Hibernate SQL查询

hibernate除了支持HQL查询外，还支持原生SQL查询。对原生SQL查询执行的控制是通过SQLQuery接口进行的，通过执行Session.createSQLQuery()获取这个接口。该接口是Query接口的子接口。执行SQL查询步骤如下：

1. 获取Hibernate Session对象
2. 编写SQL语句
3. 通过Session的createSQLQuery方法创建查询对象
4. 调用SQLQuery对象的addScalar()或addEntity()方法将选出的结果与标量值或实体进行关联，分别用于进行标量查询或实体查询
5. 如果SQL语句包含参数，调用Query的setXxxx方法为参数赋值
6. 调用Query的list方法返回查询的结果集

# 标量查询

最基本的SQL查询就是获得一个标量的列表：

|  |
| --- |
| session.createSQLQuery("select \* from person\_inf").list();  session.createSQLQuery("select id,name,age from person\_inf").list();  它们都将返回一个Object数组组成的List，数组每个元素都是person\_inf表的一个字段值。Hibernate会使用ResultSetMetadata来判定返回的标量值的实际顺序和类型。但是在JDBC中过多的使用ResultSetMetadata会降低程序的性能。所以为了过多的避免使用ResultSetMetadata或者为了指定更加明确的返回值类型，我们可以使用addScalar()方法： |
| session.createSQLQuery("select \* from person\_inf")  .addScalar("name",StandardBasicTypes.STRING)  .addScalar("age",StandardBasicTypes.INT)  .list();  这个查询指定了：  1、SQL查询字符串。  2、要返回的字段和类型。  它仍然会返回Object数组,但是此时不再使用ResultSetMetdata,而是明确的将name和age按照String和int类型从resultset中取出。同时，也指明了就算query是使用\*来查询的，可能获得超过列出的这三个字段，也仅仅会返回这三个字段。如果仅仅只需要选出某个字段的值，而不需要明确指定该字段的数据类型，则可以使用addScalar(String columnAlias)。  实例如下：  public void scalarQuery(){  Session session = HibernateUtil.getSession();  Transaction tx = session.beginTransaction();  String sql = "select \* from person\_inf";  List list = session.createSQLQuery(sql).  addScalar("person\_id",StandardBasicTypes.INTEGER).  addScalar("name", StandardBasicTypes.STRING).  addScalar("age",StandardBasicTypes.INTEGER).list();  for(Iterator iterator = list.iterator();iterator.hasNext();){  //每个集合元素都是一个数组，数组元素师person\_id,person\_name,person\_age三列值  Object[] objects = (Object[]) iterator.next();  System.out.println("id="+objects[0]);  System.out.println("name="+objects[1]);  System.out.println("age="+objects[2]);  System.out.println("----------------------------");  }  tx.commit();  session.close();  }  从上面可以看出。标量查询中addScalar()方法有两个作用：  1. 指定查询结果包含哪些数据列---没有被addScalar选出的列将不会包含在查询结果中  2.指定查询结果中数据列的数据类型 |

# 实体查询

上面的标量查询返回的标量结果集，也就是从resultset中返回的“裸”数据。如果我们想要的结果是某个对象的实体，这是就可以通过addEntity()方法来实现。addEntity()方法可以讲结果转换为实体。但是在转换的过程中要注意几个问题：

* 查询返回的是某个数据表的全部数据列
* 该数据表有对应的持久化类映射

这时才可以通过addEntity()方法将查询结果转换成实体。

|  |
| --- |
| session.createSQLQuery("select \* from perons\_inf").addEntity(Person.class).list;  session.createSQLQuery("select id,name,age from person\_inf").addEntity(Person.class).list();  假设Person被映射为拥有id,name和age三个字段的类，以上的两个查询都返回一个List，每个元素都是一个Person实体。  假若实体在映射时有一个many-to-one的关联指向另外一个实体，在查询时必须也返回那个实体（获取映射的外键列），否则会导致发生一个"column not found"的数据库错误。这些附加的字段可以使用\*标注来自动返回，但我们希望还是明确指明，看下面这个具有指向teacher的many-to-one的例子：  sess.createSQLQuery("select id, name, age, teacherID from person\_inf").addEntity(Person.class).list();  public void entityQuery(){  Session session = HibernateUtil.getSession();  Transaction tx = session.beginTransaction();  String sql = "select \* from person\_inf";  List list = session.createSQLQuery(sql).  addEntity(Person.class). //指定将查询的记录行转换成Person实体  list();  for (Iterator iterator = list.iterator();iterator.hasNext();) {  //集合的每个元素都是一个Person对象  Person person = (Person) iterator.next();  System.out.println("name="+person.getName());  System.out.println("age="+person.getAge());  }  tx.commit();  session.close();  }  面的都是单表查询，如果我们在SQL语句中使用了多表连接，则SQL语句可以选出多个数据表的数据。Hibernate支持将查询结果转换成多个实体。如果要将查询结果转换成多个实体，则SQL字符串中应该为不同数据表指定不同别名，并且调用addEntity()方法将不同数据表转换成不同实体。如下  public void multiEntityQuery(){  Session session = HibernateUtil.getSession();  Transaction tx = session.beginTransaction();  String sql = "select p.\*,e.\* from person\_inf as p inner join event\_inf as e" +  " on p.person\_id=e.person\_id";  List list = session.createSQLQuery(sql)  .addEntity("p",Person.class)  .addEntity("e", MyEvent.class)  .list();  for(Iterator iterator = list.iterator();iterator.hasNext();){  //每个集合元素都是Person、MyEvent所组成的数组  Object[] objects = (Object[]) iterator.next();  Person person = (Person) objects[0];  MyEvent event = (MyEvent) objects[1];  System.out.println("person\_id="+person.getId()+" person\_name="+person.getName()+" title="+event.getTitle());    }  } |

# 处理关联和继承

通过提前抓取将Event连接获得，而避免初始化proxy带来的额外开销也是可能的。这是通过addJoin()方法进行的，通过这个方法可以讲实体的关联实体转换成查询对象。如下：

|  |
| --- |
| public void joinQuery(){  Session session = HibernateUtil.getSession();  Transaction tx = session.beginTransaction();  String sql = "select p.\*,e.\* from person\_inf as p,event\_inf as e where e.person\_id=p.person\_id";  List list = session.createSQLQuery(sql)  .addEntity("p",Person.class)  .addJoin("e","p.myEvents")  .list();  for(Iterator iterator = list.iterator();iterator.hasNext();){  //每个集合元素都是Person、MyEvent所组成的数组  Object[] objects = (Object[]) iterator.next();  Person person = (Person) objects[0];  MyEvent event = (MyEvent) objects[1];  System.out.println("person\_id="+person.getId()+" person\_name="+person.getName()+" title="+event.getTitle());    }  }  上面的程序返回的Person对象，其属性myEvent属性已经完全被初始化，不再需要数据库的额外操作，同时将该属性转换成别名为e的实体。也就是说返回的结果是Person、Event对象数组的列表。 |

# 命名查询

我们可以将SQL语句不放在程序中，而是放在配置文件中。这样可以更好地提高程序解耦。Hibernate使用<sql-query.../>元素来配置命名SQL查询，配置<sql-query.../>元素有一个必填的name属性，该属性用于指定该命名SQL查询的名称。使用<sql-query.../>元素定义命名查询，可以包含如下几个元素：

* <return.../>：将查询结果转换成持久化实体
* <return-join.../>：预加载持久化实体的关联实体
* <return-scalar.../>：将查询的数据列转换成标量值

在使用命名SQL查询时，不需要调用addEntity()、addScalar()等方法。因为在配置命名SQL查询时，已经指定了查询返回的结果信息。

|  |
| --- |
| <!-- 命名SQL查询 -->  <sql-query name="sqlquery">  <!-- 将p别名转换为Person实体 -->  <return alias="p" class="Person" />  <!-- 将e别名转换成Event实体 -->  <return alias="e" class="MyEvent" />  <!-- 指定将person\_inf表的name属性列作为标量值返回-->  <return-scalar column="p.name" type="string"/>  select p.\*,e.\* from person\_inf as p,event\_inf as e where p.person\_id = e.person\_id and p.age=:age  </sql-query>  使用Session的getNamedQuery即可获得指定命名sql查询 |
| public void query(){  Session session = HibernateUtil.getSession();  //调用命名查询，直接返回结果  List list = session.getNamedQuery("sqlquery")  .setInteger("age", 30).list();  for(Iterator iterator = list.iterator();iterator.hasNext();){  //每个集合元素都是Person、MyEvent所组成的数组  Object[] objects = (Object[]) iterator.next();  Person person = (Person) objects[0];  MyEvent event = (MyEvent) objects[1];  System.out.println("person\_id="+person.getId()+" person\_name="+person.getName()+" title="+event.getTitle());  }  session.close();  } |
| Hibernate允许把结果集的映射信息放在<resultset.../>元素定义，这样就可让多个命名查询共有该结果集映射。  <resultset name="person\_resultSet">  <return alias="p" class="Person" />  <return-scalar column="p.age" type="int"/>  </resultset>  通过为<sql-query.../>元素指定resultset-ref属性，就可以让命名SQL查询使用一个已有的结果集映射了。  <sql-query name="sqlquery" resultset-ref="person\_resultSet">  select p.\* from person as p  </sql-query> |

# 调用存储过程

Hibernate可以通过命名SQL查询来调用存储过程或者函数。对于函数，该函数必须返回一个结果集；对于存储过程，该存储过程的第一个参数必须传出参数，且数据类型是结果集。下面是一个简单的存储过程：

|  |
| --- |
| Create procedure select\_person()  select \* from person\_inf;  如果需要使用该存储过程，可以先将其定义成命名SQL查询，然后在程序中使用。  当使用原生SQL来调用存储过程，应该为<sql-query.../>元素指定callable="true"  <sql-query name="callProcedure" callable="true">  <return class="Person">  <!-- 将查询的数据列转换成实体的属性 -->  <return-property name="name" column="name"/>  <return-property name="age" column="age" />  <return-property name="person\_id" column="id" />  </return>  </sql-query> |

程序与上面相同。调用存储过程需要注意以下几个问题：

为了在Hibernate中使用存储过程,你必须遵循一些规则.不遵循这些规则的存储过程将不可用.如果你仍然想要使用他们, 你必须通过session.connection()来执行他们.这些规则针对于不同的数据库.因为数据库 提供商有各种不同的存储过程语法和语义.对存储过程进行的查询无法使用setFirstResult()/setMaxResults()进行分页。建议采用的调用方式是标准SQL92: { ? = call functionName(<parameters>) } 或者 { ? = call procedureName(<parameters>}.原生调用语法不被支持。

对于Oracle有如下规则:函数必须返回一个结果集。存储过程的第一个参数必须是OUT，它返回一个结果集。这是通过Oracle 9或10的SYS\_REFCURSOR类型来完成的。在Oracle中你需要定义一个REF CURSOR类型，参见Oracle的手册。

对于Sybase或者MS SQL server有如下规则:存储过程必须返回一个结果集。.注意这些servers可能返回多个结果集以及更新的数目.Hibernate将取出第一条结果集作为它的返回值， 其他将被丢弃。

如果你能够在存储过程里设定SET NOCOUNT ON，这可能会效率更高，但这不是必需的。