转 hql 语法与详细解释

2013年10月10日 00:15:12　阅读数：1664

HQL查询：

Criteria查询对查询条件进行了面向对象封装，符合编程人员的思维方式，不过HQL(Hibernate Query Lanaguage)查询提供了更加丰富的和灵活的查询特性，因此Hibernate将HQL查询方式立为官方推荐的标准查询方式，HQL查询在涵盖Criteria查询的所有功能的前提下，提供了类似标准SQL语句的查询方式，同时也提供了更加面向对象的封装。完整的HQL语句形势如下：

Select/update/delete…… from …… where …… group by …… having …… order by …… asc/desc

其中的update/delete为Hibernate3中所新添加的功能，可见HQL查询非常类似于标准SQL查询。由于HQL查询在整个Hibernate实体操作体系中的核心地位，这一节我将专门围绕HQL操作的具体技术细节进行讲解。

1、 实体查询：

有关实体查询技术，其实我们在先前已经有多次涉及，比如下面的例子：

String hql="from User user ";

List list=session.CreateQuery(hql).list();

上面的代码执行结果是，查询出User实体对象所对应的所有数据，而且将数据封装成User实体对象，并且放入List中返回。这里需要注意的是，Hibernate的实体查询存在着对继承关系的判定，比如我们前面讨论映射实体继承关系中的Employee实体对象，它有两个子类分别是HourlyEmployee，SalariedEmployee,如果有这样的HQL语句：“from Employee”,当执行检索时Hibernate会检索出所有Employee类型实体对象所对应的数据（包括它的子类HourlyEmployee，SalariedEmployee对应的数据）。

因为HQL语句与标准SQL语句相似，所以我们也可以在HQL语句中使用where字句，并且可以在where字句中使用各种表达式，比较操作符以及使用“and”,”or”连接不同的查询条件的组合。看下面的一些简单的例子：

from User user where user.age=20;

from User user where user.age between 20 and 30;

from User user where user.age in(20,30);

from User user where user.name is null;

from User user where user.name like ‘%zx%’;

from User user where (user.age%2)=1;

from User user where user.age=20 and user.name like ‘%zx%’;

2、 实体的更新和删除：

在继续讲解HQL其他更为强大的查询功能前，我们先来讲解以下利用HQL进行实体更新和删除的技术。这项技术功能是Hibernate3的新加入的功能，在Hibernate2中是不具备的。比如在Hibernate2中，如果我们想将数据库中所有18岁的用户的年龄全部改为20岁，那么我们要首先将年龄在18岁的用户检索出来，然后将他们的年龄修改为20岁，最后调用Session.update()语句进行更新。在Hibernate3中对这个问题提供了更加灵活和更具效率的解决办法，如下面的代码：

Transaction trans=session.beginTransaction();

String hql="update User user set user.age=20 where user.age=18";

Query queryupdate=session.createQuery(hql);

int ret=queryupdate.executeUpdate();

trans.commit();

通过这种方式我们可以在Hibernate3中，一次性完成批量数据的更新，对性能的提高是相当的可观。同样也可以通过类似的方式来完成delete操作，如下面的代码：

Transaction trans=session.beginTransaction();

String hql="delete from User user where user.age=18";

Query queryupdate=session.createQuery(hql);

int ret=queryupdate.executeUpdate();

trans.commit();

如果你是逐个章节阅读的化，那么你一定会记起我在第二部分中有关批量数据操作的相关论述中，讨论过这种操作方式，这种操作方式在Hibernate3中称为bulk delete/update，这种方式能够在很大程度上提高操作的灵活性和运行效率，但是采用这种方式极有可能引起缓存同步上的问题(请参考相关论述)。

3、 属性查询：

很多时候我们在检索数据时，并不需要获得实体对象所对应的全部数据，而只需要检索实体对象的部分属性所对应的数据。这时候就可以利用HQL属性查询技术，如下面程序示例：

List list=session.createQuery(“select user.name from User user ”).list();

for(int i=0;i<list.size();i++){

System.out.println(list.get(i));

}

我们只检索了User实体的name属性对应的数据，此时返回的包含结果集的list中每个条目都是String类型的name属性对应的数据。我们也可以一次检索多个属性，如下面程序：

List list=session.createQuery(“select user.name,user.age from User user ”).list();

for(int i=0;i<list.size();i++){

Object[] obj=(Object[])list.get(i);

System.out.println(obj[0]);
System.out.println(obj[1]);
}
此时返回的结果集list中，所包含的每个条目都是一个Object[]类型，其中包含对应的属性数据值。作为当今我们这一代深受面向对象思想影响的开发人员，可能会觉得上面返回Object[]不够符合面向对象风格，这时我们可以利用HQL提供的动态构造实例的功能对这些平面数据进行封装，如下面的程序代码：
List list=session.createQuery("select new User(user.name,user.age) from User user ").list();
for(int i=0;i<list.size();i++){
User user=(User)list.get(i);
System.out.println(user.getName());
System.out.println(user.getAge());
}
这里我们通过动态构造实例对象，对返回结果进行了封装，使我们的程序更加符合面向对象风格，但是这里有一个问题必须注意，那就是这时所返回的User对象，仅仅只是一个普通的Java对象而以，除了查询结果值之外，其它的属性值都为null（包括主键值id），也就是说不能通过Session对象对此对象执行持久化的更新操作。如下面的代码：
List list=session.createQuery("select new User(user.name,user.age) from User user ").list();
for(int i=0;i<list.size();i++){
User user=(User)list.get(i);
user.setName("gam");
session.saveOrUpdate(user);//这里将会实际执行一个save操作，而不会执行update操作，因为这个User对象的id属性为null，Hibernate会把它作为一个自由对象（请参考持久化对象状态部分的论述），因此会对它执行save操作。
}
4、 分组与排序
A、Order by子句：
与SQL语句相似，HQL查询也可以通过order by子句对查询结果集进行排序，并且可以通过asc或者desc关键字指定排序方式，如下面的代码：
from User user order by user.name asc,user.age desc;
上面HQL查询语句，会以name属性进行升序排序，以age属性进行降序排序，而且与SQL语句一样，默认的排序方式为asc,即升序排序。
B、Group by子句与统计查询：
在HQL语句中同样支持使用group by子句分组查询，还支持group by子句结合聚集函数的分组统计查询，大部分标准的SQL聚集函数都可以在HQL语句中使用，比如：count(),sum(),max(),min(),avg()等。如下面的程序代码：
String hql="select count(user),user.age from User user group by user.age having count(user)>10 ";
List list=session.createQuery(hql).list();
C、优化统计查询：
假设我们现在有两张数据库表，分别是customer表和order表，它们的结构如下：
customer
ID varchar2(14)
age number(10)
name varchar2(20)

order
ID varchar2(14)
order_number number(10)
customer_ID varchar2(14)
现在有两条HQL查询语句，分别如下：
from Customer c inner join c.orders o group by c.age;(1)

select c.ID,c.name,c.age,o.ID,o.order_number,o.customer_ID
from Customer c inner join c.orders c group by c.age;(2)
这两条语句使用了HQL语句的内连接查询（我们将在HQL语句的连接查询部分专门讨论），现在我们可以看出这两条查询语句最后所返回的结果是一样的，但是它们其实是有明显区别的，语句（1）检索的结果会返回Customer与Order持久化对象，而且它们会被置于Hibernate的Session缓存之中，并且Session会负责它们在缓存中的唯一性以及与后台数据库数据的同步，只有事务提交后它们才会从缓存中被清除；而语句（2）返回的是关系数据而并非是持久化对象，因此它们不会占用Hibernate的Session缓存，只要在检索之后应用程序不在访问它们，它们所占用的内存就有可能被JVM的垃圾回收器回收，而且Hibernate不会同步对它们的修改。
在我们的系统开发中，尤其是Mis系统，不可避免的要进行统计查询的开发，这类功能有两个特点：第一数据量大；第二一般情况下都是只读操作而不会涉及到对统计数据进行修改，那么如果采用第一种查询方式，必然会导致大量持久化对象位于Hibernate的Session缓存中，而且Hibernate的Session缓存还要负责它们与数据库数据的同步。而如果采用第二种查询方式，显然就会提高查询性能，因为不需要Hibernate的Session缓存的管理开销，而且只要应用程序不在使用这些数据，它们所占用的内存空间就会被回收释放。
因此在开发统计查询系统时，尽量使用通过select语句写出需要查询的属性的方式来返回关系数据，而避免使用第一种查询方式返回持久化对象（这种方式是在有修改需求时使用比较适合），这样可以提高运行效率并且减少内存消耗。㊣真正的高手并不是精通一切，而是精通在合适的场合使用合适的手段。
5、 参数绑定：

Hibernate中对动态查询参数绑定提供了丰富的支持，那么什么是查询参数动态绑定呢？其实如果我们熟悉传统JDBC编程的话，我们就不难理解查询参数动态绑定，如下代码传统JDBC的参数绑定：

PrepareStatement pre=connection.prepare(“select * from User where user.name=?”);

pre.setString(1,”zhaoxin”);

ResultSet rs=pre.executeQuery();

在Hibernate中也提供了类似这种的查询参数绑定功能，而且在Hibernate中对这个功能还提供了比传统JDBC操作丰富的多的特性，在Hibernate中共存在4种参数绑定的方式，下面我们将分别介绍：

A、 按参数名称绑定：

在HQL语句中定义命名参数要用”:”开头，形式如下：

Query query=session.createQuery(“from User user where user.name=:customername and user.customerage=:age ”);

query.setString(“customername”,name);

query.setInteger(“customerage”,age);

上面代码中用:customername和:customerage分别定义了命名参数customername和customerage，然后用Query接口的setXXX()方法设定名参数值，setXXX()方法包含两个参数，分别是命名参数名称和命名参数实际值。

B、 按参数位置邦定：

在HQL查询语句中用”?”来定义参数位置，形式如下：

Query query=session.createQuery(“from User user where user.name=? and user.age =? ”);

query.setString(0,name);

query.setInteger(1,age);

同样使用setXXX()方法设定绑定参数，只不过这时setXXX()方法的第一个参数代表邦定参数在HQL语句中出现的位置编号（由0开始编号），第二个参数仍然代表参数实际值。

注：在实际开发中，提倡使用按名称邦定命名参数，因为这不但可以提供非常好的程序可读性，而且也提高了程序的易维护性，因为当查询参数的位置发生改变时，按名称邦定名参数的方式中是不需要调整程序代码的。

C、 setParameter()方法：

在Hibernate的HQL查询中可以通过setParameter()方法邦定任意类型的参数，如下代码：

String hql=”from User user where user.name=:customername ”;

Query query=session.createQuery(hql);

query.setParameter(“customername”,name,Hibernate.STRING);

如上面代码所示，setParameter()方法包含三个参数，分别是命名参数名称，命名参数实际值，以及命名参数映射类型。对于某些参数类型setParameter()方法可以更具参数值的Java类型，猜测出对应的映射类型，因此这时不需要显示写出映射类型，像上面的例子，可以直接这样写：

query.setParameter(“customername”,name);但是对于一些类型就必须写明映射类型，比如java.util.Date类型，因为它会对应Hibernate的多种映射类型，比如Hibernate.DATA或者Hibernate.TIMESTAMP。

D、 setProperties()方法：

在Hibernate中可以使用setProperties()方法，将命名参数与一个对象的属性值绑定在一起，如下程序代码：

Customer customer=new Customer();

customer.setName(“pansl”);

customer.setAge(80);

Query query=session.createQuery(“from Customer c where c.name=:name and c.age=:age ”);

query.setProperties(customer);

setProperties()方法会自动将customer对象实例的属性值匹配到命名参数上，但是要求命名参数名称必须要与实体对象相应的属性同名。

这里还有一个特殊的setEntity()方法，它会把命名参数与一个持久化对象相关联，如下面代码所示：

Customer customer=(Customer)session.load(Customer.class,”1”);

Query query=session.createQuery(“from Order order where order.customer=:customer ”);

query. setProperties(“customer”,customer);

List list=query.list();

上面的代码会生成类似如下的SQL语句：

Select * from order where customer_ID=’1’;

E、 使用绑定参数的优势：

我们为什么要使用绑定命名参数？任何一个事物的存在都是有其价值的，具体到绑定参数对于HQL查询来说，主要有以下两个主要优势：

①、 可以利用数据库实施性能优化，因为对Hibernate来说在底层使用的是PrepareStatement来完成查询，因此对于语法相同参数不同的SQL语句，可以充分利用预编译SQL语句缓存，从而提升查询效率。

②、 可以防止SQL Injection安全漏洞的产生：

SQL Injection是一种专门针对SQL语句拼装的攻击方式，比如对于我们常见的用户登录，在登录界面上，用户输入用户名和口令，这时登录验证程序可能会生成如下的HQL语句：

“from User user where user.name=’”+name+”’ and user.password=’”+password+”’ ”

这个HQL语句从逻辑上来说是没有任何问题的，这个登录验证功能在一般情况下也是会正确完成的，但是如果在登录时在用户名中输入”zhaoxin or ‘x’=’x”,

这时如果使用简单的HQL语句的字符串拼装，就会生成如下的HQL语句：

“from User user where user.name=’zhaoxin’ or ‘x’=’x’ and user.password=’admin’ ”;

显然这条HQL语句的where字句将会永远为真，而使用户口令的作用失去意义，这就是SQL Injection攻击的基本原理。

而使用绑定参数方式，就可以妥善处理这问题,当使用绑定参数时，会得到下面的HQL语句：

from User user where user.name=''zhaoxin'' or ''x=''x'' ‘ and user.password='admin';由此可见使用绑定参数会将用户名中输入的单引号解析成字符串（如果想在字符串中包含单引号，应使用重复单引号形式），所以参数绑定能够有效防止SQL Injection安全漏洞。

文章标签： hql 面向对象 安全漏洞 性能优化 数据库

个人分类： J2EE