Grails GORM

Alexis Plantin

Sommaire

- Récupérer les objets
- Lister, trier et compter
- Sauvegarder, mettre à jour et supprimer
- addTo et removeFrom
- Dynamic Finders
- Criteria queries
- HQL
- Pagination

Récupérer les objets

Récupérer les objets

- Chaque Domain Class possède un certain nombre de méthodes grâve à la métaprogrammation
- Récupérer une instance en mode read-write

```
def product = Product.get(1)
```

- Récupérer une instance en read-only

```
def product = Product.read(1)
```

- Récupérer une liste d'instances

```
def products = Product.getAll([1, 2, 3])
```

Lister, trier et compter

Lister, trier et compter

- La méthode List et ses paramètres
  - max: le nombre maximum de résultats
  - offset: l'index du premier résultat à retourner
  - sort: le nom de l'attribut sur lequel trier
  - order: l'ordre du tri: asc ou desc

```
def staffs = Staff.list()
def lastStaffs = Staff.list(max: 5, sort: 'since', order: 'desc')
```

Lister, trier et compter

- Les méthodes dynamiques listOrderBy*

```
def lastStaffs = Staff.listOrderBySince(max: 5, order: 'desc')
```

- Une méthode pour compter toutes les instance en base

```
def numberOfStaffs = Staff.count()
```

Sauvegarder, mettre à jour et supprimer

Sauvegarder, mettre à jour et supprimer

- Sauvegarder une instance

```
def product = new Product(params)
product.save()
```

- Mettre à jour une instance

```
def product = Product.get(1)
product.properties = params
product.save()
```

- Supprimer une instance

```
product.delete()
```

addTo et removeFrom

addTo et removeFrom

```
import wrh.*

def location = new Location(name: 'Turin', country: 'it',
address: 'Turin')
    .addToWarehouses(name:'Turin Warehouse 1')
    .addToWarehouses(name:'Turin Warehouse 2')
    .save()

println 'Before delete operation'
println 'Location: ' + location
println 'Warehouses: ' + location.warehouses



def warehouse = Warehouse.get(1)
location.removeFromWarehouses(warehouse)
println 'After delete operation'
println 'Location: ' + location
println 'Warehouses: ' + location.warehouses
```

Dynamic Finders

Dynamic Finders

- Utiliser les noms d'attributs des domain classes pour effectuer des requêtes
- Permet les requêtes logiques avec `And`, `Or` et `Not`
- `findBy` et `findAllBy`

```
new Product(name: 'iPad', weight: 1.1).save()
def product1 = Product.findByNameAndWeight('iPad', 1.2)
println product1 ?: 'Product not found'
def productList = Product.findAllByNameOrWeight('iPad', 1.2)
println productList ?: 'Products not found'
```

Dynamic Finders

- Between

```
Product.findByWeightBetween(1, 2)
```

- Equals

```
Product.findByNameEquals('iPad')
```

- GreaterThan

```
Product.findByWeightGreaterThan(1.1)
```

- GreaterThanEquals

```
Product.findByWeightGreaterThanEquals(1.1)
```

Dynamic Finders

- InList

```
Product.findByNameInList (['Ipad', 'iPad'])
```

- IsNull

```
Product.findByDescriptionIsNull()
```

- IsNotNull

```
Product.findByDescriptionIsNotNull()
```

- LessThan

```
Product.findByWeightLessThan(1.1)
```

Dynamic Finders

- LessThanEquals

```
Product.findByWeightLessThanEquals(1.1)
```

- Like

```
Product.findByNameLike('%Pa%')
```

- NotEqual

```
Product.findByNameNotEqual('iPad')
```

Criteria queries

Criteria queries

- Grails fournit une méthode statique sur chaque Domain Class `createCriteria()`
- 3 méthodes intéressantes
  - get: récupérer la première instance correspondant à la requête
  - list: retourner la liste des instances correspondant à la requête
  - count: nombre total d'instances correspondant à la requête

Criteria queries

```
def c = Product.createCriteria()
def results = c {
    between("weight", 0d, 5d)
    or {
        eq("name", "iMac")
        like("name", "iP%")
    }
    maxResults(10)
    order("name", "desc")
}
```

Criteria queries

- Voir les Hibernate Restrictions pour plus de détail
- Conjunctions et Disjunctions
  - `or`, `and` & `not`

```
or {
    between("weight", 1, 3)
    eq("name", "iPad")
}
```

Criteria queries

- Requêter sur les associations

```
def c = Location.createCriteria()
def results = c.list {
    eq('name', 'Carmes')
    warehouses {
        eq('name', 'Warehouse 1')
    }
}
```

Criteria queries

- Requêtes et projections

```
def c = Product.createCriteria()
def avgWeight = c.get {
    projections {
        avg('weight')
    }
}
```

- Voir les Hibernate Projections pour plus de détails

HQL

HQL

- Comme les requêtes SQL

```
def results = Product.findAll("from Product as p where p.name
like 'iP%'")
```

- Paramètres

```
def results = Product.findAll("from Product as p where p.name
like ?", ['iP%'])
```

- Paramètres et associations

```
def location = Location.findByName('Carmes')
def warehouses = Warehouse.findAll("from Warehouse as w where
w.location = ?", [location])
```

HQL

- Paramètres nommés

```
def results = Product.findAll("from Product as p where p.name
like :name", [name: 'iP%'])
```

- Chainer les paramètres

```
def location = Location.findByName('Carmes')
def warehouses = Warehouse.findAll("from Warehouse as w where
w.location = :location and w.name like :name", [location:
location, name: '%1'])
```

Questions?