Artigo

Hibernate API Criteria: Realizando consultas

Veja neste artigo como podemos fazer consultas utilizando a API Criteria fornecida pelo Hibernate e quais são as principais funcionalidades que podemos utilizar nas consultas.

O Hibernate provê três tipos diferentes para retornar informações, uma é a HQL, outra forma é através de consultas nativas e a terceira forma é através da API Criteria.

A API Criteria nos permite construir consultas estruturadas utilizando Java, e

No restante do artigo estaremos verificando melhor o que é essa API e o que ela oferece aos desenvolvedores Hibernate.

API Criteria

A API Criteria possui a interface org.hibernate.Criteria que está disponível com todos os métodos fornecidas pela API. Através dessa API podemos construir consultas de forma programática. A interface Session do Hibernate contém vários métodos createCriteria() que recebem como parâmetro uma classe de um objeto persistente ou o nome da entidade e, dessa forma, o Hibernate irá criar um objeto Criteria que retorna instâncias das classes de objetos persistentes quando a aplicação executa uma consulta com a API. Na **Listagem 1** temos um exemplo introdutório.

Listagem 1. Exemplo utilizando createCriteria.

```
1 Criteria crit = session.createCriteria(Produto.class);
2 List results = crit.list();
```

Nessa pesquisa temos como retorno todos os objetos que são instância da classe Produto. Classes derivadas de produto também são retornadas.

Restrições com Criteria

A API Criteria também permite que possamos criar restrições onde restringimos os objetos retornados. Um exemplo disso seria retornar apenas produtos que tenham um preço abaixo de 30 reais. Para usar restrições da API utilizamos o

Embora possam criar nossos próprios objetos implementando o objeto Criterion ou estendendo um objeto Criterion já existente, recomenda-se que os desenvolvedores usem objetos Criterion embutidos da lógica de negócio da aplicação. Também podemos criar nossa própria classe factory que retorna instâncias do objeto Criterion do Hibernate apropriadamente configurados para as restrições da nossa aplicação.

Usamos métodos de fábrica (factory) na classe org.hibernate.criterion.Restrictions para obter instâncias dos objetos Criterion. Para retornar objeto que possuem um valor de propriedade que é igual à nossa restrição, usamos o método eq() em Restrictions. A assinatura do método é:

```
1 | public static SimpleExpression eq(String propertyName, Object value)
```

Podemos passar o método eq() para o método add() no objeto Criteria. Segue na **Listagem 2** um exemplo de como procuraríamos um teclado nos produtos:

Listagem 2. Criando uma restrição com eq().

```
1 | Criteria crit = session.createCriteria(Produto.class);
2 | crit.add(Restrictions.eq("nome","Teclado"));
3 | List results = crit.list();
```

Se quiséssemos procurar produtos que não tivessem teclado faríamos como na **Listagem 3**, utilizando o método ne():

Listagem 3. Criando uma restrição com ne().

Objetos nulos podem ser encontrados utilizando isNull(), ou então podemos utiliza isNotNull() para pesquisar objetos não nulos, como na **Listagem 4**.

Listagem 4. Criando uma restrição com isNull().

```
1 Criteria crit = session.createCriteria(Produto.class);
2 crit.add(Restrictions.isNull("nome"));
3 List results = crit.list();
```

Outra pesquisa muito utilizada é quando queremos retornar objetos que possuem algumas propriedades em comum, como a cláusula LIKE do SQL. Para isso utilizamos o método ilike() ou like(). A diferença é que o método ilike() é case-insensitive, ou seja, maiúsculas ou minúsculas são desprezadas. Segue na **Listagem 5** um exemplo da utilização de like.

Listagem 5. Criando uma restrição com like().

```
1 Criteria crit = session.createCriteria(Produto.class);
2 crit.add(Restrictions.like("nome","Tec%"));
3 List results = crit.list();
```

O símbolo "%" é utilizado para combinar partes de uma String. Dessa forma, tudo que começar com Tec e tiver quaisquer outros caracteres serão retornados.

Os métodos ilike e like possuem duas assinaturas:

```
1 public static SimpleExpression like(String propertyName, Object valu
```

ou

Podemos notar a presença de um MatchMode matchMode a mais na terceira assinatura do segundo método. O objeto org.hibernate.criterion.MatchMode é utilizado para especificar como combinar os valores especificados. O objeto MatchMode possui quatro diferentes combinações:

- ANYWHERE: Qualquer parte na String.
- END: Final da String.
- EXACT: Combinação exata.
- START: Inicio da String.

Segue na **Listagem 6** um exemplo que usa o método ilike() para pesquisar combinações de case-insensitive no final da String.

Listagem 6. Criando uma restrição com ilike().

```
1 Criteria crit = session.createCriteria(Produto.class);
2 crit.add(Restrictions.ilike("nome","browser", MatchMode.END));
3 List results = crit.list();
```

A API Criteria também oferece métodos para comparações como gt() (greater-than) que verifica se é maior que, ge() (greater-than-or-equal-to) que verifica se é maior que e igual, lt() (less-than) que verifica se é menor, e por fim le() (less-than-or-equal-to) que verifica se é menor e igual. No exemplo da **Listagem 7** pesquisamos por produtos que tenham um valor menor que 30 reais.

Listagem 7. Pesquisando valores menores que 30.

```
1 Criteria crit = session.createCriteria(Produto.class);
2 crit.add(Restrictions.gt("preco",new Double(30.0)));
3 List results = crit.list();
```

Listagem 8. Criando mais de uma restrição (equivalente a AND).

```
1 Criteria crit = session.createCriteria(Produto.class);
2 crit.add(Restrictions.gt("preco",new Double(30.0)));
3 crit.add(Restrictions.like("nome","Tec%"));
4 List results = crit.list();
```

Nesse caso estamos pesquisando produto com preço maior que 30 reais que tenham um nome começando com Tec.

Para fazer um OR nessas duas restrições teríamos que utilizar o método or() como na **Listagem 9.**

Listagem 9. Criando mais de uma restrição com OR.

```
1 Criteria crit = session.createCriteria(Produto.class);
2 Criterion preco = Restrictions.gt("preco",new Double(30.0));
3 Criterion nome = Restrictions.like("nome","Tec%");
4 LogicalExpression orExp = Restrictions.or(preco,nome);
5 crit.add(orExp);
6 List results = crit.list();
```

Quando temos três OR ou mais podemos representar isso de uma forma melhor utilizado o objeto org.hibernate.criterion.Disjunction. Para representar uma expressão AND com mais que dois Criteria podemos usar o método conjunction(). Segue na **Listagem 10** um exemplo utilizando disjunction.

Listagem 10. Utilizando disjunction() com Criteria.

```
1 Criteria crit = session.createCriteria(Produto.class);
```

```
disjunction.add(desc);
crit.add(disjunction);
 List results = crit.list();
```

O último tipo de restrição é o sqlRestriction(). Essa restrição permite especificarmos um SQL diretamente na API Criteria. Essa restrição é necessária quando queremos usar cláusulas SQL que o Hibernate não suporta através da API Criteria. Veja a **Listagem 11.**

Listagem 11. Utilizando sqlRestriction() com Criteria.

```
Criteria crit = session.createCriteria(Produto.class);
crit.add(Restrictions.sqlRestriction("{alias}.nome like 'Tec%'"));
List results = crit.list();
```

O {alias} é utilizado quando não sabemos o nome da tabela que a nossa classe utiliza. Dessa forma, {alias} significa o nome da tabela da classe.

O método sqlRestriction() possui ainda mais dois métodos que permitem passar paramêtros e valores JDBC para o comando SQL.

Paginando Resultados

Criteria também faz paginação sobre os resultados de uma consulta. Existem dois métodos na interface da API Criteria para realizar a paginação, são eles: setFirstResult() e setMaxResults(). O método setFirstResult() requer um inteiro que representa a primeira linha no conjunto de dados retornado, iniciando com a linha 0. O método setMaxResults() retorna um número fixo de objetos. Segue na

```
Criteria crit = session.createCriteria(Produto.class);
crit.setFirstResult(1);
crit.setMaxResults(2);
List results = crit.list();
```

Retornando Resultados Únicos

Algumas vezes desejamos retornar zero ou um único objeto para algumas consultas. Isso acontece quando usamos agregações como COUNT ou porque as restrições na consulta levam a um resultado único. Podemos limitar os resultados de qualquer consulta setando apenas setMaxResults() para que ele retorne um único objeto. Porém, existe outra forma de fazer isso que tem como resultado um Object ao invés de um List, é o método uniqueResult(). O método uniqueResult() retorna um único resultado ou null. Caso tenhamos mais de um resultado é lançada uma exceção NonUniqueResultException. Na **Listagem 13** temos um exemplo da sua utilização.

Listagem 13. Retornando resultados único com uniqueResult().

```
Criteria crit = session.createCriteria(Produto.class);
Criterion preco = Restrictions.gt("preco",new Double(25.0));
crit.setMaxResults(1);
Produto produto = (Produto) crit.uniqueResult();
```

Ordenando Resultados

A API Criteria oferece aos desenvolvedores a classe org.hibernate.criterion.Order para ordenar os resultados em ascendente ou descendente de acordo com as

```
Criteria crit = session.createCriteria(Produto.class);
crit.add(Restrictions.gt("preco",new Double(25.0)));
crit.addOrder(Order.desc("preco"));
List resultado = crit.list();
```

Os métodos asc() e desc() são utilizados para ordenar de forma ascendente ou descendente respectivamente.

Associações

Para adicionar uma restrição na que classe que esta associada com Criteria, precisamos criar outro objeto Criteria, e passar o nome da propriedade da classe associada para o método createCriteria(). Dessa forma, teremos outro objeto Criteria. Segue na **Listagem 15** temos um exemplo da utilização da associação onde se criou um novo objeto Criteria para a propriedade produtos e adicionou-se uma restrição para um atributo da propriedade.

Listagem 15. Utilizando associações com Criteria.

```
Criteria crit = session.createCriteria(Suplementos.class);
Criteria prdCrit = crit.createCriteria("produtos");
prdCrit.add(Restrictions.gt("preco",new Double(25.0)));
List results = crit.list();
```

Projeção e Agregação

Para usar projeções começamos com um objeto org.hibernate.criterion.Projection que precisa de uma fábrica org.hibernate.criterion.Projections. A classe Projections é similar à classe

referência a objetos que podemos fazer um cast para o tipo apropriado. Na **Listagem 16** temos um exemplo da sua utilização.

Listagem 16. Utilizando projeções com Criteria.

```
1 Criteria crit = session.createCriteria(Produto.class);
2 crit.setProjection(Projections.rowCount());
3 List results = crit.list();
```

Podemos notar que a projeção contém uma função agregada. Outras funções agregadas são listadas abaixo:

- avg(String propertyName): Retorna a média do valor da propriedade.
- count(String propertyName): Conta o número de vezes que a propriedade ocorre.
- countDistinct(String propertyName): Conta o número de valores únicos que contém a propriedade.
- max(String propertyName): Calcula o valor máximo dos valores da propriedade.
- min(String propertyName): Calcula o valor mínimo dos valores da propriedade.
- sum(String propertyName): Calcula a soma total dos valores da propriedade.

Também podemos ter uma lista de projeções, como mostra o exemplo da **Listagem 17.**

Listagem 17. Criando uma lista de projeções.

```
1 Criteria crit = session.createCriteria(Produto.class);
```

```
8 List results = crit.list();
```

Utilizando projeções também podemos retornar apenas propriedades individuais ao invés de entidades. Para isso basta utilizarmos o método property() na classe Projections. Na **Listagem 18** temos um exemplo que demonstra isso.

Listagem 18. Retornando Propriedades ao invés de Entidades.

```
1 Criteria crit = session.createCriteria(Produto.class);
2 ProjectionList projList = Projections.projectionList();
3 projList.add(Projections.property("nome"));
4 projList.add(Projections.property("descricao"));
5 crit.setProjection(projList);
6 List results = crit.list();
```

Retornar propriedades também é melhor para a performance da aplicação, pois com isso temos uma redução substancial no tráfego de dados na rede.

Por fim, também temos a possibilidade de agrupar resultados, semelhante a clausula GROUP BY no SQL. A projeção groupProperty faz isso de forma simples. Na **Listagem 19** temos um exemplo demonstrado o seu uso.

Listagem 19. Utilizando groupProperty() com Criteria

```
1 Criteria crit = session.createCriteria(Produto.class);
2 ProjectionList projList = Projections.projectionList();
3 projList.add(Projections.groupProperty("nome"));
4 projList.add(Projections.groupProperty("preco"));
5 crit.setProjection(projList);
6 List results = crit.list();
```

facilita bastante a descoberta de erros. Portanto, se possível prefira a utilização da API Criteria ao invés do HQL, essa API é simples e muito limpa.

Bibliografia

[1]Hibernate - JBoss Community, disponível em www.hibernate.org/

[2]Documentação de Referência Hibernate, disponível em https://docs.jboss.org/hibernate/core/3.6/reference/pt-BR/html/index.htmlhttp://www.hibernate.org/hib_docs/v3/reference/en/html/queryhql.html

[3] Introdução ao Hibernate, disponível em http://hibernate.javabeat.net/articles/2007/05/hibernate-orm-framework-introduction/

[4] Jeff Linwood and Dave Minter. An introduction to persistence using Hibernate 3.5, Second Edition. Apress.

[5] Steve Perkins. Hibernate Search by Example: Explore the Hibernate Search system and use its extraordinary search features in your own applications. Packt Publishing.

Tecnologias:

Hibernate Java