**[Hibernate之HQL](http://www.blogjava.net/junglesong/archive/2008/04/30/197560.html)**

**HQL介绍**

Hibernate中不使用SQL而是有自己的面向对象查询语言,该语言名为Hibernate查询语言(Hibernate Query Language).HQL被有意设计成类似SQL,这样开发人员可以利用已有的SQL知识,降低学习坡度.它支持常用的SQL特性,这些特性被封装成面向对象的查询语言,从某种意义上来说,由HQL是面向对象的,因此比SQL更容易编写.  
本文将逐渐介绍HQL的特性.

**查询数据库中所有实例**

要得到数据库中所有实例,HQL写为”from 对象名”即可,不需要select子句,当然更不需要Where子句.代码如右.

Query query=session.createQuery("from User");   
List<User> users=(List<User>)query.list();

for(User user:users){  
     System.out.println(user);  
}

**限制返回的实例数**

设置查询的maxResults属性可限制返回的实例(记录)数,代码如右:

Query query=session.createQuery("from User order by name");   
query.setMaxResults(5);  
List<User> users=(List<User>)query.list();

System.out.println("返回的User实例数为"+users.size());  
for(User user:users){  
 System.out.println(user);  
}

**分页查询**

分页是Web开发的常见课题,每种数据库都有自己特定的分页方案,从简单到复杂都有.在Hibernate中分页问题可以通过设置firstResult和maxResult轻松的解决.

代码如右:

Query query=session.createQuery("from User order by name");   
query.setFirstResult(3);  
query.setMaxResults(5);  
List<User> users=(List<User>)query.list();

System.out.println("返回的User实例数为"+users.size());  
for(User user:users){  
      System.out.println(user);  
}

**条件查询**

条件查询只要增加Where条件即可.  
代码如右:

Hibernate中条件查询的实现方式有多种,这种方式的优点在于能显示完整的SQL语句(包括参数)如下.

select  
    user0\_.ID as ID0\_,  
    user0\_.name as name0\_,  
    user0\_.pswd as pswd0\_,  
    user0\_.email as email0\_,  
    user0\_.lastLoginTime as lastLogi5\_0\_,  
    user0\_.lastLoginIp as lastLogi6\_0\_   
from  
    USERTABLE\_OKB user0\_   
where  
    user0\_.name like '何%'

public static void fetchByName(String prefix){  
  Session session=HibernateUtil.getSession();  
    
  Query query=session.createQuery("from User where name like'"+prefix+"%'");   
  List<User> users=(List<User>)query.list();  
    
  System.out.println("返回的User实例数为"+users.size());  
  for(User user:users){  
    System.out.println(user);  
  }  
        
      HibernateUtil.closeSession(session);  
}

**位置参数条件查询**

HQL中也可以象jdbc中PreparedStatement一样为SQL设定参数,但不同的是下标从0开始.  
代码如右:

public static void fetchByPos(String prefix){  
  Session session=HibernateUtil.getSession();  
    
  Query query=session.createQuery("from User where name=?");   
  // 注意下标是从0开始,和jdbc中PreparedStatement从1开始不同  
  query.setParameter(0, prefix);  
  List<User> users=(List<User>)query.list();  
    
  System.out.println("返回的User实例数为"+users.size());  
  for(User user:users){  
    System.out.println(user);  
  }  
        
      HibernateUtil.closeSession(session);  
}

**命名参数条件查询**

使用位置参数条件查询最大的不便在于下标与?号位置的对应上,如果参数较多容易导致错误.这时采用命名参数条件查询更好.  
使用命名参数时无需知道每个参数的索引位置,这样就可以节省填充查询参数的时间.

如果有一个命名参数出现多次,那在每个地方都会设置它.

public static void fetchByNamedParam(){  
  Session session=HibernateUtil.getSession();  
    
  Query query=session.createQuery("from User where name=:name");   
  query.setParameter("name", "李白");  
  List<User> users=(List<User>)query.list();  
    
  System.out.println("返回的User实例数为"+users.size());  
  for(User user:users){  
    System.out.println(user);  
  }  
        
      HibernateUtil.closeSession(session);  
}

**命名查询**

命名查询是嵌在XML映射文件中的查询。通常，将给定对象的所有查询都放在同一文件中，这种方式可使维护相对容易些。命名查询语句写在映射定义文件的最后面。

执行代码如下：  
Session session=HibernateUtil.getSession();

Query query=session.getNamedQuery("user.sql");  
List<User> users=(List<User>)query.list();

System.out.println("返回的User实例数为"+users.size());  
for(User user:users){  
  System.out.println(user);  
}  
      
HibernateUtil.closeSession(session);

映射文件节选：  
<hibernate-mapping package="org.hibernate.auction">  
  <class name="com.sitinspring.domain.User"  
    table="USERTABLE\_OKB" lazy="false">  
    <id name="id" column="ID" >  
      <generator class="uuid.hex"/>  
    </id>  
    <property name="name" column="name" />  
    <property name="password" column="pswd" />  
    <property name="email" column="email" />  
    <property name="lastLoginTime" column="lastLoginTime" />  
    <property name="lastLoginIp" column="lastLoginIp" />  
  </class>  
    
  <query name="user.sql">  
    <![CDATA[from User where email='2@2.3']]>  
  </query>  
</hibernate-mapping>

|  |
| --- |
| HQL: Hibernate查询语言 |
| Hibernate配备了一种非常强大的查询语言，这种语言看上去很像SQL。但是不要被语法结构 上的相似所迷惑，HQL是非常有意识的被设计为完全面向对象的查询，它可以理解如继承、多态 和关联之类的概念。 第 15 章 HQL: Hibernate查询语言 Hibernate配备了一种非常强大的查询语言，这种语言看上去很像SQL。但是不要被语法结构 上的相似所迷惑，HQL是非常有意识的被设计为完全面向对象的查询，它可以理解如继承、多态 和关联之类的概念。 15.1. 大小写敏感性问题 除了Java类与属性的名称外，查询语句对大小写并不敏感。 所以 SeLeCT 与 sELEct 以及 SELECT 是相同的，但是 org.hibernate.eg.FOO 并不等价于org.hibernate.eg.Foo 并且 foo.barSet 也不等价于 foo.BARSET。  本手册中的HQL关键字将使用小写字母. 很多用户发现使用完全大写的关键字会使查询语句 的可读性更强, 但我们发现，当把查询语句嵌入到Java语句中的时候使用大写关键字比较难看。 15.2. from子句 Hibernate中最简单的查询语句的形式如下：  from eg.Cat  该子句简单的返回eg.Cat类的所有实例。 通常我们不需要使用类的全限定名, 因为 auto-import（自动引入） 是缺省的情况。 所以我们几乎只使用如下的简单写法：  from Cat  大多数情况下, 你需要指定一个别名, 原因是你可能需要 在查询语句的其它部分引用到Cat  from Cat as cat  这个语句把别名cat指定给类Cat 的实例, 这样我们就可以在随后的查询中使用此别名了。 关键字as 是可选的，我们也可以这样写:  from Cat cat  子句中可以同时出现多个类, 其查询结果是产生一个笛卡儿积或产生跨表的连接。  from Formula, Parameter  from Formula as form, Parameter as param  查询语句中别名的开头部分小写被认为是实践中的好习惯， 这样做与Java变量的命名标准保持了一致 (比如，domesticCat)。 15.3. 关联(Association)与连接(Join) 我们也可以为相关联的实体甚至是对一个集合中的全部元素指定一个别名, 这时要使用关键字join。  from Cat as cat  inner join cat.mate as mate  left outer join cat.kittens as kitten  from Cat as cat left join cat.mate.kittens as kittens  from Formula form full join form.parameter param  受支持的连接类型是从ANSI SQL中借鉴来的。   * inner join（内连接） * left outer join（左外连接） * right outer join（右外连接） * full join (全连接，并不常用)   语句inner join, left outer join 以及 right outer join 可以简写。  from Cat as cat  join cat.mate as mate  left join cat.kittens as kitten  还有，一个"fetch"连接允许仅仅使用一个选择语句就将相关联的对象或一组值的集合随着他们的父对象的初始化而被初始化，这种方法在使用到集合的情况下尤其有用，对于关联和集合来说，它有效的代替了映射文件中的外联接 与延迟声明（lazy declarations）. 查看 [第 20.1 节 “ 抓取策略(Fetching strategies) ”](http://www.hibernate.org/hib_docs/v3/reference/zh-cn/html/performance.html#performance-fetching) 以获得等多的信息。  from Cat as cat  inner join fetch cat.mate  left join fetch cat.kittens  一个fetch连接通常不需要被指定别名, 因为相关联的对象不应当被用在 where 子句 (或其它任何子句)中。同时，相关联的对象 并不在查询的结果中直接返回，但可以通过他们的父对象来访问到他们。  注意fetch构造变量在使用了scroll() 或 iterate()函数 的查询中是不能使用的。最后注意，使用full join fetch 与 right join fetch是没有意义的。  如果你使用属性级别的延迟获取（lazy fetching）（这是通过重新编写字节码实现的），可以使用 fetch all properties 来强制Hibernate立即取得那些原本需要延迟加载的属性（在第一个查询中）。  from Document fetch all properties order by name  from Document doc fetch all properties where lower(doc.name) like '%cats%' 15.4. select子句 select 子句选择将哪些对象与属性返 回到查询结果集中. 考虑如下情况:  select mate  from Cat as cat  inner join cat.mate as mate  该语句将选择mates of other Cats。（其他猫的配偶） 实际上, 你可以更简洁的用以下的查询语句表达相同的含义:  select cat.mate from Cat cat  查询语句可以返回值为任何类型的属性，包括返回类型为某种组件(Component)的属性:  select cat.name from DomesticCat cat  where cat.name like 'fri%'  select cust.name.firstName from Customer as cust  查询语句可以返回多个对象和（或）属性，存放在 Object[]队列中,  select mother, offspr, mate.name  from DomesticCat as mother  inner join mother.mate as mate  left outer join mother.kittens as offspr  或存放在一个List对象中,  select new list(mother, offspr, mate.name)  from DomesticCat as mother  inner join mother.mate as mate  left outer join mother.kittens as offspr  也可能直接返回一个实际的类型安全的Java对象,  select new Family(mother, mate, offspr)  from DomesticCat as mother  join mother.mate as mate  left join mother.kittens as offspr  假设类Family有一个合适的构造函数.  你可以使用关键字as给“被选择了的表达式”指派别名:  select max(bodyWeight) as max, min(bodyWeight) as min, count(\*) as n  from Cat cat  这种做法在与子句select new map一起使用时最有用:  select new map( max(bodyWeight) as max, min(bodyWeight) as min, count(\*) as n )  from Cat cat  该查询返回了一个Map的对象，内容是别名与被选择的值组成的名-值映射。 15.5. 聚集函数 HQL查询甚至可以返回作用于属性之上的聚集函数的计算结果:  select avg(cat.weight), sum(cat.weight), max(cat.weight), count(cat)  from Cat cat  受支持的聚集函数如下：   * avg(...), sum(...), min(...), max(...) * count(\*) * count(...), count(distinct ...), count(all...)   你可以在选择子句中使用数学操作符、连接以及经过验证的SQL函数：  select cat.weight + sum(kitten.weight)  from Cat cat  join cat.kittens kitten  group by cat.id, cat.weight  select firstName||' '||initial||' '||upper(lastName) from Person  关键字distinct与all 也可以使用，它们具有与SQL相同的语义.  select distinct cat.name from Cat cat  select count(distinct cat.name), count(cat) from Cat cat 15.6. 多态查询 一个如下的查询语句:  from Cat as cat  不仅返回Cat类的实例, 也同时返回子类 DomesticCat的实例. Hibernate 可以在from子句中指定任何 Java 类或接口. 查询会返回继承了该类的所有持久化子类 的实例或返回声明了该接口的所有持久化类的实例。下面的查询语句返回所有的被持久化的对象：  from java.lang.Object o  接口Named 可能被各种各样的持久化类声明：  from Named n, Named m where n.name = m.name  注意，最后的两个查询将需要超过一个的SQL SELECT.这表明order by子句 没有对整个结果集进行正确的排序. (这也说明你不能对这样的查询使用Query.scroll()方法.) 15.7. where子句 where子句允许你将返回的实例列表的范围缩小. 如果没有指定别名，你可以使用属性名来直接引用属性:  from Cat where name='Fritz'  如果指派了别名，需要使用完整的属性名:  from Cat as cat where cat.name='Fritz'  返回名为（属性name等于）'Fritz'的Cat类的实例。  select foo  from Foo foo, Bar bar  where foo.startDate = bar.date  将返回所有满足下面条件的Foo类的实例： 存在如下的bar的一个实例，其date属性等于 Foo的startDate属性。 复合路径表达式使得where子句非常的强大，考虑如下情况：  from Cat cat where cat.mate.name is not null  该查询将被翻译成为一个含有表连接（内连接）的SQL查询。如果你打算写像这样的查询语句  from Foo foo  where foo.bar.baz.customer.address.city is not null  在SQL中，你为达此目的将需要进行一个四表连接的查询。  =运算符不仅可以被用来比较属性的值，也可以用来比较实例：  from Cat cat, Cat rival where cat.mate = rival.mate  select cat, mate  from Cat cat, Cat mate  where cat.mate = mate  特殊属性（小写）id可以用来表示一个对象的唯一的标识符。（你也可以使用该对象的属性名。）  from Cat as cat where cat.id = 123  from Cat as cat where cat.mate.id = 69  第二个查询是有效的。此时不需要进行表连接！  同样也可以使用复合标识符。比如Person类有一个复合标识符，它由country属性 与medicareNumber属性组成。  from bank.Person person  where person.id.country = 'AU'  and person.id.medicareNumber = 123456  from bank.Account account  where account.owner.id.country = 'AU'  and account.owner.id.medicareNumber = 123456  第二个查询也不需要进行表连接。  同样的，特殊属性class在进行多态持久化的情况下被用来存取一个实例的鉴别值（discriminator value）。 一个嵌入到where子句中的Java类的名字将被转换为该类的鉴别值。  from Cat cat where cat.class = DomesticCat  你也可以声明一个属性的类型是组件或者复合用户类型（以及由组件构成的组件等等）。永远不要尝试使用以组件类型来结尾的路径表达式（path-expression） （与此相反，你应当使用组件的一个属性来结尾）。 举例来说，如果store.owner含有一个包含了组件的实体address  store.owner.address.city // 正确  store.owner.address // 错误!  一个“任意”类型有两个特殊的属性id和class, 来允许我们按照下面的方式表达一个连接（AuditLog.item 是一个属性，该属性被映射为<any>）。  from AuditLog log, Payment payment  where log.item.class = 'Payment' and log.item.id = payment.id  注意，在上面的查询与句中，log.item.class 和 payment.class 将涉及到完全不同的数据库中的列。 15.8. 表达式 在where子句中允许使用的表达式包括 大多数你可以在SQL使用的表达式种类:   * 数学运算符+, -, \*, / * 二进制比较运算符=, >=, <=, <>, !=, like * 逻辑运算符and, or, not * in, not in, between, is null, is not null, is empty, is not empty, member of and not member of * "简单的" case, case ... when ... then ... else ... end,和 "搜索" case, case when ... then ... else ... end * 字符串连接符...||... or concat(...,...) * current\_date(), current\_time(), current\_timestamp() * second(...), minute(...), hour(...), day(...), month(...), year(...), * EJB-QL 3.0定义的任何函数或操作：substring(), trim(), lower(), upper(), length(), locate(), abs(), sqrt(), bit\_length() * coalesce() 和 nullif() * cast(... as ...), 其第二个参数是某Hibernate类型的名字，以及extract(... from ...)，只要ANSI cast() 和 extract() 被底层数据库支持 * 任何数据库支持的SQL标量函数，比如sign(), trunc(), rtrim(), sin() * JDBC参数传入 ? * 命名参数:name, :start\_date, :x1 * SQL 直接常量 'foo', 69, '1970-01-01 10:00:01.0' * Java public static final 类型的常量 eg.Color.TABBY   关键字in与between可按如下方法使用:  from DomesticCat cat where cat.name between 'A' and 'B'  from DomesticCat cat where cat.name in ( 'Foo', 'Bar', 'Baz' )  而且否定的格式也可以如下书写：  from DomesticCat cat where cat.name not between 'A' and 'B'  from DomesticCat cat where cat.name not in ( 'Foo', 'Bar', 'Baz' )  同样, 子句is null与is not null可以被用来测试空值(null).  在Hibernate配置文件中声明HQL“查询替代（query substitutions）”之后， 布尔表达式（Booleans）可以在其他表达式中轻松的使用:  <property name="hibernate.query.substitutions">true 1, false 0</property>  系统将该HQL转换为SQL语句时，该设置表明将用字符 1 和 0 来 取代关键字true 和 false:  from Cat cat where cat.alive = true  你可以用特殊属性size, 或是特殊函数size()测试一个集合的大小。  from Cat cat where cat.kittens.size > 0  from Cat cat where size(cat.kittens) > 0  对于索引了（有序）的集合，你可以使用minindex 与 maxindex函数来引用到最小与最大的索引序数。 同理，你可以使用minelement 与 maxelement函数来 引用到一个基本数据类型的集合中最小与最大的元素。  from Calendar cal where maxelement(cal.holidays) > current date  from Order order where maxindex(order.items) > 100  from Order order where minelement(order.items) > 10000  在传递一个集合的索引集或者是元素集(elements与indices 函数) 或者传递一个子查询的结果的时候，可以使用SQL函数any, some, all, exists, in  select mother from Cat as mother, Cat as kit  where kit in elements(foo.kittens)  select p from NameList list, Person p  where p.name = some elements(list.names)  from Cat cat where exists elements(cat.kittens)  from Player p where 3 > all elements(p.scores)  from Show show where 'fizard' in indices(show.acts)  注意，在Hibernate3种，这些结构变量- size, elements, indices, minindex, maxindex, minelement, maxelement - 只能在where子句中使用。  一个被索引过的（有序的）集合的元素(arrays, lists, maps)可以在其他索引中被引用（只能在where子句中）：  from Order order where order.items[0].id = 1234  select person from Person person, Calendar calendar  where calendar.holidays['national day'] = person.birthDay  and person.nationality.calendar = calendar  select item from Item item, Order order  where order.items[ order.deliveredItemIndices[0] ] = item and order.id = 11  select item from Item item, Order order  where order.items[ maxindex(order.items) ] = item and order.id = 11  在[]中的表达式甚至可以是一个算数表达式。  select item from Item item, Order order  where order.items[ size(order.items) - 1 ] = item  对于一个一对多的关联（one-to-many association）或是值的集合中的元素， HQL也提供内建的index()函数，  select item, index(item) from Order order  join order.items item  where index(item) < 5  如果底层数据库支持标量的SQL函数，它们也可以被使用  from DomesticCat cat where upper(cat.name) like 'FRI%'  如果你还不能对所有的这些深信不疑，想想下面的查询。如果使用SQL，语句长度会增长多少，可读性会下降多少：  select cust  from Product prod,  Store store  inner join store.customers cust  where prod.name = 'widget'  and store.location.name in ( 'Melbourne', 'Sydney' )  and prod = all elements(cust.currentOrder.lineItems)  提示: 会像如下的语句  SELECT cust.name, cust.address, cust.phone, cust.id, cust.current\_order  FROM customers cust,  stores store,  locations loc,  store\_customers sc,  product prod  WHERE prod.name = 'widget'  AND store.loc\_id = loc.id  AND loc.name IN ( 'Melbourne', 'Sydney' )  AND sc.store\_id = store.id  AND sc.cust\_id = cust.id  AND prod.id = ALL(  SELECT item.prod\_id  FROM line\_items item, orders o  WHERE item.order\_id = o.id  AND cust.current\_order = o.id  ) 15.9. order by子句 查询返回的列表(list)可以按照一个返回的类或组件（components)中的任何属性（property）进行排序：  from DomesticCat cat  order by cat.name asc, cat.weight desc, cat.birthdate  可选的asc或desc关键字指明了按照升序或降序进行排序. 15.10. group by子句 一个返回聚集值(aggregate values)的查询可以按照一个返回的类或组件（components)中的任何属性（property）进行分组：  select cat.color, sum(cat.weight), count(cat)  from Cat cat  group by cat.color  select foo.id, avg(name), max(name)  from Foo foo join foo.names name  group by foo.id  having子句在这里也允许使用.  select cat.color, sum(cat.weight), count(cat)  from Cat cat  group by cat.color  having cat.color in (eg.Color.TABBY, eg.Color.BLACK)  如果底层的数据库支持的话(例如不能在MySQL中使用)，SQL的一般函数与聚集函数也可以出现 在having与order by 子句中。  select cat  from Cat cat  join cat.kittens kitten  group by cat  having avg(kitten.weight) > 100  order by count(kitten) asc, sum(kitten.weight) desc  注意group by子句与 order by子句中都不能包含算术表达式（arithmetic expressions）. 15.11. 子查询 对于支持子查询的数据库，Hibernate支持在查询中使用子查询。一个子查询必须被圆括号包围起来（经常是SQL聚集函数的圆括号）。 甚至相互关联的子查询（引用到外部查询中的别名的子查询）也是允许的。  from Cat as fatcat  where fatcat.weight > (  select avg(cat.weight) from DomesticCat cat  )  from DomesticCat as cat  where cat.name = some (  select name.nickName from Name as name  )  from Cat as cat  where not exists (  from Cat as mate where mate.mate = cat  )  from DomesticCat as cat  where cat.name not in (  select name.nickName from Name as name  )  在select列表中包含一个表达式以上的子查询，你可以使用一个元组构造符（tuple constructors）：  from Cat as cat  where not ( cat.name, cat.color ) in (  select cat.name, cat.color from DomesticCat cat  )  注意在某些数据库中（不包括Oracle与HSQL），你也可以在其他语境中使用元组构造符， 比如查询用户类型的组件与组合：  from Person where name = ('Gavin', 'A', 'King')  该查询等价于更复杂的：  from Person where name.first = 'Gavin' and name.initial = 'A' and name.last = 'King')  有两个很好的理由使你不应当作这样的事情：首先，它不完全适用于各个数据库平台；其次，查询现在依赖于映射文件中属性的顺序。 15.12. HQL示例 Hibernate查询可以非常的强大与复杂。实际上，Hibernate的一个主要卖点就是查询语句的威力。这里有一些例子，它们与我在最近的 一个项目中使用的查询非常相似。注意你能用到的大多数查询比这些要简单的多！  下面的查询对于某个特定的客户的所有未支付的账单，在给定给最小总价值的情况下，返回订单的id，条目的数量和总价值， 返回值按照总价值的结果进行排序。为了决定价格，查询使用了当前目录。作为转换结果的SQL查询，使用了ORDER, ORDER\_LINE, PRODUCT, CATALOG 和PRICE 库表。  select order.id, sum(price.amount), count(item)  from Order as order  join order.lineItems as item  join item.product as product,  Catalog as catalog  join catalog.prices as price  where order.paid = false  and order.customer = :customer  and price.product = product  and catalog.effectiveDate < sysdate  and catalog.effectiveDate >= all (  select cat.effectiveDate  from Catalog as cat  where cat.effectiveDate < sysdate  )  group by order  having sum(price.amount) > :minAmount  order by sum(price.amount) desc  这简直是一个怪物！实际上，在现实生活中，我并不热衷于子查询，所以我的查询语句看起来更像这个：  select order.id, sum(price.amount), count(item)  from Order as order  join order.lineItems as item  join item.product as product,  Catalog as catalog  join catalog.prices as price  where order.paid = false  and order.customer = :customer  and price.product = product  and catalog = :currentCatalog  group by order  having sum(price.amount) > :minAmount  order by sum(price.amount) desc  下面一个查询计算每一种状态下的支付的数目，除去所有处于AWAITING\_APPROVAL状态的支付，因为在该状态下 当前的用户作出了状态的最新改变。该查询被转换成含有两个内连接以及一个相关联的子选择的SQL查询，该查询使用了表 PAYMENT, PAYMENT\_STATUS 以及 PAYMENT\_STATUS\_CHANGE。  select count(payment), status.name  from Payment as payment  join payment.currentStatus as status  join payment.statusChanges as statusChange  where payment.status.name <> PaymentStatus.AWAITING\_APPROVAL  or (  statusChange.timeStamp = (  select max(change.timeStamp)  from PaymentStatusChange change  where change.payment = payment  )  and statusChange.user <> :currentUser  )  group by status.name, status.sortOrder  order by status.sortOrder  如果我把statusChanges实例集映射为一个列表（list）而不是一个集合（set）, 书写查询语句将更加简单.  select count(payment), status.name  from Payment as payment  join payment.currentStatus as status  where payment.status.name <> PaymentStatus.AWAITING\_APPROVAL  or payment.statusChanges[ maxIndex(payment.statusChanges) ].user <> :currentUser  group by status.name, status.sortOrder  order by status.sortOrder  下面一个查询使用了MS SQL Server的 isNull()函数用以返回当前用户所属组织的组织帐号及组织未支付的账。 它被转换成一个对表ACCOUNT,PAYMENT, PAYMENT\_STATUS, ACCOUNT\_TYPE, ORGANIZATION 以及 ORG\_USER进行的三个内连接， 一个外连接和一个子选择的SQL查询。  select account, payment  from Account as account  left outer join account.payments as payment  where :currentUser in elements(account.holder.users)  and PaymentStatus.UNPAID = isNull(payment.currentStatus.name, PaymentStatus.UNPAID)  order by account.type.sortOrder, account.accountNumber, payment.dueDate  对于一些数据库，我们需要弃用（相关的）子选择。  select account, payment  from Account as account  join account.holder.users as user  left outer join account.payments as payment  where :currentUser = user  and PaymentStatus.UNPAID = isNull(payment.currentStatus.name, PaymentStatus.UNPAID)  order by account.type.sortOrder, account.accountNumber, payment.dueDate 15.13. 批量的UPDATE & DELETE语句 HQL现在支持UPDATE与DELETE语句. 查阅 [第 14.3 节 “大批量更新/删除（Bulk update/delete）”](http://www.hibernate.org/hib_docs/v3/reference/zh-cn/html/batch.html#batch-direct) 以获得更多信息。 15.14. 小技巧 & 小窍门 你可以统计查询结果的数目而不必实际的返回他们：  ( (Integer) session.iterate("select count(\*) from ....").next() ).intValue()  若想根据一个集合的大小来进行排序，可以使用如下的语句：  select usr.id, usr.name  from User as usr  left join usr.messages as msg  group by usr.id, usr.name  order by count(msg)  如果你的数据库支持子选择，你可以在你的查询的where子句中为选择的大小（selection size）指定一个条件:  from User usr where size(usr.messages) >= 1  如果你的数据库不支持子选择语句，使用下面的查询：  select usr.id, usr.name  from User usr.name  join usr.messages msg  group by usr.id, usr.name  having count(msg) >= 1  因为内连接（inner join）的原因，这个解决方案不能返回含有零个信息的User 类的实例, 所以这种情况下使用下面的格式将是有帮助的:  select usr.id, usr.name  from User as usr  left join usr.messages as msg  group by usr.id, usr.name  having count(msg) = 0  JavaBean的属性可以被绑定到一个命名查询（named query）的参数上：  Query q = s.createQuery("from foo Foo as foo where foo.name=:name and foo.size=:size");  q.setProperties(fooBean); // fooBean包含方法getName()与getSize()  List foos = q.list();  通过将接口Query与一个过滤器（filter）一起使用，集合（Collections）是可以分页的：  Query q = s.createFilter( collection, "" ); // 一个简单的过滤器  q.setMaxResults(PAGE\_SIZE);  q.setFirstResult(PAGE\_SIZE \* pageNumber);  List page = q.list();  通过使用查询过滤器（query filter）可以将集合（Collection）的原素分组或排序:  Collection orderedCollection = s.filter( collection, "order by this.amount" );  Collection counts = s.filter( collection, "select this.type, count(this) group by this.type" );  不用通过初始化，你就可以知道一个集合（Collection）的大小：  ( (Integer) session.iterate("select count(\*) from ....").next() ).intValue(); |

### [hibernate------hql总结](http://kuangbaoxu.iteye.com/blog/193076)

**博客分类：**

* [hibernate](http://kuangbaoxu.iteye.com/category/31746)

[Hibernate](http://www.iteye.com/blogs/tag/Hibernate)[IDEA](http://www.iteye.com/blogs/tag/IDEA)

1. 查询整个映射对象所有字段

**Java代码  [收藏代码](javascript:void())**

1. //直接from查询出来的是一个映射对象，即：查询整个映射对象所有字段
2. String hql = "from Users";
3. Query query = session.createQuery(hql);
5. List<Users> users = query.list();
6. **for**(Users user : users){
7. System.out.println(user.getName() + " : " + user.getPasswd() + " : " + user.getId());
8. }
10. 输出结果为：
11. name1 : password1 : 1
12. name2 : password2 : 2
13. name3 : password3 : 3

2.查询单个字段

**Java代码  [收藏代码](javascript:void())**

1. //查询单个字段
2. String hql = " select name from Users";
3. Query query = session.createQuery(hql);
5. List<String> list = query.list();
6. **for**(String str : list){
7. System.out.println(str);
8. }
9. 输出结果为：
10. name1
11. name2
12. name3

3.查询其中几个字段 

**Java代码  [收藏代码](javascript:void())**

1. //查询其中几个字段
2. String hql = " select name,passwd from Users";
3. Query query = session.createQuery(hql);
4. //默认查询出来的list里存放的是一个Object数组
5. List<Object[]> list = query.list();
6. **for**(Object[] object : list){
7. String name = (String)object[0];
8. String passwd = (String)object[1];
10. System.out.println(name + " : " + passwd);
11. }
12. 输出结果为：
13. name1 : password1
14. name2 : password2
15. name3 : password3

4.修改默认查询结果(query.list())不以Object[]数组形式返回，以List形式返回

**引用**

//查询其中几个字段,添加new list(),注意list里的l是小写的。也不需要导入包，这样通过query.list()出来的list里存放的不再是默认的Object数组了，而是List集合了     
        String hql = " select new list(name,passwd) from Users";     
        Query query = session.createQuery(hql);     
        //默认查询出来的list里存放的是一个Object数组，但是在这里list里存放的不再是默认的Object数组了，而是List集合了     
        List<List> list = query.list();     
        for(List user : list){     
            String name = (String)user.get(0);     
            String passwd = (String)user.get(1);     
                 
            System.out.println(name + " : " + passwd);     
        }     
        /\*\*    
        输出结果为：    
         name1 : password1    
        name2 : password2    
        name3 : password3    
         \*/

5.修改默认查询结果(query.list())不以Object[]数组形式返回，以Map形式返回

**Java代码  [收藏代码](javascript:void())**

1. //查询其中几个字段,添加new map(),注意map里的m是小写的。也不需要导入包，这样通过query.list()出来的list里存放的不再是默认的Object数组了，而是map集合了
2. String hql = " select new map(name,passwd) from Users";
3. Query query = session.createQuery(hql);
4. //默认查询出来的list里存放的是一个Object数组，但是在这里list里存放的不再是默认的Object数组了，而是Map集合了
5. List<Map> list = query.list();
6. **for**(Map user : list){
7. //一条记录里所有的字段值都是map里的一个元素,key是字符串0,1,2,3....，value是字段值
8. //如果将hql改为：String hql = " select new map(name as username,passwd as password) from Users";,那么key将不是字符串0,1,2...了，而是"username","password"了
9. String name = (String)user.get("0");//get("0");是get(key),注意:0,1,2...是字符串，而不是整形
10. String passwd = (String)user.get("1");
12. System.out.println(name + " : " + passwd);
13. }
14. /\*\*
15. 输出结果为：
16. name1 : password1
17. name2 : password2
18. name3 : password3
19. \*/

6.修改默认查询结果(query.list())不以Object[]数组形式返回，以Set形式返回,但是因为Set里是不允许有重复的元素，所以:username和password的值不能相同。只需将hql改为：String hql = " select new set(name,passwd) from Users";   
  
7.修改默认查询结果(query.list())不以Object[]数组形式返回，以自定义类型返回   
  
自定义类： 

**Java代码  [收藏代码](javascript:void())**

1. **package** com.domain;
3. **public** **class** MyUser {
5. **private** String username;
6. **private** String password;
7. //因为：String hql = " select new  com.domain.MyUser(name,passwd) from Users";所以必须要有接受2个参数的构造函数
8. **public** MyUser(String username,String password){
9. **this**.username = username;
10. **this**.password = password;
11. }
13. **public** String getUsername() {
14. **return** username;
15. }
16. **public** **void** setUsername(String username) {
17. **this**.username = username;
18. }
19. **public** String getPassword() {
20. **return** password;
21. }
22. **public** **void** setPassword(String password) {
23. **this**.password = password;
24. }

27. }

**Java代码  [收藏代码](javascript:void())**

1. //通过query.list()出来的list里存放的不再是默认的Object数组了,而是自定义的类MyUser,必须加包名，String hql = "from Users";中的Users类也是必须加包名的，但是因为再Users.hbm.xml里<hibernate-mapping auto-import="true"> auto-import默认值为true（所以auto-import属性也可以不写）,自动导入了
2. String hql = " select new  com.domain.MyUser(name,passwd) from Users";
3. Query query = session.createQuery(hql);
4. //默认查询出来的list里存放的是一个Object数组，但是在这里list里存放的不再是默认的Object数组了，而是MyUser对象了
5. List<MyUser> myUsers = query.list();
6. **for**(MyUser myUser : myUsers){
7. String name = myUser.getUsername();
8. String passwd = myUser.getPassword();
9. System.out.println(name + " : " + passwd);
10. }
11. /\*\*
12. 输出结果为：
13. name1 : password1
14. name2 : password2
15. name3 : password3
16. \*/

8：条件查询

**Java代码  [收藏代码](javascript:void())**

1. //条件查询，参数索引值从0开始，索引位置。通过setString,setParameter设置参数
2. String hql = "from Users where name=? and passwd=?";
3. Query query = session.createQuery(hql);
4. //第1种方式
5. //      query.setString(0, "name1");
6. //      query.setString(1, "password1");
7. //第2种方式
8. query.setParameter(0, "name1",Hibernate.STRING);
9. query.setParameter(1, "password1",Hibernate.STRING);
10. List<Users> list = query.list();
11. **for**(Users users : list){
12. System.out.println(users.getId());
13. }

**Java代码  [收藏代码](javascript:void())**

1. //条件查询，自定义索引名(参数名):username,:password.通过setString,setParameter设置参数
2. String hql = "from Users where name=:username and passwd=:password";
3. Query query = session.createQuery(hql);
4. //第1种方式
5. //      query.setString("username", "name1");
6. //      query.setString("password", "password1");
7. //第2种方式,第3个参数确定类型
8. query.setParameter("username", "name1",Hibernate.STRING);
9. query.setParameter("password", "password1",Hibernate.STRING);
10. List<Users> list = query.list();
11. **for**(Users users : list){
12. System.out.println(users.getId());
13. }

**Java代码  [收藏代码](javascript:void())**

1. //条件查询,通过setProperties设置参数
2. String hql = "from Users where name=:username and passwd=:password";
3. Query query = session.createQuery(hql);
4. //MyUser类的2个属性必须和:username和:password对应
5. MyUser myUser = **new** MyUser("name1","password1");
6. query.setProperties(myUser);
7. List<Users> list = query.list();
8. **for**(Users users : list){
9. System.out.println(users.getId());
10. }