

Avant-propos

Lorsque les standards n'existent pas ou qu'ils tardent à proposer une solution universelle viable, certains outils, frameworks et projets Open Source deviennent des standards de fait. Ce fut le cas de Struts, d'Ant ou encore de Log4J, et c'est le cas d'Hibernate aujourd'hui. L'originalité d'Hibernate en la matière est cependant de s'imposer alors même qu'existent deux standards, EJB 2.0 et JDO 1.0.

Deux ans après son lancement, le succès d'Hibernate est tel qu'il inspire désormais la spécification de persistance des EJB 3.0. Après plusieurs années d'attente, J2EE se dote enfin de la brique critique et indispensable qui lui manquait jusqu'à présent. Ce mécanisme de persistance pourra parfaitement s'utiliser en dehors d'un serveur d'applications, ce qu'Hibernate propose depuis toujours.

Pourquoi insister sur EJB 3.0 alors que cet ouvrage est dédié à Hibernate ? Les entreprises ont un besoin crucial d'outils pérennes. Il est donc légitime d'informer qu'Hibernate, comme TopLink (solution commercialisée par Oracle) et d'autres, a participé à la spécification du futur standard de persistance et qu'Hibernate sera l'une des premières implémentations de la spécification EJB 3.0. En attendant, les entreprises doivent faire un choix susceptible de leur permettre de migrer facilement vers le nouveau standard. Dans cette optique, le meilleur choix est Hibernate.

Hibernate facilite la persistance des applications d'entreprise, laquelle peut représenter jusqu'à 30 % des coûts de développement des applications écrites en JDBC, sans même parler des phases de maintenance. Une bonne maîtrise d'Hibernate accompagnée d'une méthodologie adéquate et d'un outillage ciblé sur vos besoins peuvent vous faire économiser de 75 à 95 % de ces charges.

Au-delà des coûts, les autres apports d'Hibernate sont la qualité des développements, grâce à des mécanismes éprouvés, et le raccourcissement des délais, tout un outillage associé facilitant l'écriture comme la génération du code.

Objectifs de l'ouvrage

La gratuité d'un outil tel qu'Hibernate ne doit pas faire illusion, car il est nécessaire d'investir dans son apprentissage puis son expertise, seuls gages de réussite de vos

projets. Pour un développeur moyennement expérimenté, la courbe d'apprentissage d'Hibernate est généralement estimée à six mois de pratique et d'étude pour en maîtriser les 80 % de fonctionnalités les plus utilisées.

Il faut donc donner les moyens aux développeurs d'apprendre plus vite, de manière plus concrète et avec un maximum d'exemples de code. Tel est l'objectif principal de cet ouvrage.

Hibernate in Action, l'ouvrage de Christian Bauer, le créateur d'Hibernate, coécrit avec Gavin King, est indispensable pour appréhender toute la problématique de la persistance dans les applications d'entreprise, et il est recommandé de le lire. Moins théorique et résolument tourné vers la pratique, le présent ouvrage se propose d'illustrer chacune des fonctionnalités de l'outil par un ou plusieurs exemples concrets.

Questions-réponses

Cet ouvrage porte-t-il sur les EJB3.0 ?

Oui et non. Non, car la spécification n'est pas finalisée. Oui, car l'équipe d'Hibernate est très réactive quant à l'implémentation des spécifications du futur standard de persistance Java. Une fois que les spécifications seront finalisées, il n'y a aucun doute qu'Hibernate sera l'une des premières implémentations disponibles des EJB 3.0.

Cet ouvrage traite-t-il d'Hibernate 2 ?

Hibernate 3 n'est pas une refonte d'Hibernate 2. Il s'agit d'une version qui propose beaucoup de nouveautés, mais dont le cœur des fonctionnalités reste inchangé par rapport à Hibernate 2. L'ouvrage donne des repères aux utilisateurs d'Hibernate 2, qui y trouveront matière à améliorer leur utilisation de l'outil.

Où peut-on trouver les exemples de code ?

Les exemples de code sont disponibles sur la page dédiée à l'ouvrage sur le site Web d'Eyrolles, à l'adresse www.editions-eyrolles.com. Ils ont été conçus comme des tests unitaires afin que vous puissiez les exécuter facilement et y insérer des assertions.

Comment devenir contributeur du projet Hibernate ?

Il n'y a rien de particulier à faire. Hibernate est le fruit d'une interaction intense entre les utilisateurs, les contributeurs et l'équipe Hibernate.

Si vous êtes motivé pour participer à l'évolution d'Hibernate, plusieurs axes peuvent vous intéresser, notamment les suivants : développement de nouvelles fonctionnalités (généralement réservé aux développeurs expérimentés), évolution des outils ou des annotations, documentation, etc.

Les traducteurs sont également les bienvenus pour fournir à la communauté une version française à jour du guide de référence.

Organisation de l'ouvrage

La structure de cet ouvrage a parfois été un casse-tête. Il a fallu jongler dès le début entre la configuration de la persistance *via* les fichiers de mapping et l'utilisation à proprement parler des API d'Hibernate, le tout sans répéter le guide de référence de l'outil, qui est sans doute le plus complet du monde Open Source.

Le premier chapitre propose un historique et un état des lieux de la persistance dans le monde Java ainsi que des solutions actuellement disponibles sur le marché. Il présente un exemple très simple d'utilisation d'Hibernate.

Le chapitre 2 décrit le raisonnement à adopter lorsque vous utilisez un outil tel qu'Hibernate. Le vocabulaire est posé dès ce chapitre, qui montre également comment installer Hibernate.

Le chapitre 3 vous apprendra à écrire vos fichiers de mapping et propose un référentiel des métadonnées.

Dès le chapitre 4, il vous faudra avoir maîtrisé les notions abordées dans les trois premiers chapitres. À ce stade de l'ouvrage, vous commencez à entrer dans les fonctionnalités avancées d'Hibernate. Dans ce chapitre, vous découvrirez certains principes avancés de modélisation et les indications indispensables pour mapper vos choix de modélisation.

Le chapitre 5 est dédié aux techniques de récupération d'objets. Vous verrez qu'il existe plusieurs méthodes pour interroger le système de stockage de vos objets (la base de données relationnelle).

Le chapitre 6 décrit en détail comment considérer la création, la modification et la suppression des objets gérés par Hibernate. Vous y apprendrez comment prendre en compte la concurrence dans vos applications et aborderez la notion de persistance transitive.

Le chapitre 7 présente les techniques les plus répandues pour gérer une session Hibernate. Il propose plusieurs best practices permettant de mettre en œuvre une gestion simple et optimale de la session Hibernate ainsi qu'un aparté sur l'utilisation conjointe de Struts et d'Hibernate.

Le chapitre 8 introduit plusieurs nouveautés d'Hibernate 3 et revient sur certaines fonctionnalités très poussées des versions précédentes.

Le chapitre 9 se penche sur l'outillage disponible autour d'Hibernate ainsi que sur la configuration de pools de connexions et de caches de second niveau.

À qui s'adresse l'ouvrage ?

Cet ouvrage est destiné en priorité aux développeurs d'applications Java devant mettre en place ou exploiter un modèle de classes métier orienté objet. Hibernate excelle lorsque la phase de conception objet du projet est complète. Les concepteurs pourront constater que

l'outil ne les bride pas dans leur modélisation. Si l'accent est mis sur la modélisation de la base de données plutôt que sur le diagramme de classes, Hibernate sait néanmoins s'adapter aux vues des multiples fonctionnalités de mapping.

Les chefs de projet techniques, les décideurs et les concepteurs y trouveront donc aussi des éléments primordiaux pour la conception, la mise en place de l'organisation et l'optimisation de projets fondés sur un modèle métier orienté objet.

1

Persistance et mapping objet-relationnel

Ce chapitre introduit les grands principes du mapping objet-relationnel et plus généralement de la persistance dans le monde Java.

La persistance est la notion qui traite de l'écriture de données sur un support informatique. Pour sa part, le mapping objet-relationnel désigne l'interaction transparente entre le cœur d'une application, modélisé en conception orientée objet, et une base de données relationnelles.

Afin de bien comprendre l'importance qu'a pris Hibernate dans le marché de la persistance Java, nous commencerons par dresser un rapide historique de cette dernière. Nous proposerons ensuite un panorama des outils de persistance et indiquerons d'autres solutions permettant de gérer la persistance dans vos applications.

Historique de la persistance en Java

L'accès simple aux données et la persistance des données n'ont jamais vraiment posé problème dans le monde Java, JDBC ayant vite couvert les besoins des applications écrites en Java. Cependant, Java a pour objectif la réalisation d'applications dont la modélisation des problématiques métier est orientée objet. On ne parle donc plus, pour ces applications, de persistance de données mais de persistance d'objets.

La persistance d'objets en Java n'a été qu'une suite d'échecs et de déceptions. Avant d'aboutir à des solutions telles qu'Hibernate, le monde Java a dû subir la lourdeur de

plusieurs solutions. Il est important de revenir sur ce passé pour bien comprendre ce qui se joue dans le choix d'un framework de persistance.

Les EJB (Enterprise JavaBeans)

Le souvenir le plus traumatisant sur ce thème sensible de la persistance dans les applications orientées objet reste sans aucun doute la première version des EJB (Enterprise JavaBeans), il y a sept ans.

Il existait peu de frameworks de persistance à cette époque, et les entreprises se débrouillaient avec JDBC. Les applications étaient souvent orientées selon un modèle tabulaire et une logique purement relationnelle plutôt qu'objet.

Les grandes firmes du monde Java ont fait un tel forcing marketing autour des EJB que les industries ont massivement adopté cette nouvelle technologie.

Les EJB se présentent alors comme le premier service complet de persistance. Ce service consiste en la gestion de la persistance par conteneur, ou CMP (Container-Managed Persistence). Bien que personne à l'époque ne parvienne réellement à faire fonctionner CMP, l'engouement pour cette technologie est tel que les développeurs la choisissent ne serait-ce que pour l'ajouter à leur CV.

Techniquement, CMP se révèle incapable de gérer les relations entre entités. De plus, les développeurs sont contraints d'utiliser les lourdes interfaces distantes (remote). Certains développeurs en viennent à implémenter leur propre système de persistance géré par les Beans, ou BMP (Bean-Managed Persistence). Déjà décrié pour sa laideur, ce pattern n'empêche cependant nullement de subir toute la lourdeur des spécifications imposées par les EJB.

TopLink et JDO

À la fin des années 90, aucun framework de persistance n'émerge. Pour répondre aux besoins des utilisateurs des EJB, TopLink, un mappeur objet-relationnel propriétaire de WebGain, commence à se frayer un chemin.

TopLink

Solution propriétaire éprouvée de mapping objet-relationnel offrant de nombreuses fonctionnalités, TopLink comporte la même macro-architecture qu'Hibernate. L'outil a changé deux fois de propriétaire, WebGain puis Oracle. Le serveur d'applications d'Oracle s'appuie sur TopLink pour la persistance. Hibernate et TopLink seront les deux premières implémentations des EJB 3.0. <http://otn.oracle.com/products/ias/toplink/content.html>

TopLink a pour principaux avantages la puissance relationnelle et davantage de flexibilité et d'efficacité que les EJB, mais au prix d'une relative complexité de mise en œuvre. Le

problème de TopLink est qu'il s'agit d'une solution propriétaire et payante, alors que le monde Java attend une norme de persistance transparente, libre et unique. Cette norme universelle voit le jour en 1999 sous le nom de JDO (Java Data Object).

En décalage avec les préoccupations des développeurs, le mapping objet relationnel n'est pas la préoccupation première de JDO. JDO fait abstraction du support de stockage des données. Les bases de données relationnelles ne sont qu'une possibilité parmi d'autres, aux côtés des bases objet, XML, etc. Cette abstraction s'accompagne d'une nouvelle logique d'interrogation, résolument orientée objet mais aussi très éloignée du SQL, alors même que la maîtrise de ce langage est une compétence qu'une grande partie des développeurs ont acquise. D'où l'autre reproche fait à JDO, le langage d'interrogation JDOQL (JDO Query Language) se révélant à la fois peu efficace et très complexe.

En 2002, après trois ans de travaux, la première version des spécifications JDO connaît un échec relatif. Jugeant la spécification incomplète, aucun des leaders du marché des serveurs d'applications ne l'adopte, même si TopLink propose pour la forme dans ses API une compatibilité partielle avec JDO.

Du côté des EJB, les déceptions des clients sont telles qu'on commence à remettre en cause la spécification. La version 2.0 vient à point pour proposer des remèdes, comme les interfaces locales ou la gestion des relations entre entités. On parle alors de certains succès avec des applications développées à partir d'EJB CMP 2.0. Ces quelques améliorations ne suffisent toutefois pas à gommer la mauvaise réputation des EJB, qui restent trop intrusifs (les entités doivent toujours implémenter des interfaces spécifiques) et qui brident la modélisation des applications en ne supportant ni l'héritage, ni le threading. À ces limitations s'ajoutent de nombreuses difficultés, comme celles de déployer et de tester facilement les applications ou d'utiliser les classes en dehors d'un conteneur (serveur d'applications). L'année 2003 est témoin que les promesses des leaders du marché J2EE ne seront pas tenues.

Début 2004, la persistance dans le monde Java est donc un problème non résolu. Les deux tentatives de spécification ont échoué. EJB 1.x est un cauchemar difficile à oublier, et JDO 1.x un essai manqué. Quant à EJB 2.0, si elle résout quelques problèmes, elle hérite de faiblesses trop importantes pour s'imposer.

Hibernate

Le 19 janvier 2002, Gavin King fait une modeste publication sur le site [theserverside.com](http://www.theserverside.com/discussions/thread.tss?thread_id=11367) pour annoncer la création d'Hibernate (http://www.theserverside.com/discussions/thread.tss?thread_id=11367).

Hibernate est lancé sous le numéro de version 0.9. Depuis lors, il ne cesse d'attirer les utilisateurs, qui forment une réelle communauté. Le succès étant au rendez-vous, Gavin King gagne en popularité et devient un personnage incontournable dans le monde de la persistance Java. Coécrit avec Christian Bauer, l'ouvrage *Hibernate in Action* sort l'année suivante et décrit avec précision toutes les problématiques du mapping objet-relationnel.