisi doit être non nul et compris entre 1 500 et 3 500 €.
- Comment : champ libre limité à 10 caractères.

En cas d'échec de la validation, la page s'affiche à nouveau avec les champs non valides accompagnés d'un message d'erreur, comme l'illustre la figure 8.18.

Figure 8.18

Validation des champs du formulaire avec le framework Seam



Seam valide les entrées utilisateur côté serveur. Dans une approche totalement JSF, la gestion des différentes conditions de validation au sein du formulaire nécessite plusieurs lignes de code, alors qu'avec Seam, seules quelques annotations et balises JSF suffisent, comme le montre le code de la classe Person.java suivante :

```
@Id @GeneratedValue
  public long getId() { return id;}
  public void setId(long id) { this.id = id; }

  @NotNull
  @Pattern(regex="^[a-zA-Z.-]+ [a-zA-Z.-]+",
      message="Saisir le nom et le prenom")
  public String getNomPrenom() { return name; }
  public void setNomPrenom(String name) {this.name = name;}

  // @Min(value=1500) @Max(value=3500)
  @NotNull
  @Range(min=1500, max=3500,
      message="Le salaire doit etre compris entre 1500 et 3000 _")
  public long getSalaire() { return salaire; }
  public void setSalaire(long salaire) { this.salaire = salaire; }

  @NotNull
  public String getAdresse () { return adresse; }
```

```
    public void setAdresse (String adresse) {
      this.adresse = adresse;
    }

    @Length(max=10)
    public String getCommentaire() { return comment; }
    public void setCommentaire(String comment) {
      this.comment = comment;
    }

  }
```

Avec Seam, tous les formulaires sont supportés par les POJO (ou les beans entité), et les contraintes de validation sont directement annotées dans les champs des POJO, comme ici dans les champs `NomPrenom`, `Salaire`, `Adresse` et `Commentaire`.

Chaque annotation prend en argument un attribut message, qui contient le message à afficher dans le formulaire Web si les conditions de validation ne sont pas remplies. Si l'attribut message est omis, un message d'erreur par défaut est utilisé pour le champ annoté. L'attribut @Pattern peut « matcher » le champ saisi à n'importe quelle expression régulière.

Le tableau 8.2 récapitule les annotations disponibles pour la validation des champs de saisie.

Tableau 8.2 Annotations de validation des champs de saisie

Annotation	Description
@Length(max=,min=)	S'applique à un type de donnée chaîne pour vérifier que la taille de la chaîne de caractères est bien dans l'intervalle de longueur défini.
@Max(value=)	S'applique à une propriété numérique ou une représentation chaîne de caractères d'une valeur numérique afin de vérifier que la valeur de la propriété est plus basse que la valeur Max spécifée en paramètre.
@Min(value=)	S'applique à une propriété numérique ou une représentation chaîne de caractères d'une valeur numérique afin de vérifier que la valeur de la propriété est plus grande que la valeur Min spécifée en paramètre.
@NotNull	S'applique à n'importe quel type de propriété pour vérifier que cette propriété est non nulle.
@Past	S'applique à une propriété de type Date ou Calendar pour vérifier si la date est périmée.
@Future	S'applique à une propriété de type Date ou Calendar pour vérifier si la date s'applique à une date future.
@Pattern(regex="regexp", flag=)	S'applique à une propriété de type chaîne pour vérifier si la chaîne respecte l'expression régulière en paramètre. Le paramètre flag permet de valider si l'expression respecte ou non la casse.
@Range(max=,min=)	S'applique à une propriété numérique ou une représentation chaîne de caractères d'une valeur numérique afin de vérifier que la valeur de la propriété est comprise dans l'intervalle fixé.
@Size(max=,min=)	S'applique à une propriété de type collection ou tableau pour vérifier que le nombre d'éléments de la propriété est compris dans l'intervalle [min,max].
@Email	S'applique à une propriété de type chaîne pour vérifier que la valeur de la propriété est sous un format email.
@Valid	S'applique à n'importe quel type de propriété (collection, array ou map). Cette annotation effectue une validation récursive sur l'objet associé.

Même si ces annotations couvrent la majorité des besoins de validation des applications Web, Seam offre la possibilité de construire vos propres annotations de validation. Référez-vous pour cela à la documentation officielle du framework.

En résumé

Vous avez achevé l'examen de l'outillage Web avec Web Tools et découvert les fonctionnalités remarquables du framework Seam pour le développement d'applications Web complexes et riches.

Vous aborderez à la partie III de l'ouvrage l'étude et la mise en œuvre de l'outillage EJB3 avec Eclipe et JBoss.