Advanced GORM

1

Advanced GORM

- Dynamic finders
- Hibernate Query Language (HQL)
- Criteria Builder
- Where Queries
- Named Queries
- Events / Automatic timestamping

2

Dynamic Finders

3

Dynamic Finders

- Synthesized at runtime (not compiled)
- Made of up to two properties, boolean operators and comparators
- Additional parameters can be passed for pagination and sorting

4

Dynamic Finders

- findAllBy
    - returns list of objects
- findBy
    - returns single object

5

Dynamic Finders

```
findByName('mike')
findAllByName('mike')
findAllByNameAndAccountNumber('mike', '123')
findAllByNameOrAccountNumber('mike', '789')
findAllByNameLike('J%')

findAllByNameIlike('j%',
    [sort:'name'])

findAllByNameIlike('j%',
    [order:'desc',sort:'name'])

findAllByNameIlike('j%',
    [order:'desc', sort:'name',
    max:10, offset:0])
```

meta programming

modify runtime behavior

```
class Person {
    String name
    Integer age
}
Person.findByName('Mike')
```

metaClass

```groovy
def fakeDb = [
    new Person(name:'Mike', gender:'M'),
    new Person(name:'Robin', gender:'F')]

class Person {
    String name
    String gender
    String toString() {name}
}

// define a metaClass method on Person
Person.metaClass.static.findByName =
{ String name->
    return fakeDb.findAll {it.name == name}
}

Person.findByName('Robin')
```

invokeMethod

```groovy
class AddressBook {
  def people = [
    new Person(name:'Mike', gender:'M'),
    new Person(name:'Robin', gender:'F')]

  def invokeMethod(String name, args) {
    def propToFind = name - 'findBy'
    people.findAll {
      it[propToFind.toLowerCase()] == args[0]
    }
  }
}

def ab = new AddressBook()
ab.findByName('Robin')
ab.findByGender('F')
```

methodMissing

```groovy
class AddressBook {
  static people = [
      new Person(name: 'Mike', gender: 'M'),
      new Person(name: 'Robin', gender: 'F')]

  // intercept
  def methodMissing(String name, args) {
    println 'inside methodMissing'
    def impl = { Object[] theArgs ->
      def propToFind = name - 'findBy'
      people.findAll {
        it[propToFind.toLowerCase()] == args[0]
      }
    }
    // cache
    AddressBook.metaClass."${name}" = impl
    // invoke
    return impl(args)
  }
}
def ab = new AddressBook()
ab.findByName('Robin')
```

intercept

```groovy
class AddressBook {
  static people = [
      new Person(name: 'Mike', gender: 'M'),
      new Person(name: 'Robin', gender: 'F')]

  // intercept
  def methodMissing(String name, args) {
    println 'inside methodMissing'
    def impl = { Object[] theArgs ->
      def propToFind = name - 'findBy'
      people.findAll {
        it[propToFind.toLowerCase()] == args[0]
      }
    }
    // cache
    AddressBook.metaClass."${name}" = impl
    // invoke
    return impl(args)
  }
}
def ab = new AddressBook()
ab.findByName('Robin')
```

cache

```
class AddressBook {
  static people = [
      new Person(name: 'Mike', gender: 'M'),
      new Person(name: 'Robin', gender: 'F')]

  // intercept
  def methodMissing(String name, args) {
    println 'inside methodMissing'
    def impl = { Object[] theArgs ->
      def propToFind = name - 'findBy'
      people.findAll {
        it[propToFind.toLowerCase()] == args[0]
      }
    }
    // cache
    AddressBook.metaClass."${name}" = impl
    // invoke
    return impl(args)
  }
}
def ab = new AddressBook()
ab.findByName('Robin')
```

invoke

17

summary

18

- First Time
  - ab.findByName('Robin')
  - invokeMethod
  - methodMissing
  - method cached
  - method invokved
- Second Time
  - ab.findByName('Robin')
  - invokeMethod
  - method invoked

19

Problem

- Something more advanced than a dynamic finder

- Dynamic finders are great but can lead to programmer laziness and performance issues

20

Example

Find all customers with gold service level and more than 5 incidents

```
def gold = ServiceLevel.findByName('Gold')

List goldCustomers =
  Customer.findAllByServiceLevel(gold)

List moreThanFive =
  goldCustomers.findAll { cust->
    cust.incidents.size() > 5
}
```

21

```
def gold = ServiceLevel.findByName('Gold')
// select * from service_level where name = ?

List goldCustomers =
  Customer.findAllByServiceLevel(gold)
// select * from customer where service_level_id = ?

List moreThanFive =
  goldCustomers.findAll { cust->
    // iterates through each customer in the list

    cust.incidents.size() > 5
    // select * from incident where customer_id = ?
    // (for each customer!)
}
```

22

HQL

23

HQL

- Hibernate Query Language
- SQL-like query language using domain and property names rather than DB names

```
Customer.findAll(
  "from Customer as c where c.name = 'Mike'")

Customer.findAll(
  "from Customer as c where c.name = ?", ['Mike'])

Customer.findAll(
  "from Customer as c where c.name = :name",
    [name:'mike'])
```

24

---

methods

- find
- findAll
- executeQuery
- executeUpdate

25

---

find / findAll

- find: returns a single object (first found)
- findAll: returns a list of objects

26

```
Customer.findAll(
  "from Customer as c
   inner join fetch c.address
   where c.serviceLevel.name = ?
   and size(c.incidents) > 5",
    ['Gold'])



Customer.findAll(
  "from Customer as c              <--- class name
   inner join fetch c.address     <--- property
   where c.serviceLevel.name = ? <--- parameter
   and size(c.incidents) > 5",
    ['Gold'])                     <--- parameter value
```

27

---

executeQuery

- doesn’t return a domain class
- good for getting a subset of data
    - loading a single column
    - count, min, max, etc

28

```
Address.executeQuery(
    "select distinct state, count(*)
    from Address
    group by state")
```

Returns a list of results, where each result is a list itself, e.g.:

```
[ [MN, 5], [WI, 10] ]
```

29

---

executeUpdate

- Data Manipulation
    - UPDATE
    - DELETE
- e.g. update all silver statuses to platinum
- delete all accounts with no activity in the 90 days

30

```
Customer.executeUpdate(
    '''update Customer c
    set c.serviceLevel = :newSl
    where c.serviceLevel = :oldSl''',
    [
     newSl:ServiceLevel.findByName('SuperAwesome'),
     oldSl: ServiceLevel.findByName('Gold')
    ]
)
```

31

Criteria

32

Criteria

- Grails provides a Hibernate Criteria Builder

- Useful for forming dynamic queries
- Two methods:
  - createCriteria
  - withCriteria - inline criteria builder

33

---

restrictions

- `eq` - equal
- `ilike` - case insensitive like (wildcard: %) like - case sensitive like (wildcard: %)
- `in` - in a list
- `isNull`
- `lt`, `le` - less than; less than or equal
- `gt`, `ge` - greater than; greater than or equal

34

---

query methods

- list - (default) returns all matching rows
- listDistinct - return a distinct set of results
- get - retrieve one row (useful for projections)
- scroll - returns a scrollable result set

35

Criteria

```
Customer.withCriteria {
    eq('name', 'mike')
}

// is the same as

def c = Customer.createCriteria()
c {
    eq('name', 'mike')
}
```

36

Criteria

```
def c = Customer.createCriteria() c{
    or {
        eq('name', 'mike')
        eq('accountNumber', '789')
    }
}

// is the same as

findAllByNameOrAccountNumber('mike', '789')
```

Criteria

```
def customerByName(nameToFind) {
    def c = Customer.createCriteria()
    return c.list() {
        eq('name', nameToFind)
    }
}

// is the same as

def customerByName(nameToFind) {
    return Customer.withCriteria {
        eq('name', nameToFind)
    }
}
```

Example

- simple: find all customers by name
- find all customers with gold service level and more than 5 incidents
- Advanced Search function
    - name
    - account number
    - state

```
def customersByName(String theName) {
    def c = Customer.createCriteria()
    return c.list() {
        eq('name', theName)
    }
}
```

simple: find all customers by name

40

```groovy
def goldWithFiveCriteria() {
    def c = Customer.createCriteria()
    return c.list() {
        serviceLevel {
            eq('name', 'Gold')
        }
        sizeGt('incidents', 5)
    }
}
```

find all customers with gold service level and more than 5 incidents

41

```groovy
def search(accountNumber, name, state) {
  def c = Customer.createCriteria()
  return c.list {
    or {
      if (accountNumber) {
        ilike('accountNumber', "${accountNumber}%")
      }
      if (name) {
        ilike('name', "${name}%")
      }
    }
    if (state) {
      address {
        eq('state', state)
      }
    }
  }
}
```

"advanced" search function

42

Where Queries

43

Where Queries

- Define a query using boolean logic

44

Where Query

```
Customer.where {
    serviceLevel.name == 'Gold'
    &&
    incidents.size() > 5
}.list()
```

find all customers with gold service level and more than 5 incidents

45

Combining Where Queries

```
def query = Person.where {
     lastName == "Simpson"
}
def bartQuery = query.where {
     firstName == "Bart"
}
Person p = bartQuery.find()
```

46

Named Queries

- Allow you to define criteria queries as part of the domain class
- Can be chained together
- Can be combined with dynamic finders
- Break down queries into components

47

```
class Customer {
    //...
    static namedQueries = {
        goldLevelMoreThanFiveIncidents {
            byServiceLevelName('Gold')
            moreThanFiveIncidents()
        }

        moreThanFiveIncidents {
            sizeGt('incidents', 5)
        }

        byServiceLevelName { serviceLevelName ->
            serviceLevel {
                eq('name', serviceLevelName)
            }
        }
    }
```

Auto Timestamping

- Two special properties for domain classes
    - dateCreated
    - lastUpdated
- Must be nullable
- Will be set automatically when object is persisted to the database

---

Events

- beforeInsert - Executed before an object is initially persisted to the database
- beforeUpdate - Executed before an object is updated
- beforeDelete - Executed before an object is deleted
- beforeValidate - Executed before an object is validated
- afterInsert - Executed after an object is persisted to the database
- afterUpdate - Executed after an object has been updated
- afterDelete - Executed after an object has been deleted
- onLoad - Executed when an object is loaded from the database