RxLogix Corporation

GORM II

Parveen Soni (parveen-rx)

(pvintake@RxLogix)

23-Mar-2020

GORM II - Objective

- Continue from Domain class (GORM I)
- Grails Criteria
- Methods of Criteria
- Selection and Projections in Criteria
- Writing HQL queries
- Writing named queries
- Grails console plugin and its usage
- Best Practices to write/use the criteria (selections/projection)

Continue from Domain class (GORM I)

- Grails Domain Class
- Mapping of Domain class (One to One, One to Many, Many to Many etc.)
- Persistent Fields – Core to business - store in database
- Transient Fields – usage but no persistent
- Validation and Constraints
- Timestamps – dateCreated and lastUpdated – Grails Special Fields
- Grails Finders (FindBy, FindAll, etc.)
- Behind the source – Powered By Hibernate

Grails Criteria

- It is just another way of querying database using GORM. The Groovy builder (used to query) uses Hibernate’s Criteria API.

- It is mostly used when you are writing more complex queries which have more than two condition.

- Criteria can be used either via the createCriteria or withCriteria methods.

- You can write the query in criteria like:

a)
```
User.withCriteria {
        //some conditions....
}
```

b)
```
User.createCriteria().list{
        //some conditions....
}
```

- By default the criteria returns a list of objects. In above examples we will get list of User objects.

Methods of Criteria

- Four methods can be invoked in createCriteria:
  - get : This locates a unique instance for the query. (It will throw exception if your criteria do not give a unique result)
  - list : This returns the list of instances for the query. (It can give repetitive results in case of querying on associations. It comes with useful method on the list returned by criteria i.e. totalCount, useful when using pagination.)
  - listDistinct : This is used to get the distinct result from the query.
  - count : This returns total results as integer.

Selection and Projections in Criteria

- Like what we used to write in sql statement as where condition:
  - between-: between("balance", 500, 1000) – value in range
  - eq -: eq("name", "Parveen") – value is equal
  - eqProperty -: eqProperty("firstName","lastName") – two props. are same
  - gt -: gt("balance",1000) – greater than
  - gtProperty -: gtProperty("balance","overdraft") – one prop gt other
  - ge -: ge("balance",1000) – greater than equal
  - geProperty -: geProperty("balance","overdraft") - one prop gte other
  - idEq -: idEq(1) – object id equal
  - ilike -: ilike("firstName","Steph%") – case insensitive like search
  - In -: 'in'("accounts",[account1,account2]) – value in given set

Selection and Projections in Criteria (cont.)

- Like what we used to write in sql statement as where condition:
  - isEmpty -: isEmpty("accounts") – if prop. is empty
  - isNotEmpty -: isNotEmpty("accounts) – if prop. is not empty
  - isNull -: isNull("balance") - if prop. is null
  - isNotNull -: isNotNull("balance") - if prop. is not null
  - lt-: lt("balance",1000) – less than
  - ltProperty -: leProperty("balance","overdraft") – less than other prop.
  - le -: le("balance",1000) – less than equal
  - leProperty -: leProperty("balance","overdraft") – less than equal to other prop.
  - like -: like("firstName","Steph%") – case sensitive like search
  - ne -: ne("name", "London") – not equal to
  - neProperty -: neProperty("firstName","lastName") – not equal to other prop.
  - order -: order("firstLastName", "desc") – order by

Selection and Projections in Criteria (cont.)

- Criteria selection: List
  - See demo example
- Criteria selection: listDistinct
  - See demo example
- Criteria selection: and, or and not
  - See demo example
- Criteria selection: projections (property)
  - See property projection demo example
  - See count, sum, avg projection demo example

Writing HQL queries

- executeQuery: This method is used to select fields of a domain class rather than fetching the whole object

  List usersInfo = User.executeQuery("select firstName, lastName from User where age >:age ", [age:18])

- The above query will return list of list which have only firstName and lastName of User rather than the object.

Writing HQL queries (cont.)

- executeUpdate: This method is used when you want to delete or updating the objects directly without loading them.

  User.executeUpdate("update User set firstName=:firstName where id=:id", [firstName:"Test User",id:1.toLong()])

  - The above example will update the user which have id 1. The same thing can be executed by loading the object by get method and then updating its firstName , which will cause 2 query to database one for get and one for update.

Writing named queries

- NamedQueries are static closure properties of domain class which is used as reusable criteria closure

- Very useful for reusability of query

-
  ```
  static namedQueries = {
              search { AccountSearchCO co ->
                          if (co.branch) {
                                      eq('branch', co.branch)
                                  }
                      if (co.balance) {
                            ge('balance', co.balance)
                          }
                  }
          }
  ```

Writing named queries (cont.)

- Now you can query it like
  - List<Account> accounts = Account.search(co).list()

  - List<Account> accounts = Account.search(co).list(max:co.max,order:co.order,sort:co.sort, offset:co.offset)

  - Integer totalCount = Account.search(co).count()

  - List<Account> accounts = Account.search(co).findAllByDateCreatedLessThan(new Date() - 1, [max: co.max, order: co.order, sort: co.sort, offset: co.offset])

  - Account.search(co).countByDateCreatedLessThan(date - 1)

Grails console plugin and its usage

- Grails console plugin enable application access to run code on the fly.
- Can be used to evaluate the code w.r.t current application code.
- Service Bean can be accessed using *ctx.<serviceName>.*
- Domain classes can also be accessed from console like other class.
- Other code might require some import as and when needed.
- Pros and Cons of console plugin
- Demo example – dependency, and URL

Best Practices

- Never write any query (specially criteria and HQL) in controller because you cannot reuse them
- Use projections whenever the whole object in not needed.
- Try to avoid HQL but it can be used if other queries taking too much time to get or update.
- HQL should always use map parameter.
- Console plugin should never be exposed for public, only special users should have its access
- Disable grails console in production application

Q & A

- Fork on GitHub: https://github.com/parveen-rx/grails-taglib-views-sample
- Use branch: master
- email: parveen.soni@rxlogix.com
- Follow on GitHub: https://github.com/parveen-rx

Thank You