1

Introduction

Les développements Java/J2EE, notamment ceux qui utilisent les EJB, sont réputés complexes, tant en terme de développement que de tests et de maintenance. La productivité des développeurs Java/J2EE ne disposant que des standards constituant la plateforme est faible, comparée à celle obtenue avec d'autres technologies, comme les langages de type RAD (Rapid Application Development). Sur les projets Java/J2EE de taille conséquente, une couche d'abstraction est généralement développée afin de simplifier le travail des développeurs et leur permettre de se concentrer davantage sur la réelle valeur ajoutée d'une application, à savoir le métier.

C'est à partir de ce constat que les concepteurs de Spring, experts J2EE de renommée mondiale, ont imaginé une solution permettant de simplifier et structurer les développements J2EE de manière à respecter les meilleures pratiques d'architecture logicielle.

Spring est à ce titre un véritable cadre de travail. Non content de structurer les développements spécifiques d'un projet, il propose une intégration des frameworks tiers les plus répandus. Grâce aux communautés Open Source très actives du monde Java, un grand nombre de frameworks spécialisés sur différentes problématiques sont apparus, notamment Struts pour la couche présentation des applications et Hibernate pour la persistance des données. Ainsi, il n'est pas rare d'utiliser au sein d'un même projet plusieurs frameworks spécialisés.

Spring simplifie l'utilisation de ces frameworks en les insérant dans son cadre de travail. La vocation de Spring est d'offrir, non pas toutes les fonctionnalités dont un développeur pourrait rêver, mais une architecture applicative générique capable de s'adapter à ses choix technologiques en terme de frameworks spécialisés.

Cet ouvrage ambitionne de proposer une vision globale de Spring, afin d'en présenter toute la richesse et d'en faciliter la mise en œuvre dans des projets. Nous commençons dans ce chapitre par aborder les problématiques rencontrées dans les projets Java/J2EE classiques, à savoir la séparation des préoccupations techniques et fonctionnelles, la productivité des développements, l'indépendance du code vis-à-vis de la plate-forme d'exécution et les tests.

Spring apporte des réponses à ces problématiques en se reposant sur la notion de conteneur léger et sur la POA (programmation orientée aspect) et en introduisant les meilleures pratiques en matière d'architecture applicative et de développement d'applications.

En fin de chapitre, nous introduisons l'étude de cas Tudu Lists, tirée d'un projet Open Source, qui nous servira de fil conducteur tout au long de l'ouvrage. Au travers de sa mise en œuvre concrète, nous illustrerons tous les principes fondamentaux de Spring ainsi que l'intégration des différentes technologies manipulées par les projets Java J2EE (persistance des données, gestion des transactions, etc.).

Problématiques des développements Java/J2EE

J2EE constitue une infrastructure complexe, qui demande un niveau technique non négligeable pour être bien maîtrisée. Par rapport au développement d'applications métier, cette infrastructure soulève un certain nombre de difficultés, auxquelles tout développeur Java/J2EE est confronté un jour ou l'autre, notamment les quatre problèmes majeurs suivants :

- J2EE n'encourage pas intrinsèquement une bonne séparation des préoccupations, c'est-à-dire l'isolation des différentes problématiques qu'une application doit gérer (typiquement, les problématiques techniques, d'une part, et les problématiques métier, d'autre part).
- J2EE est une plate-forme complexe à maîtriser, qui pose des problèmes de productivité importants, la part des développements consacrés aux problématiques techniques étant disproportionnée par rapport à celle vouée aux problématiques fonctionnelles, qui est pourtant la véritable valeur ajoutée d'une application.
- J2EE impose une plate-forme d'exécution lourde, qui pose des problèmes d'interopérabilité entre différentes implémentations. Elle peut aussi nuire à la réutilisation des services composant une application, si ceux-ci doivent fonctionner sur du matériel ne pouvant supporter un serveur d'applications dans le cadre d'une utilisation nomade (assistants personnels, téléphones, etc.).
- Les développements utilisant J2EE s'avèrent souvent difficiles à tester du fait de leur forte dépendance vis-à-vis de cette plate-forme et de la lourdeur de celle-ci.

Nous détaillons dans cette section ces différentes problématiques, qui nous serviront de référence dans le reste de l'ouvrage pour illustrer tout l'intérêt d'utiliser un framework tel que Spring dans nos développements J2EE.

La séparation des préoccupations

Le concept de séparation des préoccupations consiste à isoler au sein des applications les différentes problématiques qu'elles ont à traiter. Deux catégories de préoccupations parmi les plus évidentes sont les préoccupations d'ordre technique, à l'image des mécanismes de persistance des données, et les préoccupations d'ordre métier, comme la gestion des données de l'entreprise.

L'idée sous-jacente à la séparation des préoccupations est de garantir une meilleure évolutivité des applications grâce à une bonne isolation des différentes problématiques. Chaque préoccupation est traitée de la manière la plus indépendante possible des autres afin d'en pérenniser au maximum le code, tout en limitant les effets de bord liés à l'évolution d'une préoccupation.

En spécifiant J2EE (Java 2 Enterprise Edition), Sun Microsystems visait à transformer le langage Java en un véritable langage d'entreprise, permettant une séparation nette entre les développeurs « techniques » et les développeurs « métier ». L'objectif avoué de cette séparation était de permettre à des non-informaticiens de prendre en charge directement le développement des composants implémentant la logique métier, en l'occurrence les EJB (Enterprise JavaBeans), en connaissant le strict minimum de programmation nécessaire. Force est de constater que les EJB n'ont pas tenu leurs promesses.

L'une des raisons de cet échec est que la séparation des préoccupations au sein d'une architecture J2EE se fondant uniquement sur les EJB n'est pas suffisante pour isoler complètement les développements métier des problématiques techniques. La conception même des composants métier est directement influencée par l'architecture technique sous-jacente. Combien de projets J2EE ont échoué pour avoir utilisé des EJB d'une manière ingérable par le serveur d'applications, par exemple en définissant des EJB Session ayant une granularité trop fine ?

Par ailleurs, la séparation des préoccupations est limitée à des préoccupations spécifiques, et J2EE ne propose pas de mécanisme générique permettant d'élargir ce périmètre. En l'occurrence, la séparation des préoccupations porte essentiellement sur les EJB, alors qu'une application ne se résume pas, loin s'en faut, à ces derniers. Les préoccupations techniques liées à la persistance des données, aux transactions ou à la sécurité peuvent concerner bien d'autres composants de l'application.

La productivité des développements

Si les normes J2EE définissent une infrastructure technique de base pour développer des applications, celle-ci s'avère insuffisante en terme de productivité par rapport à d'autres technologies. Ces normes occultent le problème de la productivité, laissant place à des solutions complémentaires masquant la complexité de la technologie mais faisant perdre en même temps les gains attendus de la standardisation.

La mise en place de ces solutions, régulièrement développées spécifiquement pour un projet, a impacté fortement les budgets d'investissement, souvent aux dépens des développements strictement métier. Si l'utilisation de technologies standards est un gage de

pérennité, la seule standardisation ne suffit pas à répondre aux besoins des clients, dont l'objectif est toujours d'avoir plus pour moins cher.

Pour combler les lacunes de la plate-forme J2EE, la lenteur de son instance de standardisation, le JCP (Java Community Process), et éviter d'avoir à développer et maintenir des solutions maison, la communauté Java, a dû s'appuyer sur des offres complémentaires, notamment Open Source.

Pour gagner en productivité, les projets Java/J2EE actuels abandonnent les solutions maison au profit de frameworks tiers implémentant les meilleures pratiques disponibles. L'exemple qui vient immédiatement à l'esprit est celui de Struts. Ce framework, ou cadre de travail, structure les développements Web selon un modèle de conception éprouvé depuis le langage SmallTalk, le modèle MVC (Model View Controller) et fournit une boîte à outils rudimentaire pour structurer et accélérer les développements.

Malheureusement, les frameworks sont des outils spécialisés dans une problématique donnée, à l'image de Struts pour le Web ou Log4J pour les traces applicatives. Pour parvenir à un niveau de productivité satisfaisante, il est donc nécessaire de recourir à plusieurs frameworks. Cela pose d'évidents problèmes d'intégration de ces frameworks au sein des applications, problèmes auxquels J2EE n'apporte aucune réponse.

L'indépendance vis-à-vis de la plate-forme d'exécution

Pour respecter un de ses principes fondateurs, le fameux « Write Once, Run Anywhere » (écrire une fois, exécuter partout), Java repose sur une machine virtuelle permettant de s'abstraire des plates-formes matérielles sous-jacentes.

Avec J2EE, ce principe fondateur se heurte à deux écueils : la difficulté de standardisation d'une technologie complexe et le choix d'une architecture exclusivement orientée serveur.

La standardisation est un processus long et difficile. Si plusieurs acteurs sont impliqués, il est nécessaire de trouver un consensus sur le contenu du standard. Par ailleurs, ce standard doit laisser le moins de place possible à l'ambiguïté et à l'imprécision, autant de niches engendrant des implémentations incompatibles.

Au finale, les premières implémentations du standard J2EE se sont avérées assez liées au serveur d'applications sous-jacent. La situation s'est nettement améliorée depuis, avec la mise en place d'implémentations de référence et d'un processus de certification très rigoureux. Cependant, certains aspects, comme la sécurité, ne sont que partiellement couverts par la norme J2EE, ouvrant la porte aux solutions propriétaires.

Le deuxième écueil vient de l'orientation exclusivement serveur de J2EE, un choix « politique » de la part de Sun Microsystems, selon qui « the Network is the Computer » (le réseau est l'ordinateur). Malheureusement, cette orientation réseau est préjudiciable à certains types d'applications, notamment celles qui doivent fonctionner sans réseau et qui nécessitent généralement des services de persistance des données et de gestion des transactions. Le code métier reposant sur les EJB nécessite un serveur d'applications pour s'exécuter. Or l'installation d'un serveur d'applications J2EE directement sur un poste de travail nomade ou un assistant personnel est difficilement justifiable.

Finalement, J2EE répond à des problématiques fortes mais pose des problèmes d'interopérabilité au niveau des EJB, qui nuisent à la réutilisation des composants construits sur cette architecture. Par ailleurs, l'orientation réseau de J2EE gêne son utilisation dans certains domaines comme le nomadisme.

Les tests

La problématique des tests est fondamentale dans tout développement d'application. La technologie J2EE s'avère complexe à tester, car elle nécessite un serveur d'applications pour s'exécuter. De cette nécessité résultent une exécution des tests lourde et une difficulté d'automatisation.

Dans les faits, il s'agit davantage de tests d'intégration que de tests réellement unitaires. Des solutions telles que Cactus, de la communauté Apache Jakarta, restent lourdes à mettre en œuvre par rapport à l'outillage disponible pour les classes Java classiques (frameworks dérivés de JUnit, Mock Objects, etc.).

La difficulté de tester une application Java/J2EE selon une granularité plus fine que les tests d'intégration nuit fortement à la qualité des applications construites sur cette plate-forme. D'où des problèmes de productivité et de maintenance évidents.

En résumé

Nous avons vu que J2EE posait des problèmes pénalisants pour le développement des applications. J2EE n'encourage pas intrinsèquement de bonnes pratiques de conception, comme la séparation des préoccupations. La lourdeur de cette plate-forme rend de surcroît l'application dépendante du serveur d'applications sur lequel elle fonctionne, et les tests de ses composants s'avèrent difficiles à mettre en œuvre.

Face à ce constat, les concepteurs de Spring ont développé un framework permettant d'adresser ces problématiques de manière efficace.

Réponses de Spring

Pour résoudre les problèmes que nous venons d'évoquer, des solutions ont émergé. En rupture avec les conteneurs J2EE, disqualifiés par de nombreux experts pour leur lourdeur, sont apparus des conteneurs dits légers. Le cœur de Spring entre dans cette catégorie de solutions.

Ce cœur a été étendu de manière à supporter la POA (programmation orientée aspect), ou AOP (Aspect Oriented Programming), un nouveau paradigme de programmation permettant d'aller au-delà de l'approche objet en terme de modularisation des composants. Au-dessus de ce socle, des modules optionnels sont proposés afin de faciliter l'intégration de frameworks spécialisés.

Les sections qui suivent détaillent la façon dont Spring résout les problèmes soulevés par l'utilisation de J2EE.

La notion de conteneur léger

La notion de conteneur EJB est une technologie lourde à mettre en œuvre, inadaptée pour les objets à faible granularité, intrusive dans le code et en partie dépendante du serveur d'applications, ne serait-ce qu'en terme de configuration.

En fournissant une infrastructure de gestion de l'ensemble des composants de l'application, les conteneurs légers adoptent une approche foncièrement différente. Cela s'effectue au travers de mécanismes de configuration, comme les fichiers XML permettant d'initialiser les composants, de gestion du cycle de vie des composants et de gestion des dépendances entre les composants. Les conteneurs légers sont indépendants de la technologie J2EE et peuvent fonctionner sur tout type de serveur d'applications, voire sans eux en utilisant une simple machine virtuelle Java.

Les conteneurs légers rendent les applications qui les utilisent à la fois plus flexibles et mieux testables, car le couplage entre les composants est géré par le conteneur, et non plus directement à l'intérieur du code. Par ailleurs, les conteneurs légers encouragent l'utilisation intensive d'interfaces afin de rendre l'application indépendante de leurs implémentations. Cela se révèle notamment utile pour les tests, dans lesquels il est souvent nécessaire de remplacer une implémentation réelle par un simulacre d'objet, ou Mock Object. Le débogage est facilité, puisqu'il n'est pas nécessaire d'exécuter le code au sein d'un serveur d'applications et que le conteneur léger est peu intrusif dans le code de l'application.

Pour nous faire une idée plus précise de la nature d'un conteneur léger, nous pouvons établir un parallèle avec les frameworks gérant les interfaces graphiques, comme Swing ou SWT (Standard Widget Toolkit). Avec ces outils, nous définissons les composants graphiques à utiliser et nous nous connectons aux événements qu'ils sont susceptibles de générer, un clic sur un bouton, par exemple. Ces frameworks prennent en charge le cycle de vie des composants graphiques de manière complètement transparente pour l'application concernée. Un conteneur léger peut offrir des services similaires, mais avec des objets de toute nature.

Nous détaillons au chapitre 2 le fonctionnement des conteneurs légers et abordons les modèles de conception, ou design patterns, sur lesquels ils se fondent.

À la différence d'autres conteneurs légers du marché, notamment HiveMind ou PicoContainer, Spring propose bien d'autres fonctionnalités, comme le support de la POA ou l'intégration de frameworks tiers.

Le support de la POA

Outre son conteneur léger, Spring supporte la POA (programmation orientée aspect), ou AOP (Aspect-Oriented Programming). Ce support est utilisé en interne par Spring pour offrir des services similaires à ceux des conteneurs EJB, mais sans leur lourdeur, car ils sont applicables à de simples JavaBeans, ou POJO (Plain Old Java Objects). Ce support peut aussi être utilisé par les utilisateurs de Spring pour leurs propres besoins.

La POA est un nouveau paradigme de programmation, dont les fondations ont été définies au centre de recherche Xerox, à Palo Alto, au milieu des années 1990. Par paradigme, nous entendons un ensemble de principes qui structurent la manière de modéliser les applications et, en conséquence, la façon de les développer.

Elle a émergé à la suite de différents travaux de recherche, dont l'objectif était d'améliorer la modularité des logiciels afin de faciliter la réutilisation et la maintenance.

La POA ne remet pas en cause les autres paradigmes de programmation, comme l'approche procédurale ou l'approche objet, mais les étend en offrant des mécanismes complémentaires, afin de mieux séparer les différentes préoccupations d'une application, et une nouvelle dimension de modularisation, l'*aspect.*

Dans son principe, la POA consiste à modulariser les éléments logiciels mal pris en compte par les paradigmes classiques de programmation. En l'occurrence, la POA se concentre sur les éléments transversaux, c'est-à-dire ceux qui se trouvent dupliqués ou utilisés dans un grand nombre d'entités, comme les classes ou les méthodes, sans pouvoir être centralisés au sein d'une entité unique avec les concepts classiques. Ainsi, grâce à la notion d'aspects, qui capturent en leur sein les préoccupations transversales, la séparation des préoccupations est nettement améliorée.

Avec les EJB, notamment les EJB Entity CMP, un premier niveau de séparation des préoccupations a été atteint. Un grand nombre d'aspects techniques sont en effet pris en charge sous forme de descripteurs de déploiement (mapping objet-relationnel, sécurité et transactions, etc.) et n'apparaissent plus dans le code métier. La maintenance en est d'autant facilitée.

La POA va au-delà en offrant des mécanismes génériques pour modulariser un grand nombre d'éléments transversaux des logiciels. Les limites des outils de POA pour capturer les préoccupations transversales tiennent essentiellement à leur capacité à exprimer le périmètre de celles-ci et à la façon dont elles sont liées au reste de l'application.

Spring n'implémente pas toutes les notions de la POA, mais l'essentiel est présent et facilement utilisable. Pour des besoins plus poussés, l'utilisation d'un framework tiers est permise. Spring s'intègre notamment avec AspectJ, l'outil précurseur de la POA et qui demeure le plus populaire à ce jour, mais cette intégration se limite aux aspects de type singleton.

Diverses fonctionnalités avancées de Spring reposent sur la POA, notamment les suivantes :

- gestion des transactions ;
- gestion de cache ;
- notification d'événements.

Grâce à la POA, Spring offre des services proches de ceux fournis par J2EE, mais sans se limiter aux seuls EJB. Ces services peuvent de la sorte être utilisés avec tout objet Java.

L'intégration de frameworks tiers

L'intégration de frameworks tiers dans Spring repose sur la notion de conteneur léger et, dans une moindre mesure, sur la POA.

Cette intégration peut s'effectuer aux trois niveaux suivants :

- intégration des composants du framework tiers au sein du conteneur léger et configuration de ses ressources ;
- mise à disposition d'un template, ou modèle, d'utilisation et de classes de support ;
- abstraction de l'API.

En fonction du framework, l'intégration de celui-ci est plus ou moins poussée, le minimum étant l'intégration des composants du framework dans le conteneur léger. Cette intégration consiste à configurer et réaliser une injection de dépendances sur ces composants. Elle est généralement peu intrusive du point de vue de l'application qui utilise le framework tiers, c'est-à-dire qu'aucune dépendance forte n'est créée à l'égard de Spring.

Le deuxième degré d'intégration consiste en la mise à disposition d'un template d'utilisation et de classes de support. Il s'agit ici de faciliter l'utilisation du framework tiers en simplifiant les appels à son API et en implémentant les meilleures pratiques.

Pour cela, différentes techniques sont utilisées. L'une d'elles consiste en la transformation systématique des exceptions vérifiées *(checked exceptions),* c'est-à-dire les exceptions devant être obligatoirement prises en charge par une clause `catch` dans le code et les clauses `throws` spécifiées dans les signatures des méthodes, en exceptions d'exécution *(runtime exceptions),* plus simples à gérer de manière centralisée. Dans ce cas, l'utilisation d'un template lie le code de l'application explicitement à Spring.

Le troisième degré d'intégration consiste à s'abstraire de l'API du framework tiers. L'objectif est ici de normaliser l'utilisation d'un ensemble de frameworks répondant à des besoins similaires. C'est typiquement le cas du support des DAO (Data Access Objects), ou objets d'accès aux données, par Spring, qui normalise l'utilisation de ce concept quel que soit le framework de persistance utilisé (Hibernate, OJB, etc.).

L'abstraction d'API ne permet toutefois pas de s'abstraire complètement des frameworks sous-jacents, car cela reviendrait à choisir le plus petit dénominateur commun et à perdre ainsi la richesse des fonctionnalités avancées de ces frameworks. Le parti pris de Spring consiste à abstraire les points communs sans pour autant masquer le framework tiers. C'est un choix pragmatique, qui diminue les coûts de remplacement d'un framework par un autre, sans occulter les gains de productivité induits par leur utilisation poussée.

Grâce à ces différents degrés d'intégration, Spring constitue le ciment permettant de lier les différents frameworks mis en œuvre avec le code applicatif dans un ensemble cohérent et implémentant les meilleures pratiques. La productivité et la maintenabilité des applications en sont d'autant améliorées.

Architecture globale de Spring

À partir de ces trois principes que sont le conteneur léger, la POA et l'intégration de frameworks tiers, Spring propose un cadre de développement permettant de construire des applications de manière modulaire.

La figure 1.1 illustre les différentes briques constituant l'architecture de Spring, ainsi que les principaux frameworks tiers (représentés en pointillés) dont elles assurent l'intégration.

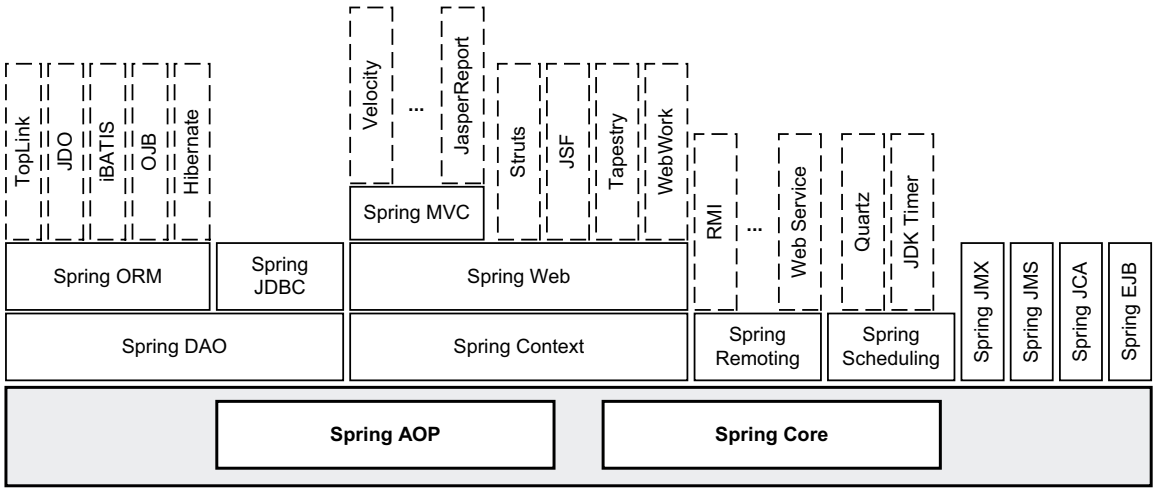


Figure 1.1
Architecture de Spring

Comme nous pouvons le voir, Spring repose sur un socle technique constitué des modules :

- Spring Core, le conteneur léger ;
- Spring AOP, le framework de POA.

Sur ce socle sont construits des modules de plus haut niveau destinés à intégrer des frameworks tiers ou à fournir des fonctions de support. Ces modules sont les suivants :

- Modules d'intégration de la persistance des données (Spring DAO pour l'abstraction de la notion d'objets d'accès aux données, Spring ORM pour l'intégration de frameworks de persistance, Spring JDBC pour simplifier l'utilisation de JDBC). Les principaux frameworks de persistance du marché sont supportés.
- Module de gestion de contexte applicatif (Spring Context), qui assure le dialogue entre Spring et l'application, indépendamment de la plate-forme technique sur laquelle fonctionne cette dernière (au sein d'un serveur d'applications, d'une simple JVM, etc.).

- Module d'intégration de frameworks Web (Spring Web), qui supporte les principaux frameworks Open Source du marché. Spring propose en outre son propre framework Web, conçu sur le modèle MVC, sous le nom de Spring MVC.
- Module de distribution d'objets (Spring Remoting), qui permet de transformer de simples classes Java en objets distribués RMI, Web Services ou autres.
- Module d'intégration d'ordonnanceur de tâches, qui supporte actuellement Quartz d'OpenSymphony et le Timer fourni par le JDK.
- Modules de support des différentes technologies J2EE (JMX, JMS, JCA et EJB).

Comme nous pouvons le constater, l'offre de Spring couvre l'essentiel des frameworks utilisés par les applications J2EE actuelles, ce qui en fait une solution pertinente pour tout développement. De plus, Spring n'est pas une solution fermée, comme nous le verrons à la section suivante.

Spring propose enfin les sous-projets suivants, qui viennent compléter ses fonctionnalités :

- Acegi Security System for Spring, qui permet de gérer l'authentification et les autorisations au sein d'une application développée avec Spring.
- Spring Web Flow, qui permet de gérer la navigation au sein de pages Web.
- Spring Rich Client, qui permet d'accélérer le développement d'applications Swing en se fondant sur Spring.
- Spring BeanDoc, qui permet de documenter l'utilisation des artefacts de Spring utilisés dans les applications.
- Spring IDE, un plug-in Eclipse qui permet d'accélérer les développements en gérant les composants de Spring au sein de cet environnement de développement.

Le projet Spring Modules (voir *https://springmodules.dev.java.net/*) a pour objectif d'intégrer d'autres frameworks et outils sans modifier le cœur de Spring. À ce jour, Spring Modules intègre des frameworks tels que OSWorkflow, Commons Validator, Drools, etc.

Spring dispose aussi d'une implémentation pour le monde Microsoft, avec Spring.Net *(http://www.springframework.net/)*.

En résumé

Spring est une solution élégante qui répond à un ensemble de problématiques essentielles des développements Java/J2EE, notamment la flexibilité du code, l'intégration de frameworks tiers et l'implémentation des meilleures pratiques de programmation.

Du fait de son indépendance vis-à-vis de la plate-forme J2EE, il est utilisable sur tout type de développement Java reposant sur J2SE. Il convient aussi bien aux applications de type Web (grâce à Spring MVC ou à son intégration des principaux frameworks MVC) ou Swing (notamment grâce au sous-projet Spring Rich Client).

Certains reprochent à Spring de ne pas être un standard, à la différence de J2EE. Force est pourtant de constater qu'un développement J2EE est souvent difficile à migrer d'un serveur d'applications à un autre, alors que les développements qui utilisent Spring et s'abstiennent d'utiliser les EJB le sont beaucoup moins. Par ailleurs, Spring laisse le choix au développeur du niveau d'adhérence de son code vis-à-vis du framework, ce qui est moins le cas de J2EE. Par exemple, en n'utilisant que le conteneur léger et la POA, l'impact du remplacement de Spring par une autre solution est relativement faible.

Présentation de l'étude de cas Tudu Lists

Tout au long de l'ouvrage, nous illustrons notre propos au moyen d'une étude de cas, Tudu Lists, qui emploie les fonctionnalités majeures de Spring. Au travers de cette étude de cas, nous donnons une vision globale de l'architecture d'une application fondée sur Spring, ainsi que de nombreux exemples de mise en œuvre.

Dans le contexte de cet ouvrage, nous utilisons une préversion 2.0 de Spring, la version finale n'étant pas disponible au moment où nous mettons sous presse. L'API de Spring étant remarquablement stable, la version finale ne devrait pas introduire de dysfonctionnements majeurs. Si tel devait néanmoins être le cas, un addendum précisant les changements à effectuer serait mis à la disposition des lecteurs sur la page Web dédiée à l'ouvrage du site des éditions Eyrolles, à l'adresse *http://www.editions-eyrolles.com.*

Tudu Lists est un projet Open Source créé par Julien Dubois, l'un des auteurs du présent ouvrage, et auquel les autres auteurs de l'ouvrage participent. Ce projet consiste en un système de gestion de listes de choses à faire *(todo lists)* sur le Web. Il permet de partager des listes entre plusieurs utilisateurs et supporte le protocole RSS (Really Simple Syndication). Les listes de choses à faire sont des outils de gestion de projet simples mais efficaces. Tudu Lists est hébergé sur le site communautaire SourceForge *(http://tudu.sourceforge.net/)* pour sa partie développement.

L'utilisation de Tudu Lists est très simple comme nous allons le voir. La page d'accueil se présente de la manière illustrée à la figure 1.2.

Pour créer un nouvel utilisateur, il suffit de cliquer sur le lien « register » et de remplir le formulaire. Une fois authentifié, l'utilisateur peut gérer ses listes de todos.

Par commodité, nous utilisons à partir de maintenant le terme « todo » pour désigner une chose à faire, comme illustré à la figure 1.3.

Pa le biais des onglets disposés en haut de la page, nous pouvons gérer notre compte (My info), nos listes de todos (My Todo Lists), nos todos (My Todos) ou nous déloguer (Log out).

La création d'une liste est on ne peut plus simple. Il suffit de cliquer sur l'onglet My Todo Lists puis sur le lien Add a new Todo List et de remplir le formulaire et cliquer sur le lien Submit, comme illustré à la figure 1.4.

La création d'un todo suit le même principe. Dans l'onglet My Todos, il suffit de cliquer sur la liste dans laquelle nous allons ajouter un todo (les listes sont affichées dans la partie gauche de la page) puis de cliquer sur Add a new Todo.

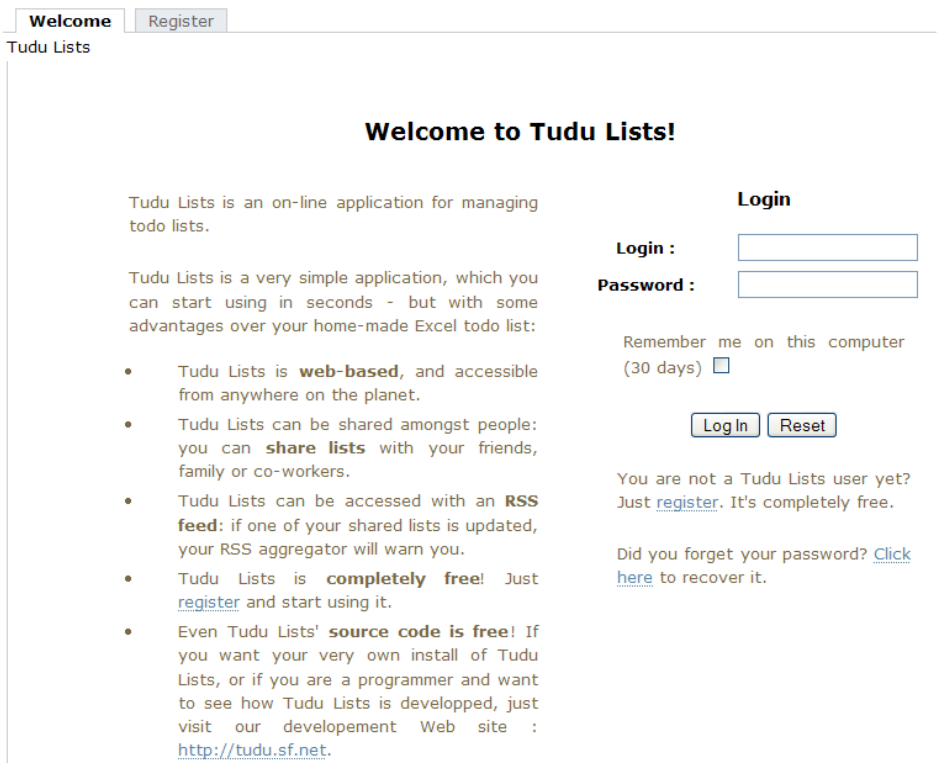


Figure 1.2
Page d'accueil de Tudu Lists

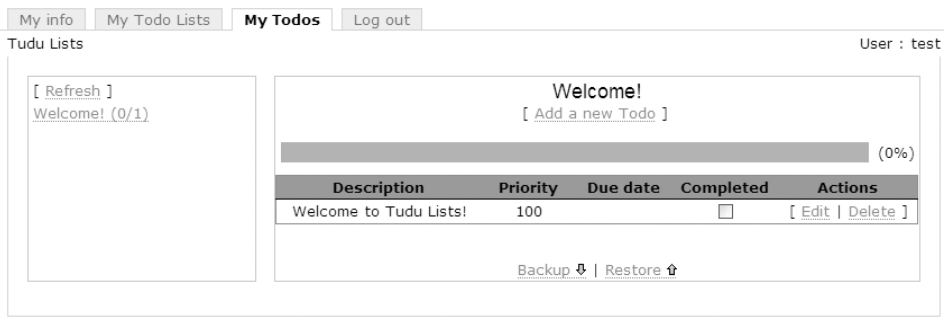


Figure 1.3
Liste de todos

Figure 1.4
Création d'une liste de todos

Pour finir, nous pouvons sauvegarder le contenu d'une liste en cliquant sur le lien Backup. De même, nous pouvons restaurer le contenu d'une liste en cliquant sur le lien Restore.

Nous avons vu l'essentiel des fonctions de Tudu Lists. Pour une description plus détaillée de son utilisation, voir la documentation en ligne, sur le site *http://tudu.sourceforge.net.*

Architecture du projet Tudu Lists

Tudu Lists est une application Web, conçue pour démontrer que l'utilisation de Spring et de frameworks spécialisés permet d'obtenir sans développement lourd une application correspondant à l'état de l'art en terme de technologie.

Dans cette section, nous décrivons de manière synthétique les principes architecturaux de Tudu Lists. Ces informations seront utiles pour manipuler l'étude de cas tout au long de l'ouvrage.

Nous listons dans un premier temps les frameworks utilisés par Tudu Lists puis abordons la gestion des données, notamment la modélisation de ces dernières. Nous terminons avec les principes de la modélisation objet de Tudu Lists.

Les frameworks utilisés

Outre Spring, Tudu Lists utilise les frameworks Open Source suivants :

- Hibernate, le célèbre framework de mapping objet-relationnel, pour assurer la persistance des données.
- Struts, pour assurer l'implémentation du modèle de conception MVC. Dans le cadre de cet ouvrage, nous proposons aussi une version spéciale de Tudu Lists utilisant Spring MVC en lieu et place de Struts.
- Struts Menu, pour gérer les menus de l'application.
- DWR (Direct Web Remoting) pour implémenter les fonctionnalités AJAX (Asynchronous JavaScript And XML) de Tudu Lists. AJAX est une technologie fondée sur JavaScript destinée à améliorer l'expérience de l'utilisateur devant une interface homme-machine (IHM) en limitant les échanges entre le navigateur Web et le serveur d'applications à des messages XML au lieu de pages Web complètes. Nous aurons l'occasion de décrire plus en détail cette technologie au chapitre 9, consacré à AJAX et DWR.
- Acegi Security, pour gérer les authentifications et les autorisations.
- JAMon (Java Application Monitor), pour surveiller les performances de Tudu Lists.
- Log4J, pour gérer les traces applicatives.
- Rome, pour gérer les flux RSS.

Comme nous le constatons, bien que Tudu Lists implémente des fonctionnalités simples à appréhender, il met en œuvre un grand nombre de frameworks.

Grâce aux services rendus par ces différents frameworks et à l'intégration assurée par Spring, le nombre de lignes de code de Tudu Lists reste raisonnable (3 200 lignes pour

une quarantaine de classes et interfaces, avec une moyenne de 6 lignes par méthode) en comparaison de l'équivalent développé uniquement avec J2EE.

Modèle conceptuel de données

Tudu Lists utilise une base de données relationnelle pour stocker les informations sur les utilisateurs et les listes de todos.

Le modèle conceptuel de données de Tudu Lists s'articule autour de quelques tables, comme l'illustre la figure 1.5.

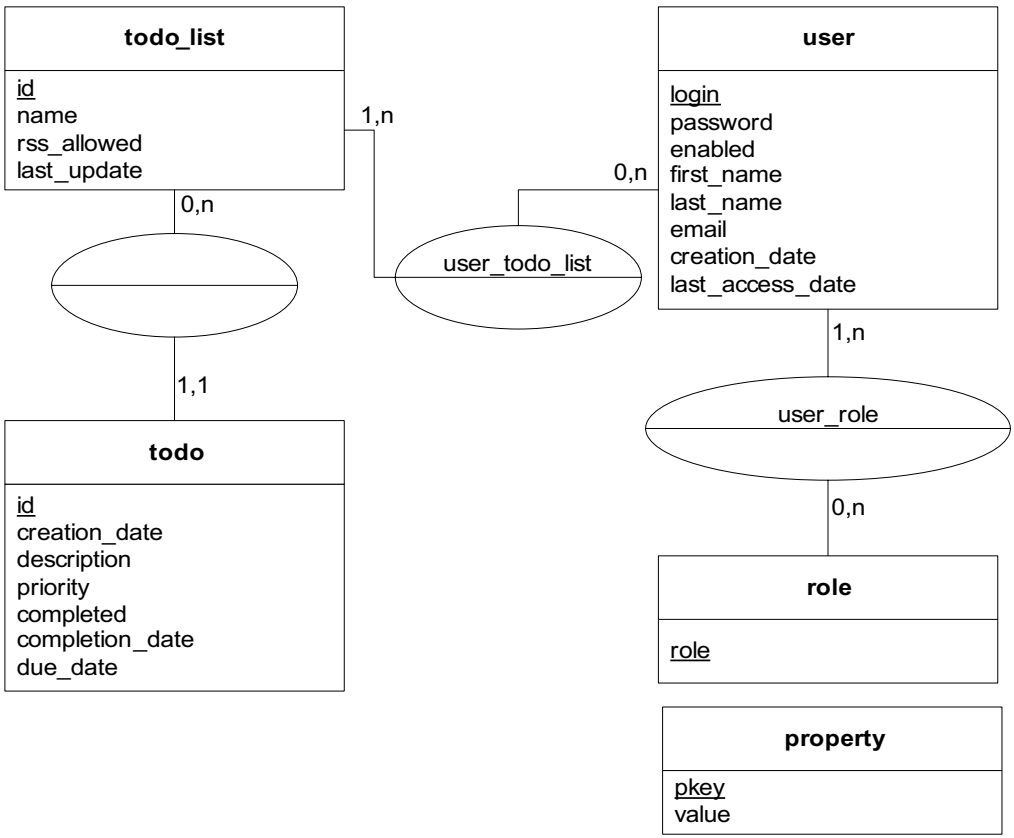


Figure 1.5

Modèle conceptuel de données de Tudu Lists

Remarquons que la table `property` sert à stocker les paramètres internes de Tudu Lists, notamment l'adresse du serveur SMTP nécessaire à l'envoi d'e-mail. Par ailleurs, la table `role` ne contient que deux lignes, chacune représentant les deux rôles gérés par Tudu Lists : administrateur et utilisateur.

Pour implémenter ce modèle de données, nous utilisons le SGBDR Open Source HSQLDB *(http://hsqldb.org/)*. À l'origine, Tudu Lists était prévu pour fonctionner avec le SGBDR MySQL, mais nous avons remplacé ce dernier pour simplifier l'installation de l'étude de cas. En effet, HSQLDB est un SGBDR écrit en Java, dont l'exécution peut se faire au sein même de l'application Web. Il est donc directement embarqué dans l'application, simplifiant grandement la problématique du déploiement.

Pour utiliser Tudu Lists dans un environnement de production, nous conseillons d'utiliser la version de MySQL disponible sur le site de Tudu Lists, car ce SGBDR est prévu pour supporter de fortes charges, contrairement à HSQLDB.

HSQLDB et MySQL

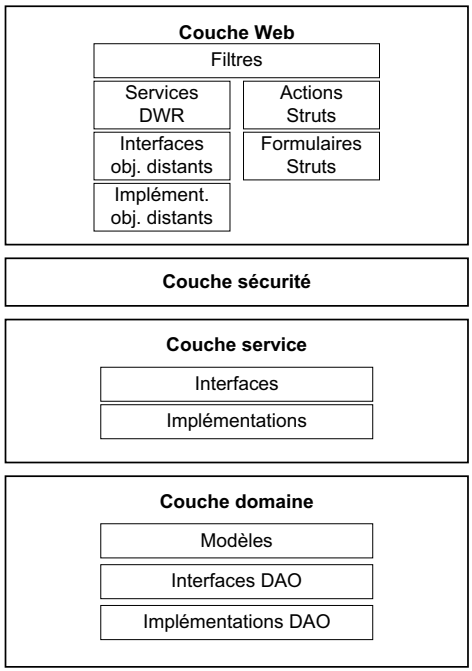
Le remplacement de MySQL est facilité par l'architecture de Tudu Lists. L'essentiel du travail consiste à adapter le script SQL de création des tables, celui de MySQL contenant des spécificités propres à ce SGBDR. Il suffit d'apporter ensuite des modifications à quelques fichiers de configuration portant sur la définition de la source de données et le réglage du dialecte SQL utilisé par Hibernate.

Les principes du modèle objet

Nous ne détaillons pas ici le modèle objet de Tudu Lists, car il est décrit tout au long des chapitres de l'ouvrage. Ce modèle suit la philosophie de conception préconisée par Spring, à savoir un découpage clair des couches composant l'application.

La figure 1.6 illustre les couches de Tudu Lists.

Figure 1.6
Couches de Tudu Lists



La couche domaine

La couche domaine prend en charge le modèle objet métier de Tudu Lists. C'est dans cette couche que nous trouvons les classes modélisant et implémentant les notions de liste de todos, de todos, etc.

Cette couche est constituée de trois sous-ensembles :

- Le sous-ensemble des modèles contient les objets qui sont manipulés par les autres couches de l'application. C'est dans ce sous-ensemble, offrant le plus haut niveau d'abstraction, que se trouvent les objets métier.
- Le sous-ensemble des interfaces DAO contient les interfaces des DAO (Data Access Objects), qui spécifient les services destinés à assurer la persistance des objets métier.
- Le sous-ensemble des implémentations DAO contient les implémentations des interfaces précédentes. Dans le cadre de l'étude de cas, nous fournissons une implémentation réalisée avec Hibernate.

L'utilisation des DAO est ainsi conforme au principe de Spring visant à séparer les interfaces des implémentations. Bien entendu, les DAO de Tudu Lists reposent sur Spring DAO et Spring ORM.

La couche service

La couche service contient les services permettant de manipuler les objets métier du modèle. Une nouvelle fois, les interfaces et leurs implémentations sont clairement séparées.

Les services sont au nombre de quatre :

- configuration de Tudu Lists ;
- gestion des utilisateurs ;
- gestion des listes de todos ;
- gestion des todos.

La couche sécurité

La couche sécurité est assurée par le framework Acegi Security for Spring. Pour Tudu Lists, cette couche se résume à fournir une implémentation d'un DAO permettant à Acegi Security d'accéder aux informations concernant les utilisateurs et de paramétrer le framework *via* un fichier de configuration Spring.

La couche Web

La couche Web est constituée de plusieurs sous-ensembles :

- Le sous-ensemble des filtres contient, comme son nom l'indique, les filtres (au sens J2EE du terme) utilisés pour analyser les requêtes HTTP. Tudu Lists possède les filtres suivants : filtre pour JAMon, filtre « session-in-view » et filtre Acegi Security.

- Les sous-ensembles des Actions Struts et des formulaires Struts contiennent les composants Web nécessaires au framework Struts pour gérer l'IHM de Tudu Lists.
- Les sous-ensembles des services DWR des interfaces des objets distants et des implémentations des objets distants contiennent respectivement les services AJAX et les interfaces et implémentations des objets métier disponibles pour ces services.

Installation de l'environnement de développement

Pour les besoins de l'étude de cas, nous avons limité au maximum les logiciels nécessaires à son fonctionnement.

Tudu Lists a simplement besoin des suivants :

- Java 5
- Eclipse 3.1+, avec WTP 1.0.1+ (Web Tools Platform)
- Tomcat 5.5+

Nous fournissons en annexe les instructions nécessaires à leur installation.

Nous supposons dans la suite de cette section que ces outils sont correctement installés sur le poste de travail.

Installation de l'étude de cas

L'étude de cas se présente sous la forme de cinq fichiers Zip, chacun contenant un projet Eclipse complet et indépendant des autres :

- **Tudu.zip :** ce fichier contient le projet qui est utilisé pour la majorité des chapitres de cet ouvrage. Il s'agit d'une version légèrement adaptée de Tudu Lists.
- **Tudu-SpringMVC.zip :** ce fichier contient le projet utilisé par les chapitres 7, 13 et 16. Il s'agit d'une version de Tudu Lists utilisant le framework Spring MVC au lieu de Struts.
- **Tudu-WebFlow.zip :** ce fichier contient le projet utilisé par le chapitre 8. Il s'agit d'une version de Tudu Lists utilisant Spring Web Flow pour gérer sa couche présentation.
- **Tudu-Portlet.zip :** ce fichier contient le projet utilisé au chapitre 10. Il s'agit d'une version de Tudu Lists transformée en portlet utilisable au sein d'un portail J2EE.
- **Tudu-Client.zip :** ce fichier contient le projet utilisé au chapitre 14. Il s'agit d'une application Java Swing capable de communiquer avec l'application Web Tudu Lists en utilisant un Web Service.

Il est possible de télécharger ces fichiers à partir soit de la page Web dédiée à l'ouvrage du site des éditions Eyrolles *(http://www.editions-eyrolles.com).*

L'installation des quatre premiers projets au sein d'Eclipse suit la même procédure quel que soit le fichier. Nous décrivons ci-après l'installation du projet contenu dans le fichier **Tudu.zip :**

1. Une fois le fichier téléchargé, lancer Eclipse, puis créer un projet Web Dynamique par le biais du menu File/Project… en sélectionnant l'option Dynamic Web Project dans le dossier Web.
2. Dans la boîte de dialogue de création de projet, définir le nom du projet (Tudu en respectant la majuscule) puis le serveur d'applications cible (en l'occurrence Tomcat) en cliquant sur le bouton New. Le serveur d'applications doit impérativement être défini à cette étape, faute de quoi le projet ne se compilera pas. Il est inutile de cocher la case Add Project to an EAR.
3. Dans la boîte de dialogue permettant de spécifier les noms de répertoires du projet, changer **src** en **JavaSource.**
4. Importer le fichier Zip *via* les menus File et Import.
5. Dans la fenêtre d'importation, sélectionner le type Archive File, et cliquer sur le bouton Next.
6. Dans la fenêtre de sélection, cliquer sur le bouton Browse en haut de la fenêtre afin de sélectionner le fichier Zip sur le disque dur.
7. Cliquer sur le bouton Finish.
8. Pour recompiler le projet, sélectionner Project et Clean, puis choisir le projet Tudu et cocher la case Start a build immediately.
9. Si des problèmes de compilation apparaissent, s'assurer que le projet utilise bien le compilateur Java 5 (les propriétés du projet sont accessibles *via* son menu contextuel).
10. Pour terminer, exporter les fichiers **hsqldb-1.8.0.1.jar** et **jta.jar** depuis la racine du projet dans le répertoire **common/lib** du répertoire d'installation de Tomcat. Pour cela, sélectionner les fichiers dans la vue et utiliser le menu contextuel Export…

Une fois l'opération terminée, il est possible de configurer et d'exécuter chaque projet, comme expliqué à la section suivante.

Pour Tudu-Client, la procédure est plus simple. Il suffit d'utiliser le menu File/Import…, de sélectionner Existing Projects into Workspace et de cliquer sur Next. Dans la boîte de dialogue d'import, il faut ensuite sélectionner l'option Select archive file puis cliquer sur Browse… pour choisir le fichier **Tudu-Client.zip.** Pour achever, l'importation du projet, cliquer sur Finish.

Configuration et exécution de l'application

Une fois l'application Tudu Lists installée, il faut configurer la connexion à la base de données puis tester son bon fonctionnement. Le projet Tudu-Client ne contenant pas l'application Tudu Lists mais une simple application Swing, il n'est pas concerné par ce qui suit.

La base de données de Tudu Lists est installée dans le répertoire **db** de chaque projet Eclipse :

1. Pour connaître le chemin physique de ce répertoire sur le disque dur, sélectionner par clic droit le dossier **db,** et cliquer sur Properties dans le menu contextuel.
2. Dans la fenêtre qui s'affiche, le chemin physique est fourni sous le nom Location. Mémoriser ce chemin, car il sera nécessaire pour configurer la connexion à la base.
3. Fermer la fenêtre.
4. Pour configurer la connexion, éditer le fichier **context.xml,** qui se trouve dans le répertoire **WebContent/META-INF** du projet. Ce fichier se présente de la manière suivante :

```
<?xml version='1.0' encoding='utf-8'?>
<Context>
<Resource auth="Container" description="Tudu database"
    driverClassName="org.hsqldb.jdbcDriver"
    maxActive="100" maxIdle="30"
    maxWait="10000" name="jdbc/tudu" password=""
    type="javax.sql.DataSource"
    url="jdbc:hsqldb:C:\Eclipse\workspace\tudu\db\tudu"
    username="sa"
/>
</Context>
```

5. Remplacer le texte en gras par le chemin physique de la **db** en veillant à ne pas supprimer le tudu final.

Nous pouvons maintenant exécuter Tudu Lists dans Tomcat (les étapes qui suivent ne sont pas valables pour le projet Tudu-Portlet, qui doit s'exécuter dans un portail ; la procédure dépendant du portail utilisé, nous ne la décrivons pas) :

1. Pour lancer le projet Tudu-SpringMVC, il est nécessaire de lancer préalablement le script **build.xml,** qui se trouve à la racine du projet. Pour cela, il suffit de le sélectionner dans la vue Package Explorer et d'utiliser le menu contextuel Run As/Ant Build. Ce script lance le logiciel ActiveMQ, nécessaire spécifiquement pour ce projet.
2. Pour lancer Tomcat, accéder au menu contextuel du projet par clic droit sur son dossier dans la vue Package Explorer.
3. Dans ce menu, sélectionner Run As et Run on Server puis Tomcat dans la liste des serveurs disponibles et cliquer sur Next.
4. Il convient de s'assurer que le projet se trouve bien dans la zone Configured Projects en utilisant au besoin le bouton Add et en cliquant sur Finish.

Tomcat s'exécute, et une fenêtre de navigation Web s'affiche dans Eclipse avec la page d'accueil de Tudu Lists *(voir figure 1.2)*. Dans le cas du projet Spring-WebFlow, l'interface présentée a été fortement simplifiée pour les besoins du chapitre 8.

Nous pouvons maintenant nous connecter à Tudu Lists en utilisant l'identifiant **test** et le mot de passe **test.**

Organisation des projets dans Eclipse

La figure 1.7 illustre l'organisation des projets (à l'exception de Tudu-Client) telle qu'elle doit apparaître dans l'environnement Eclipse.

Figure 1.7
Organisation des projets Tudu Lists dans Eclipse

Le répertoire **JavaSource** contient l'ensemble du code de Tudu Lists. Il comprend aussi les fichiers de configuration d'Hibernate et de Log4J.

Le répertoire **Tests** contient l'ensemble des tests unitaires de Tudu Lists. Le répertoire **db** contient pour sa part la base de données de l'application.

Le répertoire **WebContent** dispose de plusieurs sous-répertoires :

- **css** contient la feuille de style utilisée par Tudu Lists.
- **images** contient l'ensemble des images du projet.
- **META-INF** contient le fichier de paramétrage de la connexion à la base de données.
- **scripts** contient les fichiers JavaScript nécessaires à l'IHM.
- **WEB-INF** contient les fichiers de configuration de l'application Web, ainsi que les pages et fragments JSP (respectivement dans les sous-répertoires **jsp** et **jspf**) et les taglibs (sous-répertoire **tld**).

Conclusion

Ce chapitre a introduit les grands principes de fonctionnement de Spring et décrit ses apports fondamentaux au développement d'applications Java/J2EE, que ces dernières utilisent ou non des frameworks tiers. Il a aussi présenté l'étude de cas Tudu Lists, qui guidera le lecteur tout au long de sa progression dans l'ouvrage.

Au cours des deux chapitres suivants, nous plongeons au cœur de Spring afin de mettre en valeur les notions essentielles de conteneur léger et de POA, qui en forment l'ossature.

Le reste de l'ouvrage est consacré aux principales intégrations de frameworks tiers proposées par Spring et aux outils permettant d'accélérer les développements avec Spring.