Formation Groovy ET Grails

IHAB ABADI

Grails - Modélisation

- Grails modélise les models en utilisant par convention GORM pour la pérsistance

- GORM est un ORM construit en surcouche de Hibernate

- GROM, comme Hibernate, apporte des avantages ainsi que des inconvénients

Avantages

- - On accède à des objets métiers au lieu de tables
- - Rapidité de mise en place (petites applications)
- - Syntaxe simple et unique pour tous les SGBD
- - Gestion des transactions
- - Génération et gestion automatique de propriétés –
- - Binding poussé
- - Méthodes dynamiques

Inconvénients

- - Peut être difficile à utiliser dans des projets complexes
- - Surcouche, donc perte de performance
- - On ne sait pas ce qu'il se passe derrière
- - Blocage potentiel sans compréhension globale

Grails - Modélisation

Création du modèle

- Accès à tous les types de base (Int, String, Boolean)
- Création d'une relation 1..n
  - static hasMany = [nomDeLaCollection:Object]
- Création d'une relation 1..1 : Comme un attribut quelconque
  - Utilisation possible de la propriété hasOne

Grails – Modélisation / Relations

One to one

- Unidirectionnelle
- Bidirectionnelle
- Comportement de cascade

Exemple

Grails – Modélisation / Relations

One to one

- Unidirectionnelle
- Bidirectionnelle Utilisation de la propriété belongsTo
- Bidirectionnelle utilisation de la propriété hasOne

Exemple

Grails – Modélisation / Relations

One to Many

- Utilisation d'un Set
- Utilisation de la propriété HasMany
- Unidirectionnel par défaut.
- Comportement de cascade par défaut en save et update
- Utilisation du belongsTo entraine un comportement de cascade pour le delete également

Exemple

Grails – Modélisation / Relations

One To Many

La possibilité d’utilisation de la propriété mappedBy dans le cadre de mélange des relations

Exemple

Grails – Modélisation / Relations

Many To Many

Utilisation de la propriété hasMany

Obligation de définir un responsable de la relation Afin de profiter de la cascade des sauvegardes et updates à l'aide de la propriété belongsTo

L'initialisation des requêtes depuis le responsable de la relation

Pas de scaffolding possible

Exemple d'utilisation

Grails – Modélisation / Relations

Relation one to many est un « Set » par défaut

- Non ordonné
- Contrainte d'unicité

Surcharge possible du type en « List »

- Collection ordonnée
- Gestion automatique des index
- Colonne supplémentaire dans la table
- Si vous manipulez à la main, prudence car une incohérence dans les index fera planter vos requêtes

Grails – Modélisation / Contraintes

Propriété constraints

Permets de définir les contraintes sur les propriétés de nos classes

unique

blank

nullable

Size

url

mail

max,

maxSize …

Grails Modélisation

Possibilité d'héritage

- Eviter si possible -> Gourmand en ressources
- Récupération des attributs du parent

Par défaut « table-per-hierarchy »

- NOT NULL impossible au niveau de la base de donnée
- Possible au niveau métier

Alternative

Exemple

Grails – Modélisation - Mapping

Propriété mapping

Permets de personnaliser les noms des champs générés par GORM dans la base de donnée

Nom de la table

- Id
- Version
- Nom des champs
- Comportement eager / lazy des associations

Permets de personnaliser les propriétés de jointure si on ne veux pas garder les propriétés par défaut

Lazy loading

- Par défaut si un objet possède une référence sur un autre objet, ce dernier ne sera pas chargé en mémoire, si on désire que le comportement de base soit différent, il faut l'expliciter.
- static mapping = { books lazy:false }

Grails – Modélisation - Mapping

Définir si le cache de second-niveau d'Hibernate doit être activé

Optimisations au chargement

- Utilisation lorsque récupération fréquente
- Lorsque pas de réplication
  - Trop complexe à mettre en place
- Ne cache que les récupérations sur ID

Définir les comportements comme les transactions et les cascades SQL peuvent être définies dans ce bloc

- static mapping = { book lazy: false, cascade:"all-delete-orphan“ }

Rappel

- hasMany cascade « save-update »
- + belongsTo cascade « all »

Grails – Modélisation / Manipulation des objets

Sauvegarde

- objectInstance.save()
- Options :
  - Flush : quand défini à « true », persiste l'objet immédiatement
  - Validate : valide la persistance de l'objet
  - Insert : force hibernate à utiliser un INSERT(true) ou un UPDATE (false)
  - failOnError : si la persistance échoue, une exception sera levée (souvent utilisé dans le bootstrap)
  - deepValidate : défini si les objets fils doivent eux aussi être validé

Grails – Modélisation / Manipulation des objets

Ajout

Utilisation de méthode dynamique addTo..

Suppression

Utilisation de la méthode delete

Pour les relations, suppression à l'aide des méthodes dynamique removeFrom

Options

- Flush : quand défini à true persiste l'objet immédiatement

Grails – Modélisation / dynamique Finders

Requêtes sur une base de donnée via GORM

Récupération par Id en utilisant la méthode get

Récupération de tous les objets

- Méthode list
- Méthode listAll

Requête sur une propriété

- findByPropertyName
- findAllByPropertyName

Grails – Modélisation / dynamique Finders

Méthode get
- Retourne une seule instance, exploite le cache d'instance, sera exécuté en une seule requête en base.

getByPropertyName

Opérateur dynamique comme les autres findByXXX, pas de cache d'instance, utiliser « get » à la place

Méthode getAll
- Version complète de « get », retourne une « List »
- Sans paramètres
  - Retourne une liste de tous les objets
- getAll(1,2,3) ou getAll([1,2,3])
  - Retourne une liste contenant respectivement les instances ayant les id 1, 2 et 3

Méthode findAll
- Equivalent de getAll qui permet d'utiliser du HQL
- Supporte la pagination

Méthode list
- Retourne toutes les instances de l'objet
- Supporte la pagination

Grails – Modélisation / dynamique Finders

Accesseurs Dynamiques

Pratique pour des cas simples

Rapidement illisible

Grails – Modélisation / Where Queries

Utilisation possible pour chercher sur un « Range »

Enchainement illimité de requêtes

Requêtes possibles sur les associations

Utilisation possible des sous requêtes dans les where queries

Utilisation possible des sous requêtes dans les where queries

De nombreuses autres fonctions sont disponible

Grails – Modélisation / Utilisation de criteria

Mode de requête permettant de construire des requêtes complexes

Permets une très grande liberté au prix d'une syntaxe plus verbeuse

Si une requête ne peut être formulée via les « Critères » il faudra la faire via HQL ou directement en SQL

Grails – Modélisation / Utilisation de criteria

Comparaison

	dynamic finder	where clause	criteria	HQL	SQL
simple queries	x	x	x	x	x
complex filters		x	x	x	x
associations		x	x	x	x
property comparisons		x	x	x	x
some subqueries		x	x	x	x
eager fetches w/ complex filters			x	x	x
projections			x	x	x
queries with arbitrary return sets				x	x
highly complex queries (like self joins)					x
some database specific features					x
performance-optimized queries					x