Hibernate条件查询

条件查询一般是通过以下三个类完成的：

1. Criteria:代表一次查询
2. Criterion：代表一个查询条件
3. Restriction：代表查询条件的工具类

执行条件查询的步骤如下：

1. 获得hibernate的Session对象
2. 以Session对象创建Criteria对象
3. 使用Restriction的静态方法创建Criterion查询条件
4. 向Criteria查询中添加Criterion查询条件
5. 执行Criterion的list()方法或者uniqueResult()方法返回结果集

|  |
| --- |
| public void query(){  Session session = HibernateUtil.getSession();  Transaction tx = session.beginTransaction();  //使用ctiteria进行条件查询  List list = session.createCriteria(Person.class)  .add(Restrictions.eq("id", 1))  .list();  for (Iterator iterator = list.iterator();iterator.hasNext();) {  Person person = (Person) iterator.next();  System.out.println(person.getName());  }  }  Criteria对象本身并不具备任何数据过滤筛选功能，但程序可以通过向Criteria对象中组合多个Criterion（每一个Criterion对象代表了一个数据过滤条件）即可实现数据过滤。Criteria包含如下两个方法：  setFristResult(int firstResult):设置查询返回的第一行记录  setMaxResult(int maxResult):设置查询返回的记录数  这两个方法用于查询分页  Criteria还包含如下几个常用的方法：  add(Criterion criterion)：增加查询条件  addOrder(Order order)：增加排序规则  List list = session.createCriteria(Person.class)  .add(Restrictions.like("name", "李%")) //增加查询条件  .addOrder(Order.desc("name")) //结果集排序  .setMaxResults(50) //返回的记录数最多为50条  .list(); //返回结果集  Criterion接口代表了一个查询条件，该查询条件有Restrictions负责产生。Restrictions是专门用于产生查询条件的工具类，它的方法大部分是静态方法。 |

# 关联

如果需要使用关联实体的属性来增加查询条件，则应该对该属性再次使用createCriteria方法。

|  |
| --- |
| List list = session.createCriteria(Person.class)  .createAlias("myEvent", "e")  .add(Restrictions.like("e.title", "工业大学%"))  .list();  第二个createCriteria返回一个新的Criteria实例，该实例引用Person类中的myEvent关联属性。title是MyEvent类中的属性。  接下来，替换形态在某些情况下也是很有用的。  List list = session.createCriteria(Person.class)  .add(Restrictions.like("name", "李%"))  .setFetchMode("myEvent", FetchMode.JOIN)  .list();  createAlias()方法并不创建一个新的Criteria实例，它只是给关联实体起一个别名，让后面的过滤条件可根据该关联实体进行筛选。 |

# 动态关联

在默认情况下，条件查询将根据映射文件指定的延迟加载策略来加载关联实体，如果希望在条件查询中改变延迟加载策略，可以通过setFetchMode()方法来控制。setFetchMode()方法接受一个FetchMode参数。

* DEFAULT：使用配置文件指定的延迟加载策略处理
* JOIN：使用外连接、预初始化所有关联实体
* SELECT：启用延迟加载，系统将使用单独的select语句来初始化关联实体。只有当真正访问关联实体的时候，才会执行第二条select语句。

使用外连接方式抓取Myevent。

# 投影、聚合、分组

Hibernate的条件过滤中使用Projection代表投影运算，Projection是一个接口，而Projections作为Projection的工厂，负责生成Projection对象。一旦产生了Projection对象之后，就可以通过setProjection(Projection projection)方法来进行投影运算了。

|  |
| --- |
| List list = session.createCriteria(Person.class)  .setProjection(Projections.projectionList()  .add( Projections.avg("age"))  .add(Projections.groupProperty("name")))  .list();  在一个条件查询中没有必要显式的使用 "group by" 。某些投影类型就是被定义为分组投影，他们也出现在SQL的group by子句中。  如果我们希望对分组后属性进行排序，那就需要为投影运算指定一个别名了，有两种方法可以为投影运算指定别名  1、使用alias()方法  List list = session.createCriteria(Person.class)  .setProjection(Projections.alias(Projections.groupProperty("name"), "name"))  .addOrder(Order.asc("age"))  .list();  使用as()方法为自身指定别名  List list = session.createCriteria(Person.class)  .setProjection(Projections.groupProperty("name").as("name"))  .addOrder(Order.asc("age"))  .list();  也可以使用Property.forName()来表示投影：  List list = session.createCriteria(Person.class)  .setProjection(Projections.projectionList()  .add(Property.forName("name")))  .list(); |

# 离线查询和子查询

条件查询的离线查询有DetachedCriteria来代表，DetachedCriteria类允许在一个session范围外创建一个 查询，并且可以使用任意Session来执行它。

|  |
| --- |
| public void detachedCriteriaTest(){  //定义一个离线查询  DetachedCriteria query = DetachedCriteria.forClass(Person.class).setProjection(Property.forName("name"));  Session session = HibernateUtil.getSession();  Transaction tx = session.beginTransaction();  //执行离线查询  List list = query.getExecutableCriteria(session).list();    for (Iterator iterator = list.iterator();iterator.hasNext();) {  Person person = (Person) iterator.next();  System.out.println(person.getName());  }  }  另外DetachedCriteria还可以代表子查询，当我们把DetachedCriteria传入Criteria中作为查询条件时，DetachedCriteria就变成了子查询。  public void subQuery(){  //定义一个离线查询  DetachedCriteria query = DetachedCriteria.forClass(Person.class).setProjection(Property.forName("name"));  Session session = HibernateUtil.getSession();  Transaction tx = session.beginTransaction();  //执行子查询  List list = session.createCriteria(Person.class)  .add(Property.forName("name").in(query))  .list();  for (Iterator iterator = list.iterator();iterator.hasNext();) {  Person person = (Person) iterator.next();  System.out.println(person.getName());  }  } |