Article Adobe

O'AIR'M

Shoun Ichida – Tegnane Ly – Vincent Nguyen-Huu

Version 1.0

13 Pages

01/07/2009

Adobe ORM Flex Air MTI EPITA

Multimédia
Technologies de l'Information

ÉCOLE D'INGÉNIEURS EN INFORMATIQUE

Propriétés du document

Auteur	Shoun Ichida – Tegnane Ly – Vincent Nguyen-Huu
Version	1.0
Nombre de pages	13
Références	Adobe ORM Flex Air MTI EPITA

Historique du document

Date de révision	Version	Auteur	Changements
01/07/2009	1.0	SIC	Rédaction de l'article

Source du document

La dernière version de ce document est téléchargeable surhttp://mti.epita.net/blogs/

Sommaire

Introduction

L'Object RelationalMapping ou ORM est un concept fréquemment utilisé dans les langages qui interagissent avec une base de données. Les ORM ne sont utilisés qu'avec les langages orientés objet puisqu'il s'agit de traduire les données contenues dans la base en objet, ou inversement, puis de pouvoir les utiliser dans le programme.

Les ORM sont développés pour les développeurs afin qu'ils puissent utiliser un outil simple de manipulation de données. Ceci crée une base de données d'objets virtuels qui sera utilisée dans le langage de programmation. Nous pouvons citer par exemple Hibernate qui est le célèbre ORM pour le langage JAVA fonctionnant avec de nombreux types de base de données (MySQL, PostGRESQL, HSQLDB, etc.).

O'AIR'M est un ORM pour le FrameworkFlex sur la plateforme AIR. Ces technologies Adobe ne proposent pas dans leur version standard un tel outil. L'entreprise Cegedim a donc proposé à l'école EPITA (École Pour l'Informatique et les Techniques Avancées) un sujet visant à concevoir cette bibliothèque. Ce projet entrait dans le cadre des PFEE (projet de fin d'études en entreprise) de la spécialisation MTI, promotion 2009.

Le Framework que nous allons décrire dans cet article ne s'utilise exclusivement qu'avec une base de donnéesSQLite (http://www.sqlite.org/) puisque la plateforme Air ne permet de manipuler que ce type de SGBD. Vous trouverez tous les documents relatifs à ce projet sur le site du PFEE http://0217021.free.fr/oairm/ ainsi que les interviews de l'équipe, des démonstrations du Framework et une documentation en ligne de la bibliothèque.

Présentation

L'équipe projet

Le projet a été réalisé par une équipe de 3 étudiants de la spécialisation MTI, promotion 2009. Nous pouvons citer :

- Bruno Malaquin (Responsable du projet)
- Axel Berardino
- Adrien Revol

Ce projet, dans le cadre des PFEE, a été développé durant leur dernière année du cycle ingénierie de l'EPITA.Il a débuté en juin de l'année 2008 pour se terminer en décembre de la même année.

Cegedim

Cegedim(http://www.cegedim.fr/)est l'entreprise qui a proposé le sujet du PFEE à EPITA. Elle a été fondée en 1969 par Jean-Claude Labrune, actuel président directeur général du groupe, en collaboration avec un groupe de laboratoires pharmaceutiques désireux de mettre en commun des ressources informatiques et des savoir-faire dans les domaines de la recherche documentaire.À ce jour, Cegedimconçoit des bases de données exclusives et des solutions logicielles à forte valeur ajoutée.

Ses compétences s'exercent dans 4 secteurs :

- **CRM et données stratégiques :** regroupe les services dédiés aux laboratoires pharmaceutiques.
- **Professionnels de santé :** dédié aux médecins et aux pharmaciens.
- **Assurances et flux de santé :** dédié aux acteurs de l'assurance santé.
- **Technologies et services :** s'adresse aux entreprises de tout secteur.

Le projet

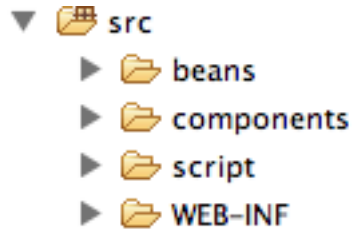
Cegedim voulant étendre ses compétences vers les nouvelles technologies Adobe, le projet O'AIR'M devait consister en l'élaboration d'un Framework de liaisons relationnelles entre les objets et la base de données pour la plateforme ADOBE AIR. Le but étant de créer un système qui permettrait de s'abstraire de la couche d'accès aux données afin d'avoir un code plus clair, stable, portable et concis.

La suite de cet article décrit la partie technique du projet, à savoir la configuration et l'utilisation du Framework.

O 'AIR'M

Sur le même modèle que le Framework Hibernate pour le Java, O'AIR'M tend à respecter une certaine arborescence. Celle que nous allons présenter n'est qu'un exemple qui se dit « propre » mais qui est modifiable.

Dans cette partie nous allons décrire comment utiliser ce Framework. Pour cela l'arborescence du dossier source est le suivant :



Configuration

La configuration du Framework se fait dans un fichier situé dans le dossier source (dans notre cas le répertoire « src »). Dans ce fichier, oairm.cfg.xml, nous spécifions tout ce qui concerne la base de données ainsi que tout ce qui concerne les objets qui vont être persistés en base.

Voici la structure générale de ce fichier accompagné d'explications :

En-tête de fichier xml avec spécification de la dtd à charger

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE oairm-configuration SYSTEM "lib/oairm/OAIRM_CONFIGURATION_DTD.dtd">
```

Balise principale

```
<oairm-configuration>
[...]
</oairm-configuration>
```

Dans cette section se trouveront toutes les balises de configuration que nous allons détailler ci-après.

Fichier de stockage de la base de donnéesSQLite

```
<db_file
	create_db_file="false"
	create_schema="false"
	erase_schema="false" >
path/to/the/file.db
</db_file>
```

Dans cette section nous indiquons le fichier de base de données où seront stockées les données. Nous pouvons spécifier la configuration par les attributs suivants :

- create_db_file : spécifie si le fichier contenant les données doit être créé s'il n'existe pas (valeur : true/false)
- create_schema : spécifie si les tables doivent être créées au début de l'exécution (valeur : true/false)
- erase_schema : spécifie si les tables doivent être supprimées à la fin de l'exécution (valeur : true/false)

Chiffrement

```
<encryptionenabled="false" type="AES"/>
```

Cette balise permet d'activer ou non le chiffrement des données stockées en base et de spécifier le type du chiffrement.Dans cet exemple le chiffrement est désactivé (l'attribut enable est à « false »).

Dans la prochaine version de Flex (4.0), le chiffrement étant nativement géré, cette option sera dépréciée.

Transactions

```
<database
	synchrone="on"
	saving_mode="explicit"
	saving_period="900"
	cache="on">
</database>
```

Les transactions avec la base de données doivent être spécifiées. Voici la signification des différents attributs :

- synchrone : lorsque sa valeur est à « on » toutes les transactions sont synchrones (non événementielles). À « off » les transactions sont asynchrones (événementielles) et on peut notifier que les actions se sont déroulées correctement en appelant la méthode de « callback ». Le mode conseillé reste le synchrone.

- saving_mode :
  - « explicite » la sauvegarde des objets en base s'effectue que lorsque le développeur appelle explicitement la méthode "save" sur l'objet
  - « immédiate » à chaque modification d'un des champs de l'objet, on sauvegarde en base
  - « period » toutes les N secondes (déterminé par saving_period), on sauvegarde tous les objets modifiés
  - « light » version soft du « immédiate », un peu plus "intelligent", qui le fait pour un certain nombre de modifications de champs avant de sauvegarder l'ensemble de l'objet
- saving_period : correspond à la durée avant que le cache n'expire.
- cache : lorsque sa valeur est à « on » le cache du Framework est activé. Le cache est utilisé pour les transactions afin qu'il se « souvienne » des objets à persister en base de données lors du commit.

Historique

```
<logginglevel="severe" init_clear="on">path/to/log.txt</logging>
```

Cette balise sert à configurer les log des transactions à la base de données et indiquer le fichier qui contiendra tous les logs. L'attribut « level » indique le niveau de log (peut avoir les valeurs : 0 | 1 | 2 | 3 | 4 | 5 | nothing | severe | sql | warning | notice | all). Quant à « init_clear », il indique si le fichier de log doit être vidé au démarrage de l'application.

Persistence

```
<po_list>
        <po>beans.ObjectOne</po>
        <po>beans.ObjectTwo</po>
        <po>beans.ObjectThree</po>
</po_list>
```

À l'intérieur de ces balises nous indiquons tous les objets qui vont être persistés (po signifie « persistedobject », objet persisté). Vous devez spécifier le nom du package suivi du nom de la classe à partir du dossier source (dans notre cas le package « beans » est dans le dossier source et contient tous les objets qui vont être persistés).

Vous pouvez donc constater que la configuration du Framework est assez intuitive et ne nécessite pas beaucoup d'efforts. Une fois configuré, il ne reste plus qu'à l'utiliser.

Afin d'utiliser correctement le Framework nous allons dans un premier temps finir la configuration en créant les différents « beans ». Dans le fichier de configuration nous avons indiqué les objets à charger. Ces objets doivent être paramétrés de sorte à être persistés correctement.

Beans

Afin de paramétrer chacun des objets nous allons utiliser les annotations. Pour commencer, pour déclarer une classe il faut l'associer à une table de la base. Exemple :

```
[Table(name="table1")]
[LazyLoading(value="true")]
publicclassObjectOne
{ ... }
```

L'annotation « [Table(name="table1")] » spécifie la table en base avec laquelle la classe va être liée et « [LazyLoading(value="true")] » indique le mode de récupération des objets (lazy ou non).

Ensuite, chaque table possède une clé primaire : un id.

```
[Id]
[Column(name="id", autoincrement="true")]
publicvar id : Number;
```

L'annotation « [Id] » indique qu'il s'agit de la clé primaire. « [Column(name="id ", autoincrement="true")] » spéficifie à quelle colonne l'attribut correspond ; nous précisons alors le nom de la colonne ainsi que la gestion par le SGBD ou non de l'incrémentation des id.

Enfin, pour chaque colonne de la table nous devons spécifier un attribut qui lui correspond :

```
[Column(name="name")]
publicvarname : String;
```

Relations entre tables

Toute base de données possède des relations entre ses tables. Voici un descriptif de celles prises en charge par le Framework.

- One-to-one
  - L'annotation « [OneToOne] » mise au dessus d'un attribut indique que celui-ci sera référencé dans une autre table par une relation one-to-one.

```
[OneToOne]
[Column(name="president_fk")]
publicvarpresident : President;
```

  - Lorsque l'annotation est spécifiée par « mappedBy » l'attribut sera sauvegardé sans action supplémentaire. Dans ce cas là on n'indique pas la colonne.

```
// L'attribut appartient à la class Customer lié à la table « Customer »
[OneToOne(mappedBy="customer")]
publicvarpassport : Passport;
```

- One-to-many&Many-to-one
  - Cette annotation sert pour lemappingone-to-many couplé au mappingmany-to-one. Nous aurons d'un côté :

```
[OneToMany(name="streets", referredColumn="city_fk")]
publicvarstreets : ArrayCollection;
```

Puis pour satisfaire la relation nous aurons dans une autre classe:

```
[ManyToOne(mappedBy="streets")]
publicvar city : JoinCity;
```

- Many-to-many&JoinTable
  - Pour cette relation qui lie deux colonnes de tables différentes nous avons d'un côté :

```
[ManyToMany(mappedBy="employees",name=" Employer")]
publicvaremployers : ArrayCollection;
```

Et de l'autre :

```
[ManyToMany(name="employees")]
[JoinTable(name="EMPLOYER_EMPLOYEE", joinColumn="emper_id", inverseJoinColumn="empee_id")]
publicfunctiongetemployees () : ArrayCollection
{
    return _employees;
}
```

  - « JoinTable » sert à spécifier la table d'association ainsi que les clés qui serviront pour associer deux objets entre eux.

Utilisation

Maintenant que tout est en place pour utiliser le Framework voici un exemple.

Ajout en base

```
varobject : ObjectOne = newObjectOne(); // Instanciation de l'objet

// Initialisation des attributs
object.name = ''test'';
object.value = parseInt(''42'');

// Récupération de la variable de sessions
varsess : Session = Session.getInstance();

// Ajout en base
sess.create(object);

// Exécution de la transaction
sess.commit();
```

Dans cet exemple nous créons un object « ObjectOne » que nous initialisons et que nous ajoutons en base.

Mise à jour

Si nous nous inspirons de l'exemple précédent voici une mise à jour d'un objet

```
varobject : ObjectOne = newObjectOne(); // Instanciation de l'objet

// Initialisation des attributs
object.name = ''test'';
object.value = parseInt(''42'');

// Récupération de la variable de sessions
varsess : Session = Session.getInstance();

// Ajout en base
sess.create(object);

// Modification d'un attribut
object.name = ''Objet mis à jour'' ;

//Mise à jour
sess.update(object) ;

// Exécution de la transaction
sess.commit();
```

Ici nous avons mis à jour l'attribut « name » de l'objet.

Suppression

Supposons que nous avons récupéré l'objet que nous venons de créer. Pour le supprimer il suffit de :

```
// Récupération de la session
varsess : Session = Session.getInstance();

// Suppression de l'objet en base
sess.remove(object);

// Exécution de la transaction
sess.commit();
```

Voici donc comment utiliser le Framework O'AIR'M.

Requêtes

Inspiré du langage Java, les requêtes s'effectuent en « Critéria ». Pour cela il suffit de créer un objet Criteria et d'y ajouter les conditions :

```
varcriteria : Criteria = newCriteria(Customer)
                .add(LogicalExpression.and(Expression.eq("login", txtLogin.text),
                                           Expression.eq("pass", txtPass.text)));
varcustomers : ArrayCollection = sess.read(Customer, criteria);
```

Ici nous récupérons les objets « Customer » dont le login et le mot de passe correspondent avec ce qui a été saisi.

Conclusion

Cet article a détaillé la configuration ainsi que l'utilisation du Framework O'AIR'M, un Object RelationalMapping (ORM) pour le Framework Flex sur la plateforme Air, deux technologies d'Adobe. Le processus de mise sous licence « Open Source » est en cours, permettant bientôt l'utilisation de ce Framework.

D'autres ORM pour cette plateforme existent sous différentes licences mais celui-ci étant inspiré d'Hibernate offre à tous les développeurs Java la possibilité de concevoir leurs applications avec un outil proche de ce qu'ils connaissent déjà.

Les différentes options de configuration permettent une implémentation libre et offrent de nombreuses possibilités sur les applications. L'utilisation des « Critéria » permet d'effectuer des requêtes de façon non ambiguë avec un outil familier.

La prise en main ne nécessite que peu de pratique et nous attendons tous avec impatience la disponibilité de ce Framework.