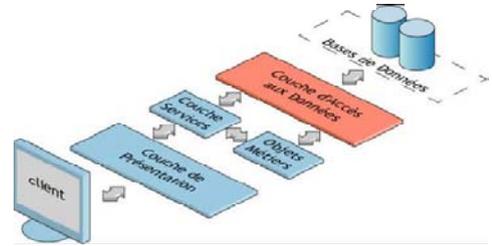


Hibernate

Un outil de *mapping* objet/relationnel pour le monde Java

Yves Bekkers

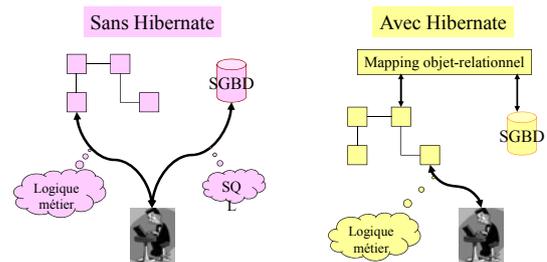
Mapping objet/relationnel O/R mapping



Hibernate

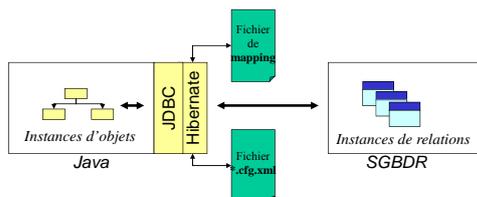
- Base de données + données de config = service de persistance
- Persistance automatisée et transparente d'objets métiers - *classes Java* - vers une bases de données relationnelles
- Description à l'aide de *méta-données* de la *transformation réversible* entre un modèle relationnel et un modèle de classes

Transparence de la persistance



Fichiers de configuration et de mapping

- Faire le lien en Java entre
 - représentation objet des données et
 - représentation relationnelle basée sur un schéma SQL



Configuration d'Hibernate

Description de la connexion

Configurer Hibernate

- Il y a trois manières d'effectuer la configuration d'Hibernate pour une application donnée :
 - Par programme
 - Par un fichier de propriétés
hibernate.properties
 - Par un document xml hibernate.cfg.xml

Hibernate - Y. Bekkers

7

Configurer par programme

- Une instance de `org.hibernate.cfg.Configuration` représente un ensemble de mappings des classes Java d'une application vers la base de données SQL.

```
Configuration cfg = new Configuration()
.addResource("Item.hbm.xml")
.addResource("Bid.hbm.xml");
```

Hibernate - Y. Bekkers

8

Éviter de câbler en dur dans le programme les noms de fichiers

```
Configuration cfg = new Configuration()
.addClass(org.hibernate.auction.Item.class)
.addClass(org.hibernate.auction.Bid.class);
```

Hibernate va rechercher les fichiers de mappings
/org/hibernate/auction/Item.hbm.xml
et
/org/hibernate/auction/Bid.hbm.xml
dans le classpath

Hibernate - Y. Bekkers

9

Préciser des propriétés de configuration par programme

```
Configuration cfg = new Configuration()
.addClass(org.hibernate.auction.Item.class)
.addClass(org.hibernate.auction.Bid.class)
.setProperty("hibernate.dialect",
"org.hibernate.dialect.MySQLInnoDBDialect")
.setProperty("hibernate.connection.datasource",
"java:comp/env/jdbc/test")
.setProperty("hibernate.order_updates",
"true");
```

Hibernate - Y. Bekkers

10

Fichiers de propriétés

- Ce n'est pas le seul moyen de passer des propriétés de configuration à Hibernate. Les différentes options sont :
 1. Passer une instance de `java.util.Properties` à `Configuration.setProperties()`.
 2. Placer `hibernate.properties` dans un répertoire racine du classpath
 3. Positionner les propriétés System en utilisant `java -Dproperty=value`.
 4. Inclure des éléments `<property>` dans le fichier `hibernate.cfg.xml` (voir plus loin).

Hibernate - Y. Bekkers

11

Exemple de document hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    ...
  </session-factory>
</hibernate-configuration>
```

Hibernate - Y. Bekkers

12

hibernate.cfg.xml (1) connection à la base

```
<property name="connection.driver_class">
  org.hsqldb.jdbcDriver
</property>
<property name="connection.url">
  jdbc:hsqldb:film
</property>
<property name="connection.username">sa</property>
<property name="connection.password"></property>
```

Hibernate - Y. Bekkers

13

hibernate.cfg.xml (2) autres propriétés

```
<!-- JDBC connection pool (use the built-in) -->
<property name="connection.pool_size">1</property>
<!-- SQL dialect -->
<property name="dialect">
  org.hibernate.dialect.HSQLDialect
</property>
<!-- Enable Hibernate's automatic session context management -->
<property name="current_session_context_class">thread</property>
<!-- Disable the second-level cache -->
<property name="cache.provider_class">
  org.hibernate.cache.NoCacheProvider
</property>
<!-- Echo all executed SQL to stdout -->
<property name="show_sql">>true</property>
<!-- Drop and re-create the database schema on startup -->
<property name="hbm2ddl.auto">create</property>
```

Hibernate - Y. Bekkers

14

Mapping

Objet ↔ relationnel
Java ↔ SGBD

Hibernate - Y. Bekkers

15

Types Java ↔ Types SQL

- Java ↔ SGBD
 1. Valeurs composées : *entités persistantes*
Instance d'objet Java ↔ ligne dans une table
 2. Valeurs simples :
Type simple Java ↔ Colonne dans une table type SQL
- Transferts automatiques dans les deux sens
 - Via JDBC
 - Réduit le temps de développement
 - Code SQL et JDBC généré automatiquement

Hibernate - Y. Bekkers

16

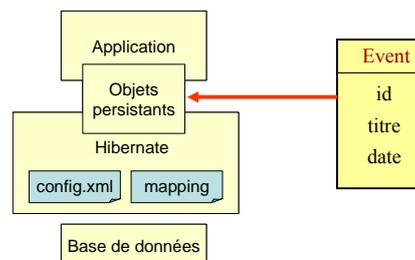
Un premier exemple

Agenda composé d'évènements

Hibernate - Y. Bekkers

17

Conserver des évènements



Hibernate - Y. Bekkers

18

De quoi est composée l'application ?

- Une classe : entité persistante
 - Un Java bean
- Un document de « *mapping* »
 - Associer des propriétés Java à des propriétés d'un SGBD relationnel
- Un document de configuration
 - Informations de connection à la base et références aux *mapping*

 Dans cet exemple pas de SGBD au départ 

Hibernate - Y. Bekkers

19

1 - Entité persistante

Java bean

Hibernate - Y. Bekkers

20

Entité persistante Event

- Java bean Event représente un *évènement*

```
public class Event {  
    private Long id;  
    private String titre;  
    private Date date;  
}  
  
public Event() {  
}  
...  
}
```

Attributs cachés

Constructeur sans argument

← Getter + setter (page suivante)

Event
id
titre
date

Hibernate - Y. Bekkers

21

Getters

```
public Long getId() {  
    return id;  
}  
public Date getDate() {  
    return date;  
}  
public String getTitre() {  
    return titre;  
}
```

Event
id
titre
date

Hibernate - Y. Bekkers

22

Setters

```
private void setId(Long id) {  
    this.id = id;  
}  
public void setDate(Date date) {  
    this.date = date;  
}  
public void setTitre(String titre) {  
    this.titre = titre;  
}
```

Event
id
titre
date

Hibernate - Y. Bekkers

23

Constructeur sans argument

- Le constructeur sans argument est requis pour toute classe persistante :
 - Hibernate crée des instances d'objets en utilisant la réflexion Java et les informations prises la base de données.

Hibernate - Y. Bekkers

24

Identifiant unique d'évènement

- La propriété `id` contient la valeur d'un identifiant unique pour un évènement particulier.
- Toutes les classes d'entités persistantes (ainsi que les classes dépendantes de moindre importance) ont besoin d'une propriété identifiante
- Les identifiants uniques servent à Hibernate et à l'application
- Noter que la méthode `setId()` est privée :
 - l'application n'a pas à modifier ces identifiants
 - On va laisser cette tâche de gestion à Hibernate

Hibernate - Y. Bekkers

25

2 - Mapping O/R

Java ↔ SGBDR

Hibernate - Y. Bekkers

26

Structure d'un document de mapping

```
<hibernate-mapping>
  <class >...</class>
  <class >...</class>
  ...
</hibernate-mapping>
```

Hibernate - Y. Bekkers

27

Fichier de « mapping »

```
<hibernate-mapping>
  <class name="Event" table="EVENTS">
    <id name="id" column="EVENT_ID">
      <generator class="increment"/>
    </id>
    <property name="date" type="timestamp"
      column="EVENT_DATE"/>
    <property name="titre"/>
  </class>
</hibernate-mapping>
```

Event
id
titre
date

Hibernate - Y. Bekkers

28

DTD du dialecte hibernate-mapping

```
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-
mapping-3.0.dtd">
```

Hibernate - Y. Bekkers

29

Élément <class>

- Permet d'associer une classe Java à une table de la base de données
 - Ce sont les classes d'entités persistantes
- Attributs de l'élément <class>
 - Attribut `name` : nom de la classe Java
 - Attribut `table` : nom de la table correspondante

```
<class name="Event" table="EVENTS">
  ...
</class>
```
- Contenu : voir suite

Hibernate - Y. Bekkers

30

Contenu d'un élément <class>

- Identifiant unique
 - Élément <id>
- Propriétés
 - Élément <property>

Hibernate - Y. Bekkers

31

Élément <id> (1)

- Identifiant unique pour les événements

```
<id name="id" column="EVENT_ID">
  <generator class="increment" />
</id>
```
- Attributs de l'élément <id>
 - name : nom de l'attribut dans la classe Java
 - column : (optionnel) nom de la propriété dans la table SQL
 - type : (optionnel) type de *mapping Hibernate* de l'identifiant

Hibernate - Y. Bekkers

32

Élément <id> (2)

- Élément contenu <generator>
 - Attribut `class` spécifie le mode (la stratégie) de gestion de l'identifiant unique
 - Hibernate supporte aussi les identifiants
 - générés par les bases de données,
 - globalement uniques,
 - ainsi que les identifiants assignés par l'application
 - n'importe quelle stratégie que vous avez écrit en extension.

Hibernate - Y. Bekkers

33

Stratégies de génération des identifiants

- Attribut `class` de l'élément <generator>
 - `<generator class="increment" />`
- Valeurs possibles
 - `increment` : Génère des identifiants de type Java `long`, `short` ou `int`
 - `identity`, `sequence` : Génère des identifiants `long`, `short` ou `int`
 - Spécifique aux bases DB2, MySQL, MS SQL Server, Sybase et HypersonicSQL

Hibernate - Y. Bekkers

34

Autres stratégies de génération

- `hilo` : Hibernate génère des identifiants de `long`, `short` ou `int` à partir d'une table dédiée et d'un algorithme performant
- `native` : Génère des identifiants `long`, `short` ou `int`. Choisit `identity`, `sequence` ou `hilo` selon les possibilités offertes par la base de données sous-jacente
- `assigned` : Laisse l'application effectuer elle-même l'affectation avant que la méthode `save()` ne soit appelée

Hibernate - Y. Bekkers

35

Élément <property>

- L'élément <property> déclare une propriété de la classe au sens *JavaBean*.
 - Attributs de l'élément <property>
 - name : nom de la propriété, avec une lettre initiale en minuscule.
 - column : (optionnel - par défaut au nom de la propriété) : le nom de la colonne mappée. Cela peut aussi être indiqué dans le(s) sous-élément(s) <column>
 - type : (optionnel) : nom indiquant le type de *mapping Hibernate*. Ici il faut spécifier un choix parmi `date`, `timestamp` ou `time` car Hibernate ne peut inférer seul ce type
- ```
<property name="date" type="timestamp"
 column="EVENT_DATE" />
```

Hibernate - Y. Bekkers

36

## Fichier de mapping - rappel

```
<class name="Event" table="EVENTS">
 <id name="id" column="EVENT_ID">
 <generator class="increment" />
 </id>
 <property name="date"
 type="timestamp" column="EVENT_DATE" />
 <property name="titre" />
</class>
```

Event
id
titre
date

Hibernate - Y. Bekkers

37

## Attribut column par défaut

```
<property name="date"
 type="timestamp"
 column="EVENT_DATE" />
<property name="titre" />
```

- Le nom de la colonne est le même que le nom de la propriété Java

Hibernate - Y. Bekkers

38

## Attribut type par défaut

```
<property name="date"
 type="timestamp"
 column="EVENT_DATE" />
<property name="titre" />
```

- Les types ne sont ni des types Java ni des types SQL, ce sont des types Hibernate
- détection automatique (utilisant la réflexion sur la classe Java)

Hibernate - Y. Bekkers

39

## Types basiques de « mapping »

Java	Hibernate	SQL
java.lang.String	string	VARCHAR
java.math.BigDecimal java.math.BigInteger	big_decimal big_integer	NUMERIC (NUMBER)
java.util.date, ...	Date, time, timestamp	TIMESTAMP, DATE

Hibernate - Y. Bekkers

40

## Utilisation d'un fichier de mapping Hibernate

- Sauver le fichier : event.hbm.xml
- Le suffixe hbm.xml est une convention dans la communauté des développeurs Hibernate.
- Ranger ce fichier à côté des sources Java
  - Ils doivent être visible via le classpath

Hibernate - Y. Bekkers

41

## Utilisation des objets persistants

Hibernate - Y. Bekkers

42

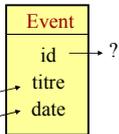
## Utilisation

```

Session session = new Configuration()
 .configure().buildSessionFactory()
Transaction tx =
 session.beginTransaction();
// créer un évènement
Event theEvent = new Event();
theEvent.setTitre(title);
theEvent.setDate(theDate);
session.save(theEvent); // sauver l'évènement

tx.commit();

```



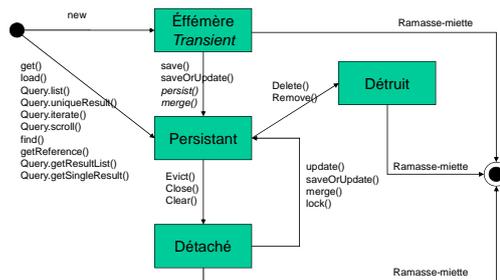
## Rendre une instance persistante

- `session.save(objet)`
  - Pour rendre persistante une instance éffémère (transiente)
  - Génération d'une commande `INSERT` uniquement exécutée au moment du lancement de la méthode `session.commit()`
  - Génération de l'identificateur de l'instance (de type `Serializable` et retourné par la méthode) sauf si de type `assigned` ou `composite`
- `session.merge(objet)`
  - Fusionne une instance détachée avec une instance persistante (existante ou chargée depuis la base)
  - Effectue un `SELECT` avant pour déterminer s'il faut faire `INSERT` ou `UPDATE` ou rien

## Sessions et persistance d'objet

- Le but principal d'une session
  - permettre la création, la lecture et l'effacement d'instances d'objets mappés sur des classes d'entités persistantes.
- Une instance est dans un des trois états :
  - *transient*: jamais persistant, non associé à une session,
  - *persistent*: associé à une unique session
  - *detached*: persistant au préalable, mais plus associé à aucune session

## États d'un objet persistant



## Rendre persistante les modifications apportées à une instance persistante

- Pas de méthode particulière
- Toute modification d'une *instance persistante transactionnelle* (objet chargé, sauvegardé, créé ou requêté par la Session) est rendu persistant par la méthode `flush()`
- Surveillance (*dirty checking*) de toutes les instances persistantes par la session
- Instance persistante modifiée = instance sale (*dirty*)
- Synchronisation avec la base définitive une fois la transaction sous-jacente validée

## Rendre persistante les modifications apportées à une instance détachée

- Pas de surveillance possible des instances détachées
  - nécessité de ré-attacher les instances en rendant persistant les modifications apportées
- `session.merge(objet)`
  - Effectue un `SELECT` avant l'`UPDATE` pour récupérer les données dans la base et les fusionner avec les modifications apportées
  - Retourne l'instance persistante correspondante
  - En accord avec les spécifications EJB 3.0
- `session.update(objet)`
  - Force la mise à jour (`UPDATE`) de l'objet dans la base
  - Lève une exception si une instance de même identificateur existe dans la Session

## Détacher une instance persistante

- Plus (pas) de surveillance de l'instance par la Session
  - Plus (pas) aucune modification rendue persistante de manière transparente
- Trois moyens de détacher une instance :
  - En fermant la session : `session.close()`
  - En vidant la session : `session.clear()`
  - En détachant une instance particulière `session.evict(objet)`

Hibernate - Y. Bekkers

49

## Détacher une instance persistante

- Extraction définitive de l'entité correspondante dans la base de données
- `session.delete(objet)`
  - Enregistrement correspondant plus(pas) présent dans la base
  - Instance toujours présente dans la JVM tant que l'objet est référencé – instance effémère (*transient*)
  - Objet ramassé par la ramasse miette dès qu'il n'est plus référencé

Hibernate - Y. Bekkers

50

## Méthodes d'une session et requêtes sql

- `save()` et `persist()`
  - SQL INSERT
- `delete()`
  - SQL DELETE
- `update()` ou `merge()`
  - SQL UPDATE.
- Modification d'une instance persistante détectée au moment du `flush()`
  - SQL UPDATE.
- `saveOrUpdate()` and `replicate()`
  - INSERT ou UPDATE.

Hibernate - Y. Bekkers

51

## Détacher/réattacher un objet persistant - 1

- Première unité de travail

```
Session session = HibernateUtil
 .getSessionFactory().getCurrentSession();
session.beginTransaction();
Personne aPerson = (Personne) session
 .load(Personne.class, personId);
session.getTransaction().commit();
```

- L'objet `aPerson` est détaché

Hibernate - Y. Bekkers

52

## Détacher/réattacher un objet persistant - 2

- Seconde unité de travail

```
Session session2 = HibernateUtil
 .getSessionFactory().getCurrentSession();
session2.beginTransaction();
aPerson.setAge(age);
session2.update(aPerson);
```

- L'objet `aPerson` est de nouveau persistant

Hibernate - Y. Bekkers

53

## Objets Hibernate utilisés

```
org.hibernate.SessionFactory;
org.hibernate.cfg.Configuration;
org.hibernate.Session;
org.hibernate.Transaction;
```

Hibernate - Y. Bekkers

54

## Transaction typique

```
Session sess = factory.openSession();
Transaction tx;
try {
 tx = sess.beginTransaction();
 //do some work
 ...
 tx.commit();
}
catch (Exception e) {
 if (tx!=null) tx.rollback();
 throw e;
}
finally {
 sess.close();
}
```

Hibernate - Y. Bekkers

55

## Requêtes d'interrogation

Hibernate - Y. Bekkers

56

## Trois types de requêtes SELECT

- SQL
  - sess.createSQLQuery().list()
- HQL
  - sess.createQuery().list()
- Criteria
  - sess.createCriteria().list()

Hibernate - Y. Bekkers

57

## Deux moyens d'effacer

- Avec chargement
  - Acteur p = (Acteur) sess.load(Acteur.class, currentId);
  - sess.delete(p);
- Sans chargement
  - String sqlRequest = "DELETE FROM acteur
  - WHERE p="+currentId;
  - int i = sess.createSQLQuery(sqlRequest).executeUpdate();

Hibernate - Y. Bekkers

58

## Requête SQL

- La méthode list() rend en résultat une liste de tableaux d'Object

```
List<Object[]> result = sess.createSQLQuery(
 "SELECT * FROM personne").list();
for (Object[] o : result) {
 Integer p = (Integer)o[0];
 String nom = (String)o[1];
 String email = (String)o[2];
 ...
}
```

- Chaque objet du tableau est un attribut typé de la ligne résultat

Hibernate - Y. Bekkers

59

## Requêtes HQL

```
result = sess.createQuery("from Personne p,
 Vedette v where p.p=v.id.p and v.id.f=18
 order by p.nom asc").list();
for (Object[] o : result) {
 Personne p = (Personne)o[0];
 Vedette v = (Vedette)o[1];
 ...
}
```

- On obtient deux objets une Personne et une Vedette par ligne résultat

Hibernate - Y. Bekkers

60

## Requêtes criteria

- Recherche sur une classe persistante donnée
- ```
Criteria crit =
  sess.createCriteria(Personne.class);
List<Personne> personnes =
  crit.list();
for(Personne personne : personnes) {
  System.out.println(personne);
}
```
- On obtient directement une liste d'objets de la classe considérée

Hibernate - Y. Bekkers

61

Ajouter un filtre à un criteria

```
Criteria crit =
  sess.createCriteria(Personne.class);
crit.add(Expression.eq("nom", "dupont"));
List<Personne> personnes = crit.list();
for (Personne personne : personnes) {
  sess.delete(personne);
}
```

Hibernate - Y. Bekkers

62

L'équation Hibernate

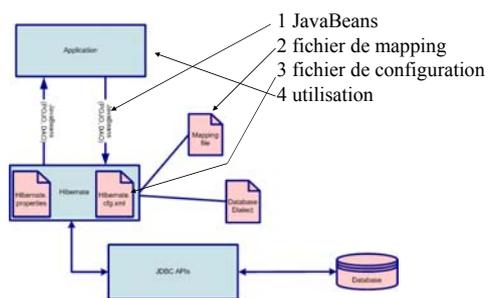
JavaBeans
+ SGBDR
+ Données de mapping et de configuration

= Persistence de données

Hibernate - Y. Bekkers

63

Résumé



Hibernate - Y. Bekkers

64

Intégrer Hibernate dans une application

1. Installer dans votre projet la librairie Hibernate et ses librairies dépendantes
2. Créer vos objets métier (classes persistantes - *javabeans*)
3. Créer le fichier `hibernate.cfg.xml` qui décrit l'accès à la base de données
4. Choisir un dialect SQL pour la base
5. Créer un fichier individuel de « *mapping* » pour chaque classes persistantes

Hibernate - Y. Bekkers

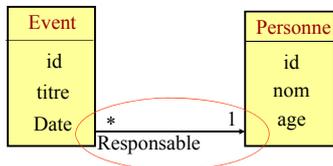
65

Associations

Hibernate - Y. Bekkers

66

Clés étrangères - 1



Association unidirectionnelle plusieurs à 1

Hibernate - Y. Bekkers

67

Modifier la classe Event

- Ajouter le champ responsable

```

public class Event {
    private Long id;
    private String titre;
    private Date date;
    private Personne responsable;
    ...
}
  
```



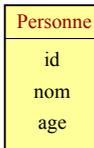
Hibernate - Y. Bekkers

68

Créer une classe Personne

```

public class Personne {
    private Long id;
    private String nom;
    private int age;
    ...
}
  
```



Hibernate - Y. Bekkers

69

Mapping des personnes

- Fichier Personne.hbm.xml

```

<hibernate-mapping>
  <class name="fr.ifsic.events.Personne"
    table="PERSONNE" >
    <id name="id" column="PERSONNE_ID">
      <generator class="native" />
    </id>
    <property name="nom" />
    <property name="age" type="integer" />
  </class>
</hibernate-mapping>
  
```

Hibernate - Y. Bekkers

70

Mapping des Events

- Association unidirectionnelle plusieurs à 1

```

<hibernate-mapping>
  <class name="fr.ifsic.events.Event"
    table="EVENTS" >
    <id name="id" column="EVENT_ID">
      <generator class="native" />
    </id>
    <property name="date" type="timestamp"
      column="EVENT_DATE" />
    <property name="titre" />
    <many-to-one name="responsable"
      column="responsable" not-null="true"
      class="fr.ifsic.events.Personne" />
  </class>
</hibernate-mapping>
  
```

Hibernate - Y. Bekkers

71

Déterminer le type de la classe

```

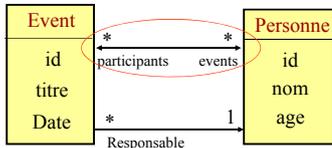
<many-to-one
  name="responsable"
  column="responsable"
  not-null="true"
  class="fr.ifsic.events.Personne"
>
  
```

- L'attribut class est optionnel,
 - il peut être déterminé par réflexion sur le type Java de l'attribut name="responsable"

Hibernate - Y. Bekkers

72

Clés étrangères - 2

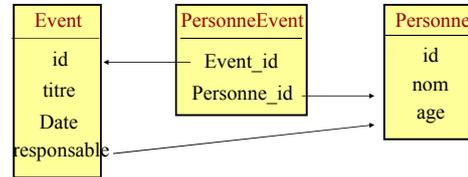


Association bidirectionnelle plusieurs à plusieurs

Hibernate - Y. Bekkers

73

Nécessité d'une table de jointure



Hibernate - Y. Bekkers

74

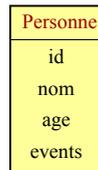
Modifier la classe Personne

- Utiliser un Set

```
public class Personne {

    private Long id;
    private String nom;
    private int age;
    private Set events = new HashSet();

    ...
}
```



Hibernate - Y. Bekkers

75

Types Java pour une collection persistante

- java.util.Set
- java.util.Collection
- java.util.List
- java.util.Map
- java.util.SortedSet
- java.util.SortedMap
- Ou bien toute implémentation de l'interface org.hibernate.usertype.UserCollectionType

Hibernate - Y. Bekkers

76

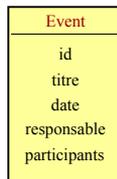
Modifier la classe Event

- Ajouter le champ participants

```
public class Event {

    private Long id;
    private String titre;
    private Date date;
    private Personne responsable;
    private Set participants = new HashSet();

    ...
}
```



Hibernate - Y. Bekkers

77

Mapping des personnes

- Fichier Personne.hbm.xml

```
<hibernate-mapping>
<class name="fr.ifsic.events.Personne"
table="PERSONNE" >
...
<set name="events" table="PersonEvents">
<key column="personId"/>
<many-to-many column="eventId"
class="Event"/>
</set>
</class>
</hibernate-mapping>
```

Hibernate - Y. Bekkers

78

Mapping des Evènement

- Fichier Event.hbm.xml

```
<hibernate-mapping>
  <class name="fr.ifsic.events.Event" table="EVENT" >
    ...
    <set name="participants" table="PersonEvents">
      <key column="eventId"/>
      <many-to-many column="personneId"
        class="Personne"/>
    </set>
  </class>
</hibernate-mapping>
```

Hibernate - Y. Bekkers

79

Manipuler des objets persistants

Hibernate - Y. Bekkers

80

Création d'objets Personne

```
Personne createAndStorePersonne(String nom, int
age) {
  Session session =
  HibernateUtil.getSessionFactory().getCurrentSessi
on();
  Transaction tx = session.beginTransaction();
  Personne p = new Personne();
  p.setNom(nom);
  p.setAge(age);

  session.save(p);
  tx.commit();
  return p;
}
```

Hibernate - Y. Bekkers

81

Création d'objets Event

```
Event createAndStoreEvent(String title, Date theDate,
Personne responsable) {
  Session session =
  HibernateUtil.getSessionFactory().getCurrentSession();
  Transaction tx = session.beginTransaction();

  Event theEvent = new Event();
  theEvent.setTitre(title);
  theEvent.setDate(theDate);
  theEvent.setResponsable(responsable);

  session.save(theEvent);
  tx.commit();
  return theEvent;
}
```

Hibernate - Y. Bekkers

82

Peupler la base

- Ajouter des personnes
Personne p1 = createAndStorePersonne("marcel", 25);
Personne p2 = createAndStorePersonne("jules", 42);
Personne p3 = createAndStorePersonne("toto", 10);
- Ajouter des évènements
Event e1 = createAndStoreEvent("My Event",
new Date(), p2);
Event e2 = createAndStoreEvent("another Event",
new Date(), p1);
- Créer une liste de personnes
Personne[] listP = { p1, p2 };
addParticipant(listP, e1);

Hibernate - Y. Bekkers

83

Ajouter des participants à un évènement

```
void addParticipant(Personne[] persons, Event e) {
  Session session = HibernateUtil.
getSessionFactory().getCurrentSession();
  Transaction tx = session.beginTransaction();
  for (Personne p : persons) {
    e.getParticipants().add(p);
  }
  session.update(e);
  tx.commit();
}
```

Hibernate - Y. Bekkers

84

Les tables initialisées

Personne_id	nom	age
1	Marcel	25
2	Jules	42
3	Toto	10

P_id	E_id
1	1
2	1

id	date	titre	Responsable
1	Xxx	My event	2
2	Yyy	Another event	1

Hibernate - Y. Bekkers

85

Récupérer une instance persistante si on connaît son identifiant

- Cas où on est certain que l'objet existe


```
Personne p = (Personne)
sess.load(Personne.class, 1524);
```
- Cas où on n'est pas certain que l'objet existe


```
Personne p = (Personne)
sess.get(Personne.class, id);
if (p==null) {
    // créer l'objet d'identifiant id
}
```

Hibernate - Y. Bekkers

86

Requêtage

- Quand utiliser les requêtes
 - Lorsque l'on ne connaît pas l'identification d'une entité
- Deux sortes de requêtes
 - HQL


```
List cats = session.createQuery( "from Cat as cat
where cat.birthdate < ?" )
.setDate(0, date)
.list();
```
 - SQL


```
List cats = session.createQuery( "SELECT {cat.*}
FROM CAT {cat} WHERE ROWNUM<10", "cat", Cat.class
).list();
```

Nom de classe

Hibernate - Y. Bekkers

87

Exécution de requêtes si on ne connaît pas d'identifiant

- Par une requête simple


```
session.beginTransaction();
List<Event> result = session.createQuery( "from
Event" ).list();
session.getTransaction().commit();
```
- Par une requête paramétrée


```
session.beginTransaction();
List<Personne> result = session.createQuery( "from
PERSONNE as p WHERE p.nom = ?" )
.setString(0,nom).list();
session.getTransaction().commit();
```



Commencent à 0 (et non à 1 comme JDBC)

Hibernate - Y. Bekkers

88

Paramètres nommés

- Par une requête paramétrée


```
session.beginTransaction();
List<Personne> result =
session.createQuery( "from PERSONNE as p
WHERE p.nom = :nom" )
.setString( "nom", nom ).list();
session.getTransaction().commit();
```

Hibernate - Y. Bekkers

89

Requête qui retourne des tuples

```
Iterator kittensAndMothers = sess.createQuery(
"select kitten, mother from Cat kitten join
kitten.mother mother" )
.list()
.iterator();

while ( kittensAndMothers.hasNext() ) {
    Object[] tuple = (Object[])
kittensAndMothers.next();
    Cat kitten = tuple[0];
    Cat mother = tuple[1];
    ....
}
```

Hibernate - Y. Bekkers

90

Requêtes par critères

```
Criteria crit =
    sess.createCriteria(Cat.class).
Add(Restrictions.eq("p",
    currentId));
crit.setMaxResults(50);
List cats = crit.list();
```

Hibernate - Y. Bekkers

91

Diverses

Hibernate - Y. Bekkers

92

Identifiants composés

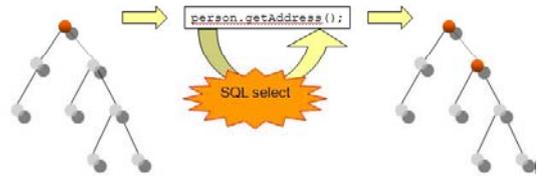
```
<composite-id>
  <key-property name="nom" />
  <key-property name="prénom" />
</composite-id>
```

- On peut utiliser comme contenu des « mappings »
 - <key-property>
 - <many-to-one>
- Les classes persistantes doivent surcharger les méthodes
 - equals()
 - hashCode()

Hibernate - Y. Bekkers

93

Chargement à la demande



Hibernate - Y. Bekkers

94

Héritage

Hibernate - Y. Bekkers

95

Table unique avec discriminant - 1

```
<class name="Payment" class="PAYMENT">
  <id name="id" type="long"
    column="PAYMENT_id">
    <generator class="native" />
  </id>
  <discriminator column="PAYMENT_TYPE"
    type="string" />
  <property="montant" />
  ...
</class>
```

Hibernate - Y. Bekkers

96

Table unique avec discriminant - 2

```
<subclass name="carte"
  discriminator="CREDIT">
  <property name="cctype" column="CCTYPE" />
  ...
</subclass>
<subclass name="cheque"
  discriminator="CHEQUE">
  ...
</subclass>
```

Hibernate - Y. Bekkers

97

Une table par classe fille - 1

```
<class name="Payment" class="PAYMENT">
  <id name="id" type="long"
    column="PAYMENT_id">
    <generator class="native" />
  </id>
  <property="montant" />
  ...
</class>
```

Hibernate - Y. Bekkers

98

Une table par classe fille - 2

```
<joined-subclass name="carte">
  <property name="cctype" column="CCTYPE" />
  <key column="PAYMENT_id" />
  ...
</joined-subclass>
<joined-subclass name="cheque">
  <key column="PAYMENT_id" />
  ...
</joined-subclass>
```

Hibernate - Y. Bekkers

99

Outils

- Plugin Eclipse
 - JBoss Eclipse IDE
 - <http://www.hibernate.org/6.html>
- Ant

Hibernate - Y. Bekkers

100