OJB中的多表查询和更新 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/145/2021_2022_OJB_E4_B8_AD_E7_9A_84_E5_c104_145042.htm OJB的文章实在是少的可怜，自己总结了一下一些重点的内容。想到哪里写到哪里吧比如有课程(event)和类别(catalog)两张表，二者是多对多的关系，关系表叫做catalogEventBridge，字段为[catalogID,eventID]，现在要做的是一旦查询出一个catalog，就要把相关的event也全部搜索出来。1.repository_user.xml中作如下定义： primarykey="true" /> jdbc-type="VARCHAR" /> class-ref="org.pie.vls.Catalog.CatalogEventBridge" auto-retrieve="true" auto- 0update="true" auto-0delete="true"> class="org.pie.vls.Catalog.CatalogEventBridge" table="catalogEventBridge"> primarykey="true" /> jdbc-type="INTEGER" primarykey="true" /> class="org.pie.vls.EventType.EventType" table="eventType" > name="eventTypeID" column="eventTypeID" jdbc-type="INTEGER" primarykey="true" autoincrement="true" /> name="eventTypeTitle" column="eventTypeTitle" jdbc-type="VARCHAR" /> ... ... 2. bean文件的定义EventType.javapublic class EventType implements Cloneable, Comparable { private String eventTypeTitle. private String eventTypeCode. private String eventTypeDesc. private int eventTypeID. public int getEventTypeID() { return this.eventTypeID. } public void setEventTypeID(int value) { this.eventTypeID = value. } ... ... } catalog.javapublic class Catalog

implements Comparable { private int catalogID. private String catalogName = "". private List catalogEventList = new Vector(). /** * @return Returns the catalogID. */ public int getCatalogID() { return catalogID. } /** * @param catalogID The catalogID to set. */ public void setCatalogID(int catalogID) { this.catalogID = catalogID. } /** * @return Returns the catalogName. */ public String getCatalogName() { return catalogName. } /** * @param catalogName The catalogName to set. */ public void setCatalogName(String catalogName) { this.catalogName = catalogName. } /** * @return Returns the catalogEventList. */ public List getCatalogEventList() { return catalogEventList. } /** * @param catalogEventList The catalogEventList to set. */ public void setCatalogEventList(List catalogEventList) { this.catalogEventList = catalogEventList. } } CatalogEventBridge.javapublic class CatalogEventBridge { private int catalogID = 0. private int eventTypeID = 0. /** * @return Returns the catalogID. */ public int getCatalogID() { return catalogID. } /** * @param catalogID The catalogID to set. */ public void setCatalogID(int catalogID) { this.catalogID = catalogID. } /** * @return Returns the eventTypeID. */ public int getEventTypeID() { return eventTypeID. } /** * @param eventTypeID The eventTypeID to set. */ public void setEventTypeID(int eventTypeID) { this.eventTypeID = eventTypeID. } } 3. 定义DAOimport java.util.Collection. import java.util.Iterator. import org.apache.log4j.Logger. import org.apache.ojb.broker.PersistenceBroker. import org.apache.ojb.broker.PersistenceBrokerFactory. import

org.apache.ojb.broker.query.Criteria. import org.apache.ojb.broker.query.QueryByCriteria. import org.apache.ojb.broker.query.QueryFactory. import org.odmg.Implementation. import org.pie.vls.Application.AbstractVLSBase. public class CatalogDAO extends AbstractVLSBase { private Collection catalogList. private PersistenceBroker broker. private static final Logger logger = Logger.getLogger(Catalog.class). private int catalogID = 0. public CatalogDAO(Implementation impl) { super(impl). init(). } public CatalogDAO(Implementation impl, int catalogID) { super(impl). this.catalogID = catalogID. init(). } /* (non-Javadoc) * @see org.pie.vls.Application.VLSBase#init() */ public void init() { // TODO Auto-generated method stub try { broker = PersistenceBrokerFactory.defaultPersistenceBroker(). broker.clearCache(). Criteria crit = new Criteria(). if (this.catalogID > 0) { crit.addEqualTo("catalogID", new Integer(this.catalogID)). } QueryByCriteria tquery = QueryFactory.newQuery(Catalog.class, crit). this.catalogList = broker.getCollectionByQuery(tquery). broker.clearCache(). broker.close(). } catch (Throwable t) { logger.error(t.getMessage(), t). } } public boolean isEmpty(){ return this.catalogList.isEmpty(). } public Collection getCatalogList() { return this.catalogList. } public Catalog getCatalog() { Iterator iter = this.catalogList.iterator(). Catalog catalog = (Catalog) iter.next(). return catalog. } public void create(Catalog catalog) { 0update(catalog). } public void 0update(Catalog catalog) { try { broker = PersistenceBrokerFactory.defaultPersistenceBroker().

broker.clearCache(). broker.beginTransaction(). broker.store(catalog). broker.commitTransaction(). broker.close(). } catch (Throwable t) { logger.error(t.getMessage(), t). } } public void 0delete(Catalog catalog) { try { broker = PersistenceBrokerFactory.defaultPersistenceBroker(). broker.clearCache(). broker.beginTransaction(). broker.0delete(catalog). broker.commitTransaction(). broker.close(). } catch (Throwable t) { logger.error(t.getMessage(), t). } } } catalogEventBridgeDAO.java和eventTypeDAO.java跟上面的这个DAO类似，就不再重复。 使用方法就很简单了，只要通过CatalogDAO得到了一个catalog的实例，那么就可以通过catalog.getCatalogEventList()来获得bridge表中相关的数据，同样，调用catalogDAO中的create,0update和0delete方法，不仅对catalog表作操作，同样也对catalogEventBridge表中的数据作操作。