* connect with database and setup hibernate
* check one to one, one to many, many to many code on the github
* <https://github.com/Arthur-Shuahua-Zhang/Hibernate-Mappings>
* HQL
* Criteria query
* Native SQL

Hibernate provides three ways to retrieve data from database.

**Native SQL**

Hibernate provides a way to use native SQL statements directly through Hibernate.

Why use native SQL? – database supports some special features through its dialect of SQL that are not supported in HQL. Or you may want to call stored procedures from your Hibernate application.

Example:

|  |
| --- |
| String sql = "select avg(product.price) as avgPrice from Product product";  SQLQuery query = session.createSQLQuery(sql);  query.addScalar("avgPrice", Hibernate.DOUBLE);  List results = query.list(); |

Example: (returns a result set of objects)

String sql = "select {supplier.\*} from Supplier supplier";

SQLQuery query = session.createSQLQuery(sql);

query.addEntity("supplier", Supplier.**class**);

List results = query.list();

//Hibernate modifies the SQL and executes the following command against the database:

select Supplier.id as id0\_, Supplier.name as name2\_0\_ from Supplier supplier

**HQL (Hibernate Query Language)**

HQL is an object-oriented query language, working with persistent objects and their properties.

HQL queries are translated by Hibernate into conventional SQL queries, which in turns perform action on database.

In HQL, keywords like SELECT, FROM, and WHERE, etc., are not case sensitive, but properties like table and column names are case sensitive in HQL.

* FROM Clause

To load a complete persistent object into memory.

String hql = "FROM Employee";

Query query = session.createQuery(hql);

List results = query.list();

Or with package and class name:

String hql = "FROM com.hibernatebook.criteria.Employee";

Query query = session.createQuery(hql);

List results = query.list();

* AS Clause

To assign aliases to the classes in HQL queries, especially when you have the long queries.

String hql = "FROM Employee AS E";

Query query = session.createQuery(hql);

List results = query.list();

Also, the AS keyword is optional:

String hql = "FROM Employee E";

Query query = session.createQuery(hql);

List results = query.list();

* SELECT Clause

To obtain few properties of objects instead of the complete object.

String hql = "SELECT E.firstName FROM Employee E";

Query query = session.createQuery(hql);

List results = query.list();

Here, Employee.firstName is a property of Employee object rather than a field of the EMPLOYEE table.

* WHERE Clause

To narrow the specific objects that are returned from storage.

String hql = "FROM Employee E WHERE E.id = 10";

Query query = session.createQuery(hql);

List results = query.list();

* ORDER BY Clause

To sort your HQL query’s results.

String hql = "FROM Employee E WHERE E.id > 10 ORDER BY E.salary DESC";

Query query = session.createQuery(hql);

List results = query.list();

Sort by more than property:

String hql = "FROM Employee E WHERE E.id > 10 " +

"ORDER BY E.firstName DESC, E.salary DESC ";

Query query = session.createQuery(hql);

List results = query.list();

* GROUP BY Clause

To poll information from database and group it based on a value of an attribute and use the result to include an aggregate value.

String hql = "SELECT SUM(E.salary), E.firtName FROM Employee E " +

"GROUP BY E.firstName";

Query query = session.createQuery(hql);

List results = query.list();

* Using Named Parameters

Hibernate supports named parameters in its HQL queries.

String hql = "FROM Employee E WHERE E.id = :employee\_id";

Query query = session.createQuery(hql);

query.setParameter("employee\_id",10);

List results = query.list();

* UPDATE Clause

In Hibernate 3, it contains a method called executeUpdate() for executing HQL UPDATE or DELETE statements.

To update one or more properties of one or more objects.

String hql = "UPDATE Employee set salary = :salary " +

"WHERE id = :employee\_id";

Query query = session.createQuery(hql);

query.setParameter("salary", 1000);

query.setParameter("employee\_id", 10);

int result = query.executeUpdate();

System.out.println("Rows affected: " + result);

* DELETE Clause

To delete one or more objects.

String hql = "DELETE FROM Employee " +

"WHERE id = :employee\_id";

Query query = session.createQuery(hql);

query.setParameter("employee\_id", 10);

int result = query.executeUpdate();

System.out.println("Rows affected: " + result);

* INSERT INTO Clause

HQL supports INSERT INTO clause only where records can be inserted from one object to another object.

String hql = "INSERT INTO Employee(firstName, lastName, salary)" +

"SELECT firstName, lastName, salary FROM old\_employee";

Query query = session.createQuery(hql);

int result = query.executeUpdate();

System.out.println("Rows affected: " + result);

SQL directly works on databases through queries whereas HQL operates on objects and their properties.

**SQL vs. HQL**

|  |  |
| --- | --- |
| SQL | HQL |
| Concerns about the relation between two tables or columns | Concerns about the relation between two objects |
| Uses tables and columns | Uses JAVA classes and variables |
| Directly interacts with the database | Uses Hibernate interface to interact with the database |
| Fast | Non-native HQL is usually slower since its runtime is based on mapping, but its speed can be increased by setting the right cache size of query plan |

**Criteria Query**

The Criteria query API lets you build nested, structured query expressions in Java, providing a compile-time syntax checking that is not possible with a query language like HQL or SQL.

It also includes query by example (QBE) functionality.

How to use?

Pass the persistent object’s class or its entity name to the createCriteria() method, and hibernate will create a Criteria object that returns instances of the persistence object’s class when your application executes a criteria query.

|  |
| --- |
| Criteria crit = session.createCriteria(Product.**class**);  List<Product> results = crit.list(); |

* Restrictions

To retrieve objects that have a property value that “equals” your restriction, use Restrictions.eq()

Criteria crit = session.createCriteria(Product.**class**);

crit.add(Restrictions.eq("description","Mouse"));

List<Product> results = crit.list();

To retrieve objects that have a property value “not equal to” your restriction, use Restrictions.ne()

Criteria crit = session.createCriteria(Product.**class**);

crit.add(Restrictions.ne("description","Mouse"));

List<Product> results = crit.list()

To retrieve all objects that have a property matching part of a given pattern, use Restriction.like() or Restriction.ilike() where ilike() is case-insensitive.

Criteria crit = session.createCriteria(Product.**class**);

crit.add(Restrictions.like("name","Mou%",MatchMode.ANYWHERE));

List<Product> results = crit.list();

The example uses org.hibernate.criterion.MatchMode object to specify how to match the specified value to the stored date.

MatchMode object (a type-safe enumeration) has four matches:

ANYWHERE: Anyplace in the string

END: The end of the string

EXACT: An exact match

START: The beginning of the string

To do a search for objects that have or do not have null property values, use Restrictions.isNull() or Restrictions.isNotNull().

|  |
| --- |
| Criteria crit = session.createCriteria(Product.**class**);  crit.add(Restrictions.isNull("name"));  List<Product> results = crit.list(); |

To do math comparison, use Restrictions.gt(), Restrictions.ge(), Restrictions.lt(), and Restrictions.le()

|  |
| --- |
| Criteria crit = session.createCriteria(Product.**class**);  crit.add(Restrictions.gt("price", 25.0));  List<Product> results = crit.list(); |

To combine two or more Criteria:

1. AND – just add constraint to a criteria query

Criteria crit = session.createCriteria(Product.**class**);

crit.add(Restrictions.lt("price",10.0));

crit.add(Restrictions.ilike("description","mouse", MatchMode.ANYWHERE));

List<Product> results = crit.list();

1. OR – use or() method on the Restrictions class

|  |
| --- |
| Criteria crit = session.createCriteria(Product.**class**);  Criterion priceLessThan = Restrictions.lt("price", 10.0);  Criterion mouse = Restrictions.ilike("description", "mouse", MatchMode.ANYWHERE);  LogicalExpression orExp = Restrictions.or(priceLessThan, mouse);  crit.add(orExp);  List results=crit.list(); |

1. more than two OR – use Restrictions.disjunction()

Criteria crit = session.createCriteria(Product.**class**);

Criterion priceLessThan = Restrictions.lt("price", 10.0);

Criterion mouse = Restrictions.ilike("description", "mouse", MatchMode.ANYWHERE);

Criterion browser = Restrictions.ilike("description", "browser", MatchMode.ANYWHERE);

Disjunction disjunction = Restrictions.disjunction();

disjunction.add(priceLessThan);

disjunction.add(mouse);

disjunction.add(browser);

crit.add(disjunction);

List results = crit.list();

To directly specify SQL in the Criteria API. It’s useful if you need to use SQL clauses that Hibernate does not support through the Criteria API.

Criteria crit = session.createCriteria(Product.**class**);

crit.add(Restrictions.sqlRestriction("{alias}.description like 'Mou%'"));

List<Product> results = crit.list();

* uniqueResult()

obtain a single Object reference instead of a List. If more than one result, HiberateException

Criteria crit = session.createCriteria(Product.**class**);

Criterion price = Restrictions.gt("price",**new** Double(25.0));

crit.setMaxResults(1);

Product product = (Product) crit.uniqueResult();

* obtain distinct result

!!! unlike SELECT DISTINCT with SQL, the distinct result transformer compares each of your results using their default hashCode() methods, and only adds those results with unique hash codes to result set. Be careful!

Criteria crit = session.createCriteria(Product.**class**);

Criterion price = Restrictions.gt("price",**new** Double(25.0));

crit.setResultTransformer( DistinctRootEntityResultTransformer.INSTANCE )

List<Product> results = crit.list();

* sort

Criteria crit = session.createCriteria(Product.**class**);

crit.add(Restrictions.gt("price",10.0));

crit.addOrder(Order.desc("price"));

List<Product> results = crit.list();

* joins

going from either one-to-many or from many-to-one.

Criteria crit = session.createCriteria(Supplier.**class**);

Criteria prdCrit = crit.createCriteria("products");

prdCrit.add(Restrictions.gt("price",25.0));

List results = crit.list();

Criteria crit = session.createCriteria(Product.**class**);

Criteria suppCrit = crit.createCriteria("supplier");

suppCrit.add(Restrictions.eq("name","Hardware Are We"));

List results = crit.list();