ARE 程序开发指南

JBO 参考手册

上海安硕信息技术有限公司

2011 年 11 月

上海安硕信息技术有限公司授权:

THESE LISTED PEOPLE:

上海安硕信息技术股份有限公司及其下属公司之技术人员

使用本文件

ARE AUTHORIZED BY AMARSOFT TECHNOLOGY CO., LTD. TO USE THIS DOCUMENT

文档信息

文档名称：	ARE 程序开发指南-JBO 参考手册
初稿作者：	翟涛
初稿日期：	2011-2-23
内容概述：	本参考手册介绍如何使用 ARE 中 JBO 的进行应用程序开发，包括 JBO 基本概念，使用 API，环境配置，查询语法和一般扩展原理。本手册主要面向使用 JBO 的应用开发，主要从应用角度介绍如何使用 JBO。

修订历史

版本	修订日期	修订人	复核日期	复核人	修改内容简述
1.0	2011-2-23	翟涛			完成基本文档
1.1	2011-4-6	翟涛			增加事务控制，Function 引入，虚拟字段等新增功能说明
1.2	2011-9-8	翟涛			增加查询语法中虚拟属性和扩展属性的 label 支持
1.3	2011-11-8	翟涛			修正一些文字错误，纠正文档格式问题。补充第 5 章基础实现特性、第 6 章环境配置

发布历史

发布日期	提交人	质检人	发布说明

密级：

目 录

密级：

1 文档说明

阅读本文档时请注意，除非特别声明，本文档出现的以下名词均指上海安硕信息技术股份有限公司，这些名词包括：

- 上海安硕信息技术股份有限公司
- 上海安硕信息技术有限公司
- 安硕信息
- 安硕软件
- 安硕科技
- 安硕
- Amarsoft Technology Co., Ltd.
- Amarsoft

1.1 文档概述

本文是 Java 业务对象（Java Business Object）管理和服务技术使用参考文档。本文档详细介绍 JBO 在应用开发中的使用，通过本文档应用开发人员可以学习到从 JBO 运行环境到程序开发所需要的基本知识，并能在此基础上进行基于 JBO 的 Java 程序开发。

本文档面向的作者是应用程序开发人员，重点介绍的是如何使用 JBO 技术开发应用程序，对面向 JBO 底层实现开发者可参考其他技术文档。本文档从面向应用开发的角度组织，对具体的 API 细节和不重要的 API 可参考 Java DOC 的 API 文档。

本文完整包括 6 章介绍 JBO 的使用：

- 第一章：文档说明。即本章，说明本文档的基本信息；
- 第二章：JBO 概述。介绍 JBO 基本技术概念和架构以及和其他技术的关系；

- 第三章：应用入门。说明如何在应用程序中使用 JBO 的基本技术实现应用功能；
- 第四章：查询语法。详细说明 JBO-QL 的语法；
- 第五章：基础实现特性。介绍标准 JBO 实现包含的特性；
- 第六章：环境配置。介绍如何配置 JBO；

1.2 名词定义

本节定义本文中出现的可能引起混淆，或者不宜理解的技术名词。

- ARE。Amarsoft Java 应用程序运行时刻环境（Amarsoft Runtime Environment），安硕基于 Java 技术的基本程序运行环境，提供基本程序库和服务支持；
- JBO。Java Business Object 缩写，安硕公司一种用于表示业务对象的技术，JBO 缩写代表了这种对象模型和相关技术，JBO 技术是 ARE 的一部分；
- JBO 对象和类，JBO 类是 JBO 实体对象定义，JBO 对象是实体对象的实例，在不混淆的情况下，本文中经常混用；
- JBO 配置文件。配置 JBO 使用的文件，本文特指 ARE 缺省实现的 JBO 管理器相关 XML 配置文件。

1.3 相关资料

阅读本文时可参考下面的资料：

《ARE 程序开发指南-环境管理手册》

《ARE 程序开发指南-基本应用参考手册》

《ARE 程序开发指南-JAVA API》(HTML 格式 Java Doc)

1.4 示例程序

本文中所有引用的例子程序或者程序片段均来自 ARE 工程的测试和实例程序，即该工程的 test_src 和 demo 部分。所用数据库为 ARE 开发、测试、DEMO 数据库 sample_db，在 ARE 开发环境中有创建数据库的脚本和加载测试数据的程序。

所用例子和查询语句均在 mysql5.0 测试数据库上验证。

序号	技术标准	版本	说明
1	J2SE	1.4.2	本系统开发使用 Java 2 Standard Edition 平台
2	AmarTech	2.0	本产品开发基于新的技术架构，不依赖原有的 com.amarsoft 类包。
3	JDBC	2.0 Type 4	本产品中对数据库的连接使用 JDBC2.0 技术规范，并且推荐使用 Type4 的驱动。
4	Language	GBK	本程序的假定标准语言使用 GBK

2 JBO 概述

2.1 技术背景

Java 作为面向对象的编程语言，用对象化语义来表现业务对象模型是最直接自然的方式，而用于业务对象持久化的最重要的方式是关系数据库，在做业务模型设计时会使用数据库的语义和模型来进行。 这就造成了开发 Java 应用程序中必须面临两个技术领域：基于 Java 对象模型和基于关系数据库的数据模型。应用程序必须完成两种模型的相互转换，早期的 Java 程序只能通过 JDBC 完成所有的数据和 Java 数据模型的转换，这些重复性的程序代码费时费力又容易出错，于是出现了 O-R Map 的概念，其核心思想就是把枯燥、机械却极易出错的 Java 对象持久化管理从一般应用程序中剥离出来形成基础的技术平台，以便于应用把精力集中于应用逻辑。

最初， J2EE 中引入了 EJB 的实体 Bean（Entity Bean）技术模型来解决这一问题，实体 Bean 被定义为数据库记录的 Java 包装，是一种特殊的 Java 类，定义了明确的持久化管理接口，明显区分出了一般意义的 Java 对象和面向应用的业务对象。不幸的是 EJB 实现实在是过于复杂，对应用来说是极大的负担，虽然是官方标准，但一直没有广泛应用。

由于重型 EJB 使用困难性，在实际开发中逐渐形成了以轻量的普通 Java 对象（POJO）作为业务对象载体的流派，如果说 EJB 是从规范和技术上确定了业务实体对象的载体，而 POJO 则是更多是从概念上去认定业务实体对象，因为它们和普通的 Java 对象并无技术上区分。以 Hibernate 为代表的轻量化 O-R Map 实现取得极大成功，成为事实上标准。顺应自然潮流，在 Java EE5 中，放弃了 EJB 的套路，以官方形式推出了 JPA 规范，以 POJO 作为业务对象载体，并充分利用 Java5 之后推出的 Annotation 技术，极大简化了 Java EE 开发。至此，Java 对象持久化或者说 O-R Map 的技术模型和规范终于统一。

然而，纵观这些技术模型，其核心是把 Java 对象和关系数据库建立对象关系，并建立统一管理平台。这在实际应用体系中依然存在不理想之处：

1. 削弱了业务对象的广泛性。过于强调了了 Object 和关系数据库的，几乎忽略了对象持久化的多样性；

2. 过于强调了 O-R 映射的共性，限制了个性。从一定程度上限制了应用对业务对象的个性化管理需求实现， JPA 取代了 EJB 后，更是难以实现应用级别的持久化管理；

3. 业务含义表达不充分。以简单 Java 类来代表业务对象损失了很多业务特性，无法完全体现业务对象的业务含义。这一点比之数据库技术尚且不如，数据库还可以通过 MetaData 获取更多信息。这在一定程度上为应用层开发带来难度，比如界面生成自动化、工具支持等，这在非英语环境下尤其突出；

4. 业务对象固化为 Java 类之后缺少灵活性。Java 是静态编译的语言，在实际应用中业务对象属性增减、变化非常频繁，尤其是管理型应用系统不可能完全固化业务模型，此时要不断的修改 Java 程序、编译、发布非常不便；

5. Java 对象操作复杂。一旦固化为 Java 类，对象的属性只能通过 Java 的 get/set 属性方法实现。在一些场景下，需要频繁、大量使用属性时会非常不便，要不断用到反射等高级编程技巧，而且属性的加减改变都会引起调用程序的改动。

2.2 概念说明

为解决 JPA 技术在实际应用中无法完全满足表达业务对象管理的问题，我们需要建立一种更为实用的业务对象管理体系，要求达到以下目标：

1. 以业务对象为核心理念，突出业务特性，而不是技术特性；

2. 平衡的模型特征，模型特征介于 Java 类和数据库之间，程序员、数据库设计人员、业务人员都易于理解和使用；

3. Java 程序亲和性，业务对象便于被 Java 程序操作，便于被程序员理解，接近于 JPA 和数据库的技术观念和语义、语法，能够比 Java 类更容易被应用获取内部结构和信息；

4. 有较强的业务表达能力。提供不低于数据库模型的业务表达能力，提供充分的业务元数据，并有较好的扩展性；

5. 灵活、易于扩展的持久化管理框架。80%应用场景下可用基本技术实现直接支持，对特别场景提供易于扩展的机制；

6. 工具友好性。便于工具进行管理、设计和使用。

A3 体系中在 ARE 级别建立了业务对象管理模型，希望能够达到上述目标。这个模型定名为 Java Business Object，简称 JBO，包括一系列概念定义和技术实现。JBO 将作为 A3 体系中基本的业务对象模型在多方面得以使用。

JBO 模型包括了业务对象定义、表示、管理一套机制，提供辅助工具进行对象的辅助操作。JBO 体系主要特征如下：

1. 动态对象定义和生成。JBO 对象通过动态定义产生，通过改动定义改变对象的特性，运行环境在运行时动态产生和确定对象；

2. 个性化对象管理理念。JBO 理念中强调业务对象的个性，理论上每个对象都有自己独立的管理器实现对象持久化管理。这一点是在概念区别于 JPA 强调的统一的 O-R Map 技术；

3. 高质量基础管理器实现。尽管从概念上 JBO 强调业务对象个性，强调逐一管理的理念，在实现上平台提供基础的一些管理器实现，绝大多数的对象管理可以通过基础的管理器实现；

4. 丰富的元数据管理。在对象级完整提供业务元数据访问，应用程序在对象实体上直接获取原始的业务信息，比一般的元数据和类级别访问更为简单方便；

5. 对象自解释设计。JBO 体系中对象具有自解释性，除了对象本身的业务数据，还包含了丰富的业务元数据信息、自身状态信息，完整的独立性保证对象和管理器、运行环境的松耦合，管理器对对象的管理都是无状态服务，对象也可以在系统间迁移而不丢失信息；

6. 易于接受的技术实现。JBO 的技术实现上借鉴 JPA、EJB、DB 以及 Java 自身的一些技术概念、名词、语法，尽量让 Java 程序员可以较为容易的熟悉和使用 JBO；
7. EL 扩展支持。JBO 提供和 JSF-EL 的扩展，在表达式中可以直接访问 JBO 对象；
8. 辅助工具。JBO 技术实现提供常用的工具来方便 JBO 的使用，包括序列化、反序列化、JSON 对象转换工具，POJO 对象转换工具，XML 序列化工具等。

借助于 JBO 对象的自解释性和辅助工具，JBO 技术除了作为业务系统核心的业务对象使用之外，在系统间交互和集成时能发挥特别作用。

2.3 JBO 组成

JBO 体系中，主要分为业务对象定义、业务对象管理、业务对象查询、辅助工具四个部分，整体上采用定义和使用分离的模式，应用程序只访问业务对象和对象管理的定义部分，实现部分对应用透明。JBO 体系中强调对象的管理，概念上查询是属于对象管理的组成部分，不能独立于管理器存在。辅助工具相对独立，是为了应用更为方便的使用 JBO，不是核心的部分。JBO 程序包括五部分：

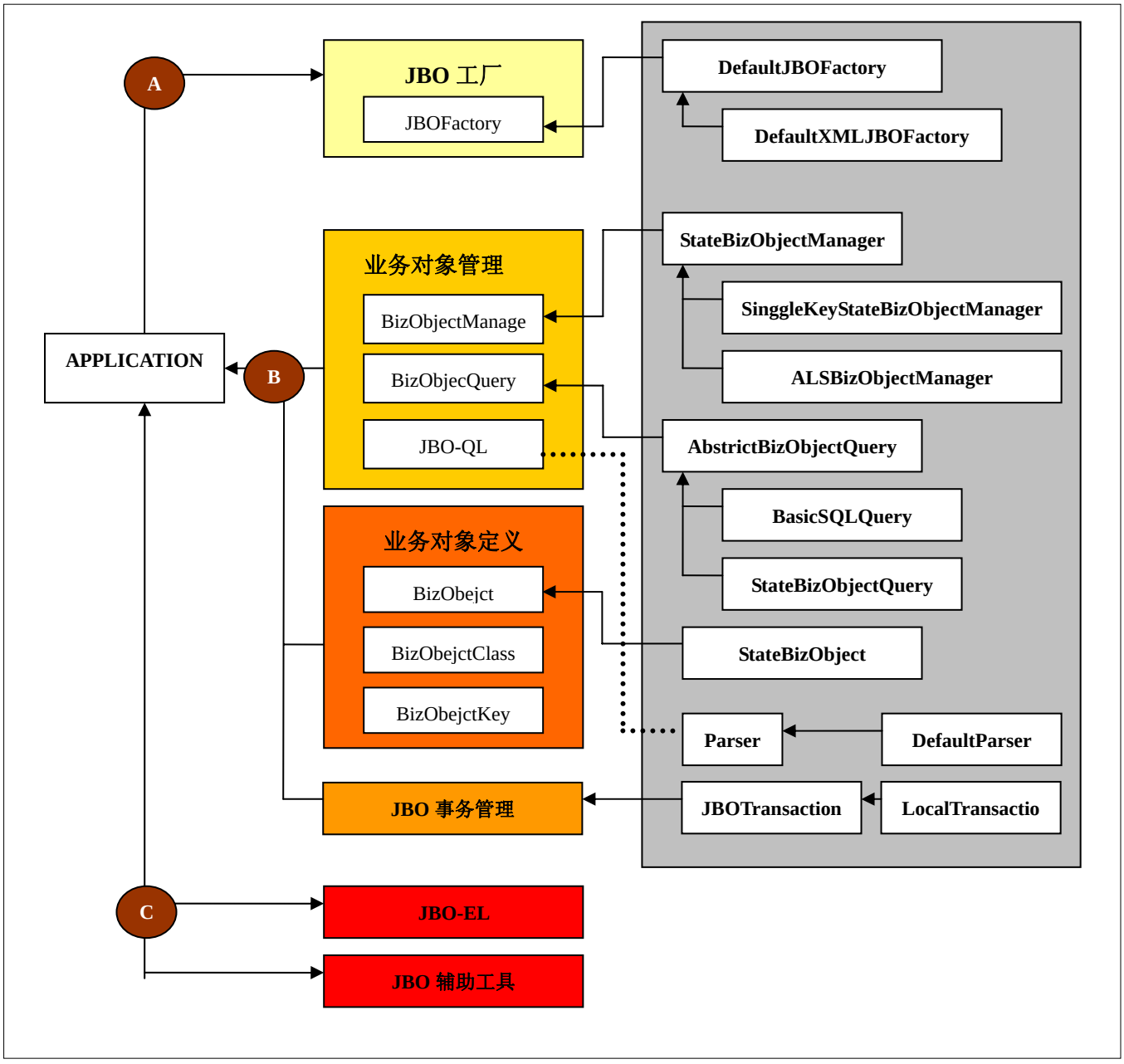
1. JBO 工厂，配合 AREService 建立的 JBO 使用总的入口，负责在 ARE 初始化时建立 JBO 运行环境，应用程序通过 JBOFactory 获取 JBO 应用的其他部分；
2. 业务对象定义。定义业务对象技术规范和技术接口，应用程序只使用明确定义的标准接口，以达到最大的可移植性和扩展性。JBO 业务对象定义包括：
   - 对象。BizObject 代表一个业务对象实体；
   - 关键字。BizObjectKey 唯一标识业务对象实体，可以有多个属性组成；
   - 对象类；BizObjectClass 是对象所属类，是它的元信息定义。
3. 对象管理器。对象管理器定义对象管理的标准方法，是应用程序存取对象的入口。对象管理器部分包括一般定义和一些基础实现，在基础实现中可以根据管理器的特性定义合适的 JBO 对象；

4. 对象查询。对象查询时对象管理器的扩展和衍生，对象管理器对单个对象进行管理，对象查询用比较接近于 SQL 的查询语法（JBO-QL）对对象进行批量操作；

5. 辅助工具。辅助工具部分实现 JBO 和其他对象模型的相互转换，是辅助应用使用 JBO 对象的工具模块；

6. 表达式语言扩展。JBO-EL 扩展 JSF-EL，扩展原来的 EL 中支持 JBO 对象访问。

2.4 JBO 基本程序对象



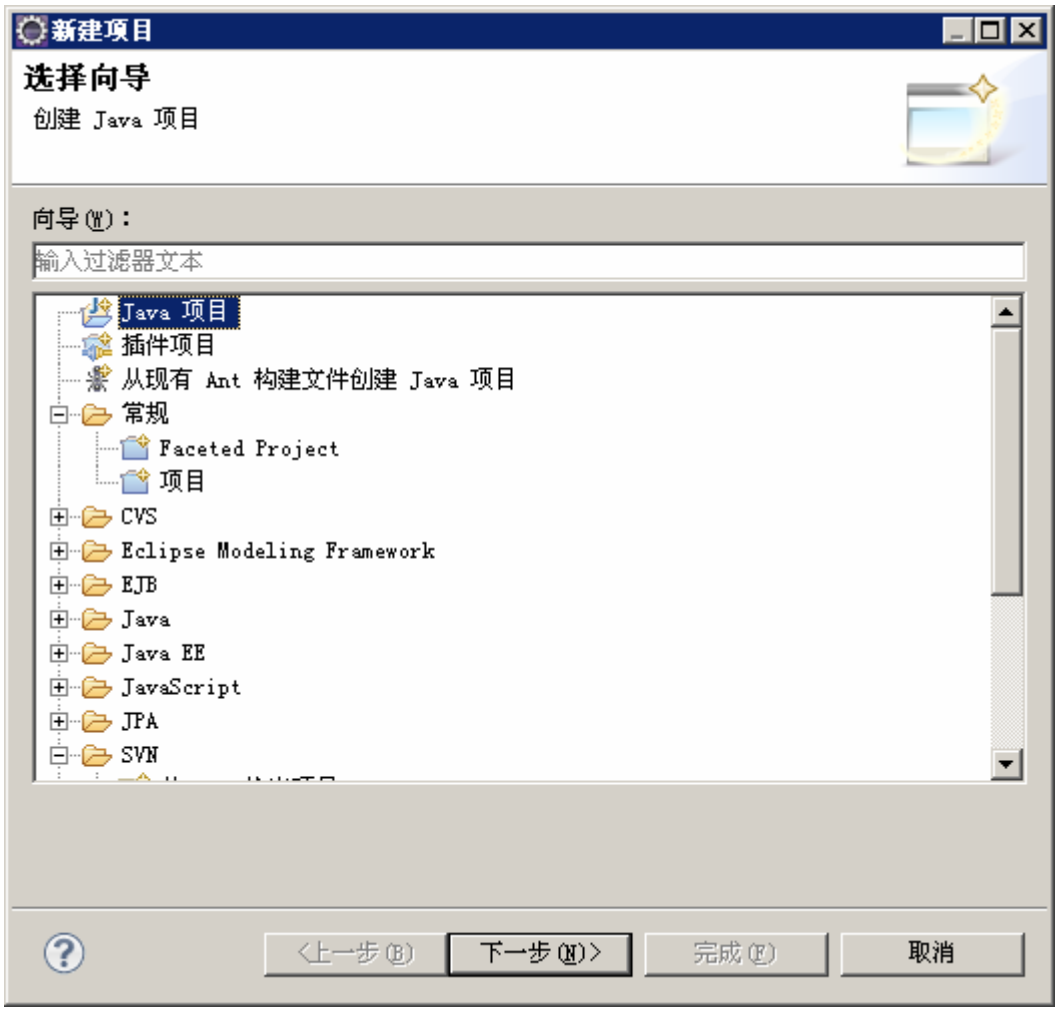
在应用程序中，主要使用 BusinessObject，BizObjectManager, BizObjectQuery, JBOTransaction 等上层的接口和对象进行开发。

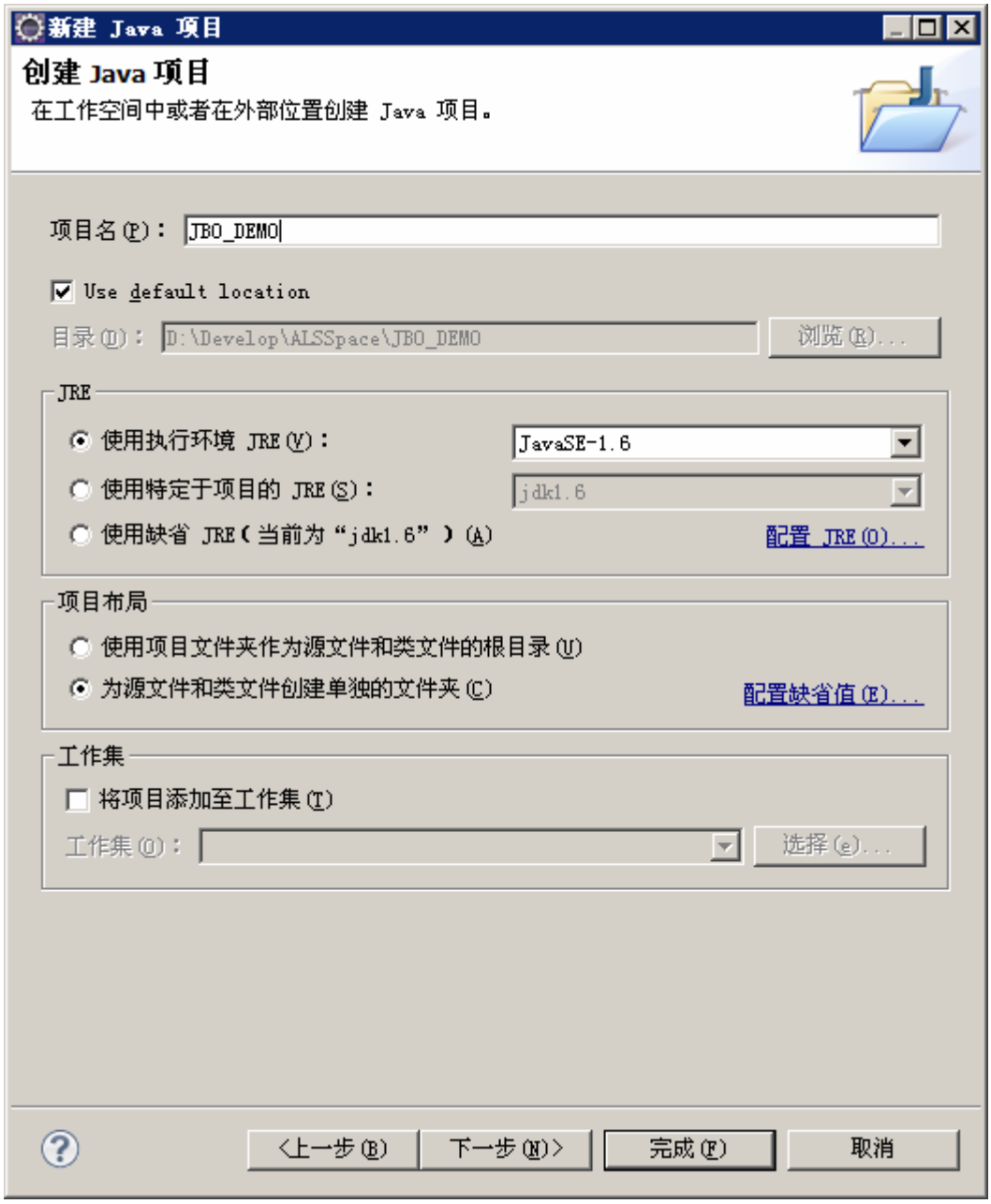
3 应用开发

3.1 创建 JAVA 项目

为了便于进行程序的调试，需要在 Eclipse 中新建 Java 项目进行程序的调试。

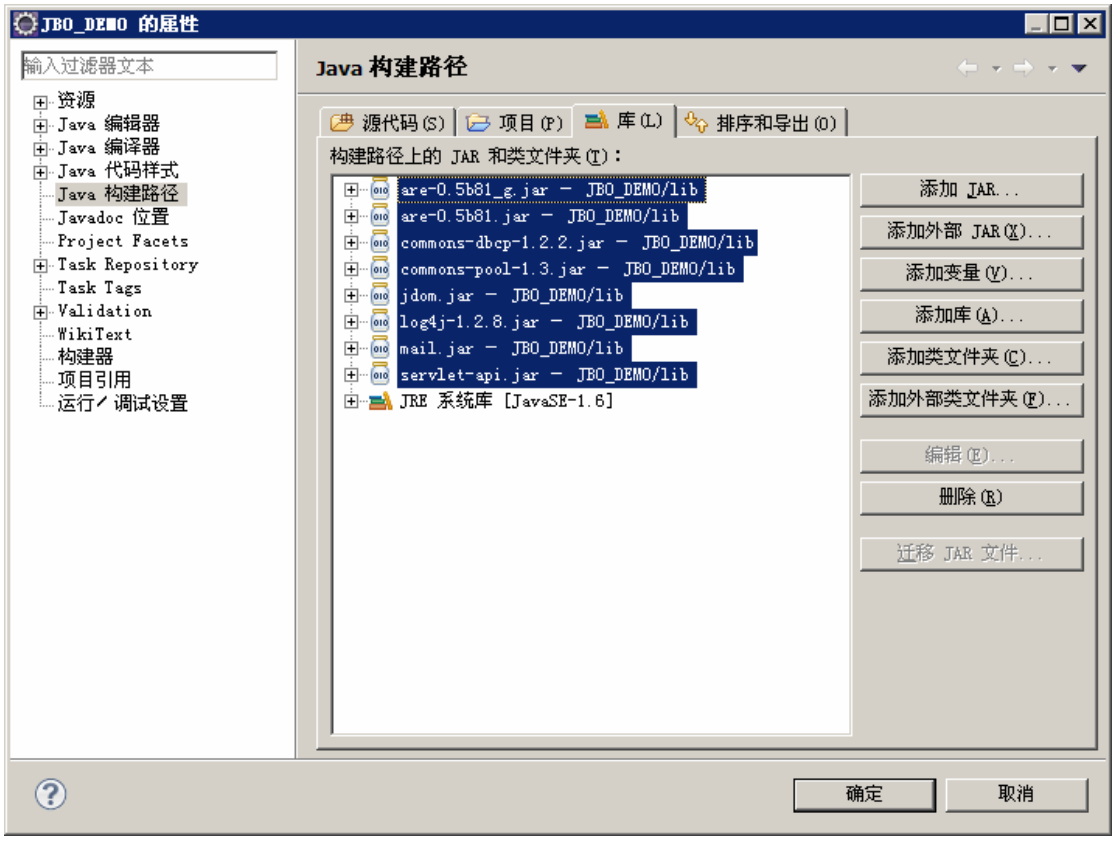
1、 在 Eclipse 中，新建 JAVA 项目：



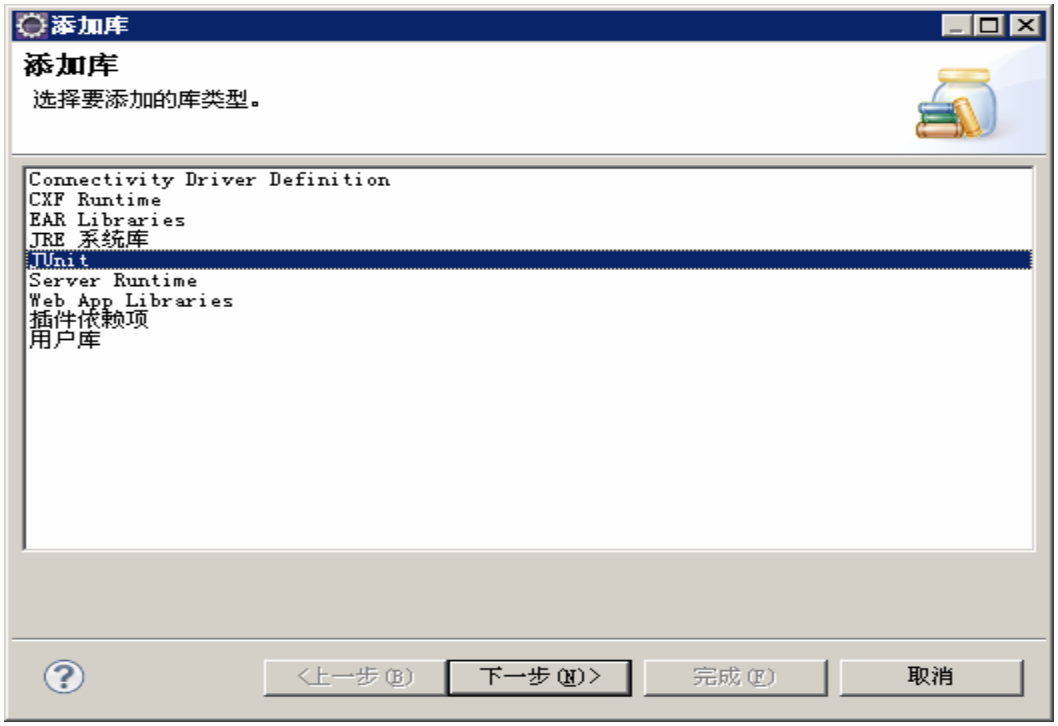


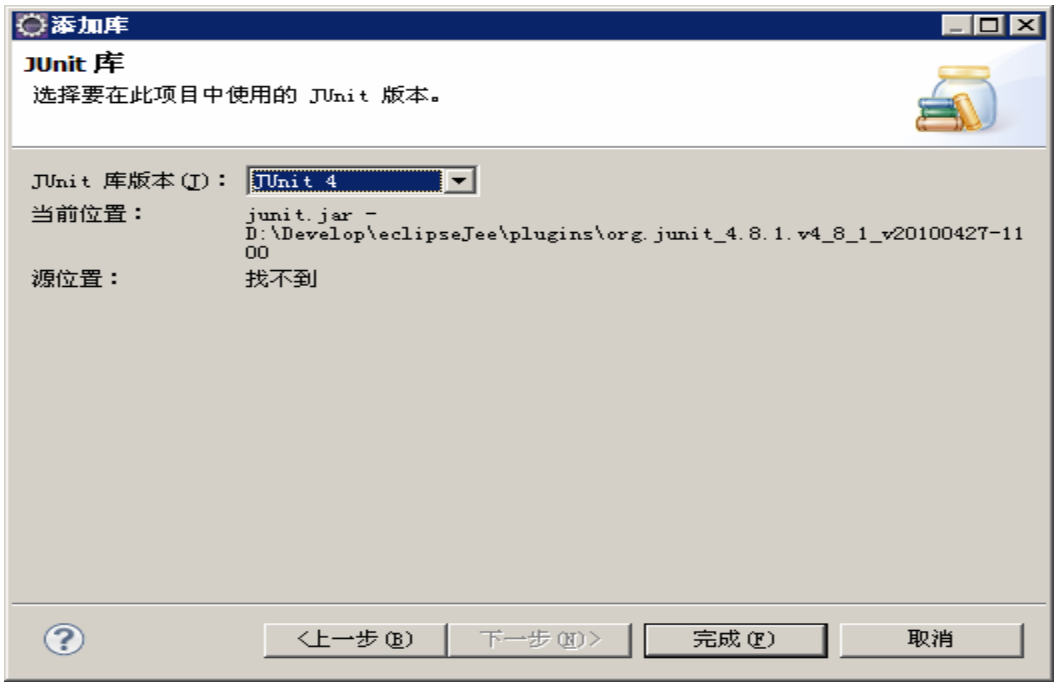
2、 将 JBO_TRAIN 中的程序全部拷贝到项目 JBO_TRAIN 中。

3、 设置 JBO_DEMO 项目的库，引用项目 lib 目录下的所有 jar 包



4、 设置 JBO_DEMO 项目的库，添加 junit 相关包：





5、 开始以下程序的调试工作。

3.2 第一个 JBO 程序

JBO 的模型和实现很复杂，但在应用中使用 JBO 非常简单。本节以一个简单的程序直观介绍对一个 JBO 对象的新增、更新、查询、删除。

```
1  package demo.jbo;
2  import com.amarsoft.are.ARE;
3  import com.amarsoft.are.jbo.*;
4
5  public class HelloWorld {
6    public static void main(String[] args) {
7      ARE.init("etc/are.xml");
8      JBOFactory f = JBOFactory.getFactory();
9      try {
10       BizObjectManager m = f.getManager("jbo.sample.customer.Employee");
11
12       BizObject me = m.newObject();
13       me.setAttributeValue("EmployeeID", "1000");
14       me.setAttributeValue("EmployeeName", "SUPPER MAN");
15       m.saveObject(me);
16
17       BizObjectKey k = m.getBizObjectKey().setAttributeValue("EmployeeID","1000");
18       me = m.getBizObject(k);
19       System.out.println("Hello World, I'm " + me.getAttribute("EmployeeName"));
20
21       me.setAttributeValue("EmployeeName", "WHO AM I");
22       m.saveObject(me);
23       BizObjectQuery q = m.createQuery("SELECT EmployeeID,EmployeeName FROM O WHERE EmployeeID=:employeeID");
24       q.setParameter("employeeID","1000");
25       me = q.getSingleResult(); //me = (BizObject)q.getResultList().get(0);
26       System.out.println("Hi, I have renamed to " + me.getAttribute("EmployeeName"));
27
28       m.deleteObject(k);
29       me = m.getBizObject(k);
30       System.out.println("Oh, who killed me?, now I'm " + me);
31     } catch (JBOException e) {    e.printStackTrace(); }
32   }
33 }
```

上面程序完成了一个对象的创建、查询、更新和删除的过程。运行结果如下：

```
Hello World, I'm SUPPER MAN
Hi, I have renamed to WHO AM I
Oh, who killed me?, now I'm null
```

Hello World 有效的程序只有十多行，包含一个典型的业务对象生命周期管理和应用对 JBO 使用的典型入口：

1. 运行环境初始化。JBO 是 ARE 一部分，首先要做 ARE 初始化（Line 7）；
2. 获取 JBO 管理总入口。JBO 所有管理动作永远是从 JBOFactory 开始，因此总要获取一个 JBO 工厂（Line 8）；
3. 从对象管理器开始。绝大部分的对象管理动作都是通过 BizObjectManager 进行，要用 JBOFactory 获取一个特定对象的管理器（Line 10）；
4. 创建新业务对象。创建对象的唯一方法是 BizObjectManager.createObject()（Line 12）；

5. 通过关键字获取对象。首先从 BizObjectManager 中得到一个 Key，设置 Key 的值后可以通过 getBizObject()方法取得一个对象（Line 17,18,29）；

6. 通过查询获取对象。通过查询可以获取一个或多个对象，首先通过 BizObjectManager.createQuery()创建一个查询，然后设置参数，之后即可查询对象。这里我们看到了查询的参数是类似于 SQL 的语句，在这个查询中两个最重要的区别是在 SQL 出现表名的位置代之以“O”，它代表了 Manager 管理的查询对象，在 SQL 的参数部分用:param 的形式代替了?，这是 JBO-QL 的命名参数（Line 23-25）；

7. 对象更新保存。不管是新增的还是更新过后的对象统一通过 BizObjectManager.saveObject()方法完成最后的存储过程；

8. 对象的删除。删除业务对象的方法是 BizObjectManager.deleteObject()，删除操作即时完成，不需要保存操作；

9. 对象属性存取。获取对象的属性和更新对象属性是应用的主要方面，其语法比 Java Bean 的属性复杂一点，要通过获取属性本身来操作，但也提供了简单的方法，总体语法和 Collection 语法接近（Line 13,14,21）。

3.3 起点：总是从 JBOFactory 开始

JBOFactory 是整个 JBO 应用的总入口，所有的操作都是以它为起点的。顾名思义 JBOFactory 实现的是工厂模式中的工厂，用来产生各种 JBO 使用对象。

3.4 起点的起点：获取 JBOFactory

JBOFactory 是一个抽象类，不能用于实例化，具体 JBOFactory 对象实例的获取必须通过类方法 getFactory()得到，获取的对象是 JBOFactory 的一个具体实现对实现类的实体，这个有系统决定，应用无须关心。获取工厂语法如下：

```
JBOFactory factory = JBOFactory.getFactory();
```

3.5 基本方法

JBOFactoy 实例可进行的基本动作包括：

- 缺省包：String getDefaultPackage();
- 包列表：String[] getPackages();
- 类(名字)列表：String[] getClasses();
- 特定类：BizObjectClass getClass(String className);
- 特定管理器：BizObjectManager getClass(String className);
- 事务体：JBOTransaction createTransaction();

前三个方法一般是给管理工具使用，用于整体了解工厂的情况。后三个方法则是给应用使用，用于在应用使用 JBO 对象，尤其是最后两个方法是应用中最主要的用法，因为：

- 对象的管理都是通过 BizObjectManager 进行的，BizObjectManager 相当于是每一类对象的总管，管理这个家族每个成员实体；
- 当多个对象（操作）综合应用时，必须用事务来保证一致性。

3.6 直接用法

直接用法为了简化应用书写而设置，本质上并没有增加 JBOFactory 的功能。因为 JBO 业务对象的大部分操作都是通过 BizObjectManager 进行，使用起来总是要重复获取 JBOFactory 实例和 BizObjectManager 的代码，使得应用不太简练。因此在 JBOFactoy 增加了一些静态的方法方便应用简单的使用 JBO 的功能。

- 直接可用的方法是静态方法，因此可直接使用而无须实例化 JBOFactory 以及 BizObjectManager 对象，包括：
- 直接获取管理器：BizObjectManager

JBOFactory.getBizObjectManager(String className);

- 直接创建查询：BizObjectQuery JBOFactory.createBizObjectQuery(String className,String query);
- 直接创建事务：JBOTrasaction JBOFactoy.createJBOTransaction();

静态方法的特点是方法名字中都包含的类全名，而上节中的实例的基本方法名字中则都是对象的核心单词。

静态方法可省去创建中间的 Factory 和 Manager 的书写，但当中间对象对次使用时，还是创建出来比较合适。

3.7 JBO 对象管理

JBO 对象管理指的是 JBO 对象的持久化生命周期管理，包括：

- 对象创建(Create)；
- 对象获取(Retrieve)；
- 对象更新(Update)；
- 对象删除(Delete)；

也就是常说的 CRUD，有了这些基础管理应用程序可以专注于业务实现，无须关心繁琐持久化操作。JBO 对象的管理总的来说都是通过 BizObjectManager 来进行的，具体的又可以分为单个对象的管理和批量管理：

- 单一对象管理通过 BizObjectManager 实现；
- 批量对象管理通过 BizObjectQuery 进行，BizObjectQuery 由管理器创建,用来执行查询语言 JBO-QL；

两种管理方法主要方法对比如下：

序号	管理内容	单一管理	批量管理

1	管理对象产生语法	BizObjectManager m = JBOFactory.getManager(className)	BizObjectQuery q= BizObjectManager.createQuery(queryStr) BizObjectQuery q = JBOFactory.createQuery(className,queryStr);
2	获取对象	BizObjectManger.getBizObject()	SELECT 查询
2	创建新对象	BizObjectManger.newObject()	无
3	更新对象	BizObjectManager.saveObject()	UPDATE 查询
4	删除对象	BizObjectManager.deleteObject()	DELETE 查询

3.7.1 单一对象管理

单一对象管理，通过 BizObjectManger 的四个方法完成，在 Java 程序中直接调用，属于原生 API 模式的编程技术，如上表。

3.7.2 批量对象管理

批量对象管理是通过 JBO-QL 来实现，其本质是执行另一种语法的代码，属于动态脚本类型的编程技术。JBO-QL 支持三种类型的查询语句，查询的语法接近于 SQL 写法，具体语法参考 3.5 和第 5 章，查询只有**获取、更新、删除**三种语法，**没有新增的语法。**

查询对象支持的主要特征为：

- 用 O 代表管理器管理的对象，这是不同于数据库查询的地方；
- 参数化查询，支持命名参数查询。如：query.setParameter(“myparameter”,”myvalue”)，参数类型可以是基本数据类型和 String、Date、DataElment 等 JBO 常用类型，这是区别于 SQL 查询的位置参数的；
- 获取结果语法类似于 JDBC，但支持单个查询结果和结果列表。如：BizObject o = query.getSingleResult()，List l = query.getResultList();
- 可以设置查询的开始位置和最大条数，setFirstResult(),setMaxResult()。应用可以利用这个两个方法优化性能，或者在界面上做分页；

- 支持在查询结果中直接获取汇总数据，getTotalCount(),getSumOf(String attr)，获取总条数和某个数字字段的汇总数据。此处的汇总都是针对整个查询的，和 firstResult,maxResult 无关；
- 和 JDBC 一样的更新/删除执行方式。如：query.executeUpdate();

3.8 JBO 对象使用

3.8.1 对象访问

对象的信息包括自身的类型信息和业务数据信息，信息的获取主要通过 BizObject 的几个方法和对应的 BizObjectClass 的方法完成。BizObjectClass 对象可通过三种方法获取：

- JBOFactory.getClass(String className);
- BizObjectManager.getManagedClass();
- BizObject.getBizObjectClass();

BizObject 总体访问方法：

序号	方法	说明
1	BizObjectKey getKey()	获取对象的关键字
2	DataElement[] getBriefAttributes()	获取对象的简要描述字段
3	boolean instanceOf(String bizobjectClass)	判断当前对象是否是参数中类的实例
4	boolean instanceOf(BizObjectClass bizobjectClass)	判断当前对象是否是参数中类的实例
5	int indexOf(String attributeName)	确定属性在属性列表内的序号，如果<0 表示不是属性
6	int getAttributesNumber()	获取属性个数
7	DataElement[] getAttributes()	获取属性列表
8	DataElement getAttribute(String attributeName)	根据名字获取属性
9	DataElement getAttribute(int attributeIndex)	根据顺序获取属性

BizObjectClass 用于描述 BizObject，拥有 BizObject 的很多原始信息，在没有对象实体的情况下，应用可以通过 BizObjectClass 构造需要的程序：

序号	方法	说明
1	String getPackageName()	获取对象的关键字

2	String getName()	
3	String getAbsoluteName ()	判断当前对象是否是参数中类的实例
4	String getLabel ()	判断当前对象是否是参数中类的实例
5	int indexOf(String attributeName)	确定属性在属性列表内的序号，如果<0 表示不是属性
6	BizObjectClass getParnet()	获取父类
7	DataElement[] getAttributes()	获取属性列表
8	DataElement getAttribute(String attributeName)	获取某个属性
9	DataElement[] getKeyAttributes()	获取本类的 key 属性数组
10	DataElement[] getBriefAttributes()	获取本类的简要描述字段
11	BizObjectClass extent(String clazzName,DataElemet[]attributes)	创建一个命名扩展类，包含扩展的属性
12	BizObjectClass extent(DataElemet[] attributes)	创建一个自动命名扩展类，包含扩展的属性
13	AttributeUIHint[] getAttributeUIHint(String attributeName)	获取某个属性的 UIHint

3.9 属性存取

属性存取是应用中对对象数据操作的最主要的方面，JBO 对象的属性是泛基本数据元素 DataElement。DataElement 可以存放 String、int、long、double、Date、boolean 六种基本数据类型，DataElement 本身具有 Type，是上述基本类型之一，但可以任何一种类型进行复制和读取，程序实现为尽力转换模式。

属性存取有两种格式：

- 先获取属性对象，再按照进行 DataElement 的各种方法进行操作。获取对象属性的方法有三个：
  - getAttribute(String name)，根据属性名称获取；
  - getAttribute(int index)，根据序号获取；
  - getAttributes()，获取所有属性为一个数组，再进一步操作。
- 直接用 BizObject 的方法进行快速操作；

这两种方式有部分是相同的，但也有部分是特有的。

No	DataElement 方法	BizObject 方法	说明
1		setValue(BizObject value)	把对象的值设置给主对象，参数对象必须和主对象类型相容。即是对象同类或者是子类
2		setAttributesValue(BizOnject obj)	把对象相同名字的属性批量设置，参数类型可以和主对象不一致，只进行属性名匹配
3		setAttributesValue(Map valueMap)	根据 Map 批量设置属性，按照 Key 和 AttributeName 对应
4	getAttribute(attributeName). setValue(int value)	setAttributeValue(attrName,int value)	设置命名属性的 int 值
5	getAttribute(attributeName). setValue(long value)	setAttributeValue(attrName,long value)	设置命名属性的 long 值
6	getAttribute(attributeName). setValue(doublevalue)	setAttributeValue(attrName,double value)	设置命名属性的 double 值
7	getAttribute(attributeName). setValue(Boolean value)	setAttributeValue(attrName,boolean value)	设置命名属性的 boolean 值
8	getAttribute(attributeName). setValue(Object value)	setAttributeValue(attrName,Object value)	设置第 index 属性的 Object 值，对 Date，String，DataElement 可 直接设置，对 Integer，Long，Double 值进行其基础类型值设置，对其他类型用 value.toString() 的值设置
9	getAttribute(index). setValue(int value)	setAttributeValue(index,int value)	设置第 index 属性的 int 值
10	getAttribute(index). setValue(long value)	setAttributeValue(index,long value)	设置第 index 属性的 long 值
11	getAttribute(index). setValue(doublevalue)	setAttributeValue(index,double value)	设置第 index 属性的 double 值
12	getAttribute(index). setValue(Boolean value)	setAttributeValue(index,boolean value)	设置第 index 属性的 boolean 值
13	getAttribute(index). setValue(Object value)	setAttributeValue(index,Object value)	设置第 index 属性的 Object 值，对 Date，String，DataElement 可 直接设置，对 Integer，Long，Double 值进行其基础类型值设置，对其他类型用 value.toString() 的值设置
14	setNull()		设置属性为空
15	setValue(DataElement value)		用具体的某种对象设置属性值
16	setValue(Date value)		
17	setValue(String value)		
18	getInt(); getLong();		获取属性的制定类型的值

	getDouble();
	getBoolean();
	getString();
	getDate();
	getValue();		

19	getName();
	getLabel();
	getType();
	getLength();
	getScale();
	getProperty(String propName)
			获取属性的元数据信息

20	compareTo(Object o); equals(Object o)		比较属性和其他对象

3.10事务支持

JBO 模型中包含了事务处理机制，事务机制采用的是显式事务对象模式，应用通过明确的创建事务控制对象，然后加入事务操作对象，在所有操作后显式的用事务控制对象的提交和回滚方法完成事务。

构成 JBO 事务处理机制的有两个关键部件构成：

- 事务控制对象(Tansaction Controller，TC)。此接口用于把有事务性质的操作（主要是 JBO 的存储，也可包括其他数据操作）统一到一个事务中进行控制。
- 支持事务的对象（Transacted Object）。一般是用来操作 JBO 对象的对象，如管理器，查询等。

一个 TC 对加入其中的事务对象（TO）的操作进行统一管理，保证所有 TO 操作一致性，加入事务体的语法是：JBOTransaction.join(Transacted o)。

事务控制过程要点：

- 事务周期从第一次调用 join 方法加入 TO 开始；
- 事务周期到第一个 commit/rollback 结束；

- 在周期内的 join 不再构成新的事务，只是加入当前的事务中；
- 当调用了 commit/rollback 之后，当前事务就结束，再次调用 join 会开启新的事务过程。

缺省实现事务对象的是 LocalTransaction，实现同一个数据库内的事务，未来可能支持分布式事务，为保证定义和实现无关，事务对象的获取统一通过 JBO 工厂获取，语法为 JBOFactory.createJBOTransaction()或 JBOFactory.getFactory().createTransaction()。可以进入事务的共享事务体包括 JBO 内置的标准查询和管理器。

下面是一个基本事务使用的代码例子：

```
15  public void testLocalTransactionBasic(){
16    try {
17      JBOTransaction tx =JBOFactory.getFactory().createTransaction();
18  //  JBOTransaction tx =JBOFactory.createJBOTransaction();
19      assertTrue(tx instanceof LocalTransaction);
20      assertNull(((LocalTransaction) tx).getDatabase());
21      EmployeeManager em = new EmployeeManager();
22      tx.join(em);
23      assertEquals(((LocalTransaction) tx).getDatabase(),em.getDatabase());
24      assertNotNull(em.getTransaction());
25      assertEquals(em.create("3001","TEST NAME1"),1);
26      assertEquals(em.getName("3001"),"TEST NAME1");
27      assertEquals(em.changeName("3001","TEST NAME2"),1);
28      assertEquals(em.getName("3001"),"TEST NAME2");
29      assertEquals(em.delete("3001"),1);
30      assertNull(em.getName("3001"));
31      tx.commit();
32      assertNull(em.getTransaction());
33    } catch (JBOException e) {
34      e.printStackTrace();
35      fail(e.getMessage());
36    } catch (SQLException e) {
37      e.printStackTrace();
38      fail(e.getMessage());
39    }
40  }
41
```

1. 首先创建事务对象（Line 17），可以有两种方法;
2. 加入 1 到多个要管理的共享事务对象（Line 22）；
3. 进行正常的 JBO 操作（Line 25-30）；

4. 提交或回滚事务（Line 31）.

实际上单个对象的操作不是事务的主要场合，更重要的是多个操作的情况，使用方法和上面的单个对象的使用完全一样，如下：

```
114  public void testJBOQuery() throws JBOException{
115     JBOFactory f = JBOFactory.getFactory();
116     JBOTransaction tx =f.createTransaction();
117     BizObjectManager em = f.getManager("jbo.sample.customer.Employee");
118     BizObjectManager cm = f.getManager("jbo.sample.customer.Customer");
119     BizObjectManager pm = f.getManager("jbo.sample.project.Project");
120
121     //多Query事务测试，混合事务和非事务模式
122     tx.join(em);
123     tx.join(cm);
124     em.saveObject(em.newObject().setAttributeValue("EmployeeId", "T001").setAttributeValue("EmployeeName","NameT000"));
125     cm.saveObject(cm.newObject().setAttributeValue("CustomerId", "T0001").setAttributeValue("CustomerName","NameT000"));
126     pm.saveObject(pm.newObject().setAttributeValue("ProjectID", "T00001").setAttributeValue("ProjectName","NameT000"));
127     tx.commit();
128     tx.join(em);
129     tx.join(cm);
130     em.createQuery("delete from o where EmployeeId='T001'").executeUpdate();
131     cm.createQuery("delete from o where CustomerID='T0001'").executeUpdate();
132     pm.createQuery("delete from o where ProjectID='T00001'").executeUpdate();
133     tx.rollback();
134     assertNotNull(em.getBizObject(em.getBizObjectKey().setAttributeValue(0, "T001")));
135     assertNotNull(cm.getBizObject(cm.getBizObjectKey().setAttributeValue(0, "T0001")));
136     assertNull(pm.getBizObject(pm.getBizObjectKey().setAttributeValue(0, "T00001")));
137  }
```

1. 程序首先创建了一个事务对象和三个管理器 em、cm、pm（Line 116-119）；
2. 第一步把 em、cm 加入到事务中，进行了操作（Line 122-126）；
3. em、cm 进行成功提交，此时事务对象 tx 完成了一个完整的事务过程。tx 回到初始状态可以重复使用（Line 127）；
4. 事务已经提交，必须重新开始，em,cm 重新加入 tx（Line 128,129）；
5. em,cm 创建了两个查询，并执行了更新操作，因为 Manager 加入到了事务中，它们创建的查询也自动加入到了事务中(Line 130,131)；
6. pm 创建了另一个查询，但没有加入到 tx，所以 executeUpdate()即使生效（Line 132）；
7. 事务回滚，cm 和 em 执行的查询回滚，但 pm 执行的查询完成了实际更新（133）。

参考代	以上代码选自：test_src/com/amarsoft/are/jbo/TestTransaction.java 更多案例可请参阅此程序。

重要提示

1. 事务体必须显式进行 commit/rollback，否则会锁死数据库或者其他资源；
2. 技术上，事务处理过程中，可以进行其他非事务的操作，但实际程序中应尽量把不必要的事情放在事务之外，缩短事务过程的时间，减少资源锁定时间；
3. JBO 缺省 Manager 实现 StateBizObjectManager 加入事务后，新创建的查询时会加入同一事务，但不会加入之前创建的查询，如果需要可以显式的把查询加入到事务中。

3.11高级查询

JBO 查询语法由 JBO-QL 定义，其语法格式类似于 SQL，详细语法定义参照第 5 章，本节说明和 SQL 明显差异的 JBO-QL 特有的查询语法用法。

3.11.1 属性表达式

在 JBO-QL 的 SELECT 查询列表中，可以用模式匹配写法来简化查询属性列表的书写。匹配模式有两种情况：

- “*“匹配全部对象属性；
- “O.正则表达式“匹配部分属性；

在查询语句的 SELECT 列表中，用双引号““包括的被看做是属性表达式，在执行查

询时会进行扩展。例如：

1. SELECT "O.*" FROM …，解析为 SELECT O 的全部属性列表；
2. SELECT "O..id",o.workyear,"O..*m.*" FROM O…，解析后的属性列表为三部分：
   - 以 id 结尾的所有属性；
   - 名字为 workyear 的属性；
   - 所有包含字母‘m’的属性。

重要提示

1. 属性表达式仅限于主对象，也就是 O 的属性才可以用表达式；
2. 属性表达式必须用双引号“”包括起来，否则认为是常规属性；
3. 程序中经常使用字符常量写查询语句的，此时要把属性表达式的引号做转义。如:“SELECT \” O.*\” FROM O”；
4. 属性列表最后会转换成真实的选择列表，最终选择的列表如果不是对象的全部属性，得到的对象的对应属性会是 null 值，如果做了修改是可以保存的。

3.11.2 扩展查询对象

JBO 对象类是先定义好的，因此查询出来的对象只能是这个既定类的实体，包含这个类的属性。而实际使用中，在很多查看信息的情况下需要使用本对象之外的信息，这些信息通，虽然通过对象关联关系可以得到，但如果每个对象都由应用去单独获取，会比较复杂。因此 JBO-QL 支持查询时对基础对象的扩展。扩展分为两种：

- 扩展属性。扩展属性从其他的对象中获取，其概念类似于 SQL 的 JOIN 关系，实际上其语法也是和 SQL 的 JOIN 语法一致。通过查询中加入扩展对象和扩展属性，查询出的对象就有了扩展的属性，扩展对象必须用“类全路径”+“ ”

+“别名”的形式引入，扩展属性必须用“扩展对象别名”+“.”+“扩展属性”形式；

- 虚拟属性。虚拟属性是另外一种扩展方法，他和上面的扩展属性不同的是虚拟属性不是直接来自于其他 JBO 对象，而是通过计算得到，计算是某种表达式或者函数，通常计算虚拟属性时会用到主属性和扩展属性。虚拟属性必须是：“计算表达式”+“空白”+“AS”+“V.虚拟属性名”，其写法类似于 SQL 的字段 AS 别名，但必须是加上 **V.**的前缀。虚拟属性除了选择列表外，不能出现于任何别的地方。

```
84      //Join
85      BizObjectManager mm = JBOFactory.getFactory().getManager("jbo.sample.project.Project");
86      s = "Select \"o.*\",m.employeeid,m.position from o,jbo.sample.project.ProjectMember m where o.projectid=m.projectid and o.projectid like '%0001'";
87      q = new BasicSQLQuery(mm,s);
88      qs = q.getQuerySql();
89      q.setDatabase(m.getDatabase());
90      System.out.println(qs);
91      assertTrue(q.getResultList().size()>0);
92      assertTrue(qs.startsWith("SELECT"));
93
94      s = "select \"o.*\",m.employeeid,m.position from o JOIN jbo.sample.project.ProjectMember m ON  o.projectid=m.projectid where o.projectid like '%0001'"
95      q = new BasicSQLQuery(mm,s);
96      qs = q.getQuerySql();
97      q.setDatabase(m.getDatabase());
98      System.out.println(qs);
99      assertTrue(q.getResultList().size()>0);
100     assertTrue(qs.startsWith("SELECT"));
101
```

上例中第一段显示了在选取 Project 主对象时，联合选取了 ProjectMember 的 employeeid 和 position 属性。这两个属性前都加入了前缀“m.”，m 是在 from 后面的“jbo.sample.project.ProjectMember m” 中定义的，这是隐性的扩展，关联条件在 WHERE 字句中体现；

第二段例子和第一段的含义一样，不同之处在于 m 是在 JOIN 后的“jbo.sample.project.ProjectMember m” 中定义的，这是显性的扩展，关联条件在 ON 子句中体现。

```
28      s = "Select Education,count(*) as v.count_all,sum(Salary) as v.sum_salary ,min(email) as v.min_email,max(salary) as v.max_salary, avg(salary) as v.avg_salar
29          "from o  where o.EmployeeID>:arg and o.EmployeeName is not null and (WorkYear>=:la and WorkYear<=:lb) GROUP BY Education ORDER BY Education";
30      q = m.createQuery(s);
31      q.setParameter("arg","0002");
32      q.setParameter("la",1990);
33      q.setParameter("lb",2010);
34      List l = q.getResultList();
35      assertTrue(l.size()>0);
36      BizObjectUtil.printObjects(l);
```

上面的例子显示了虚拟属性的用法，查询中加入了若干个标准统计函数做成虚拟属性。

扩展查询属性并非主对象的原生属性，得到的查询结果中这些属性的 label 是没有定义的。JBO-QL 提供了一种动态的方式来为扩展属性指定 label，其语法是在扩展属性的名称后面紧跟上用花括号“{}”包括的 label，如：e.attribute{中文标签}，这个语法适用于扩展属性和虚拟属性。但又所不同：

- 对扩展属性，如果定义了 lable，则使用这个 label。如果没有 label，查询尝试获取属性来源的那个 JBO 扩展类的定义；
- 而虚拟属性则没有可能通过原来的对象获取 label，只能通过这种方式获取。

参考代	以上代码选自：test_src/com/amarsoft/are/jbo/impl/TestBasicQuery.java, test_src/com/amarsoft/are/jbo/impl/TestFunctions.java

重要提示

1. 主属性和扩展属性都是实际的对象属性，在查询字段中必须如实反映，不允许做运算和转转义，如：函数加工和 AS 别名都是不允许的；
2. 属性表达式必须用双引号““包括起来，否则认为是常规属性；
3. 程序中经常使用字符常量要用到””，此时要把属性表达式的引号做转义。如：“SELECT \” O.*\” FROM O”；
4. 属性列表最后会转换成真实的选择列表，最终选择的列表如果不是对象的全部属性，得到的对象的对应属性会是 null 值，如果做了修改是可以保存的。

3.11.3 合并查询结果

在 JBO 查询中，查询是针对特定对象的，查询的结果反映的是该类对象实际的情况，一般不需要扩展结果。但在一些场景下，应用确实希望在查询主对象的同时能够按照同样的属性维度合并其他对象的查询，这在一些界面展示时尤其常用。因此 JBO-QL 支持类似于 SQL UNION 的语法进行支持。

但无论如何，合并查询实际上是有多个独立查询通过 UNION 关键字关联起来的，和先后查询多个对象，手工加以合并没有本质区别。

使用 UINON 查询的结果并非真正的来自于主对象，其更新、保存等行为可能产生意外的结果，应慎重使用。

```
122         //uinon
123         s = "Select ProjectID,ProjectName from O  where o.projectid  like '%0001' union " +
124             "Select E.EmployeeId,E.EmployeeName from jbo.sample.customer.Employee E where E.EmployeeId<'0005'";
125         q = new BasicSQLQuery(mm,s);
126         qs = q.getQuerySql();
127         q.setDatabase(m.getDatabase());
128         System.out.println(qs);
129         assertTrue(q.getResultList().size()>0);
130         assertTrue(qs.startsWith("SELECT"));
131         System.out.print(q.getTotalCount());
132         System.out.print(q.getSumOf("workyear"));
```

如上代码，在 Line 122,123 定义了一个 UNION 查询，其语法特点如下：

- 第一个子查询为主查询，决定了查处的对象的类型；
- UNION 之后的查询为子查询，需如实写出每个属性，并且要加对象前缀。子查询要原样写出，不需要也不能用 AS 转义。

参考代	以上代码选自：test_src/com/amarsoft/are/jbo/impl/TestBasicQuery.java,

重要提示

1. 主查询的属性列表确定查询的结果；
2. 子查询必须保证查询属性和主查询个数类型匹配；
3. 主查询和子查询中都不允许进行属性运算和转移。

3.11.4 汇总信息

在 3.6.2 中看到在查询中通过扩展虚拟字段可以进行类似于 SQL 的 GROUP BY 的汇总查询。实际上 JBO 查询除了这种方法外，还有一种更简单的方法，就是当不需要分组时，可直接获取查询的总记录数和数字属性的汇总数字。如果既需要明细又需要汇总时，这个方法更方便。

```
37        s = "Select EmployeeID,EmployeeName,Salary  from o  where o.EmployeeID>:arg and " +
38              "o.EmployeeName is not null and (WorkYear>=:la and WorkYear<=:lb) ORDER BY O.EmployeeID";
39        q = new BasicSQLQuery(m,s);
40        q.setDatabase(m.getDatabase());
41        q.setParameter("arg","0002");
42        q.setParameter("la",2005);
43        q.setParameter("lb",2010);
44        count = q.getTotalCount();
45        sum = q.getSumOf("Salary");
46        System.out.println(count+","+sum);
47        assertTrue(count>0);
48        assertTrue(sum>0);
49
```

在创建了一个查询后，并没有执行查询获取明细信息。直接用 getTotalCount()和 getSumOf()来获取汇总数据（Line 44,45）。

需要注意的是，汇总是根据这个查询的条件产生的，并不因获取明细查询数据的多少而改变。

参考代	以上代码选自：test_src/com/amarsoft/are/jbo/impl/TestBasicCountSum.java,

3.11.5 查询数量控制

查询条件有可能返回很大的结果集，如果直接取出来可能会产生严重的资源问题甚至是内存溢出。BizObjectQuery 提供了两个方法来限制返回的结果大小和范围，应用可利用此选项来控制返回的对象实体多少。

BizObjectQuery.setFirstResult(int firstResult)，设置在查询结果集合中首个返回对象的位置，在此之前的结果被忽略不返回给应用，位置从 0 开始计数；

BizObjectQuery.setMaxResults(int maxResults)，设置查询结果返回的最大结果数，查询返回不超过 maxResults 个结果。

```
32      query = "employeeId>:id1 and employeeId<:id2";
33      q = m.createQuery(query);
34      q.setParameter("id1", "0");
35      q.setParameter("id2", "0006");
36      q.setFirstResult(2);
37      q.setMaxResults(3);
38      r = q.getResultList();
39      assertEquals(3,r.size());
40
```

3.12跟踪运行状况

从技术上讲，JBO 的管理器和查询都是可任意扩展的实现，并不容易监控。但真实的引用中绝大部分管理器和查询都是继承 JBO 标准实现中基于数据库的处理，其运行状况主要可以通过日志监控和分析。

标准实现在日志中记录从 JBO 查询到最后实现 SQL 以及 SQL 运行效率的每个转

换环节，对调试程序有重要用途。日志会记载过程中几个过程信息：

- 从原始查询语句转换为 SQL 查询语句；
- SQL 执行过程，包括：
- 原始语句；
- 转换参数后语句；
- 查询参数；
- 执行时间（起始时间、结束时间、耗时）。

缺省情况下，SQL 执行时间小于 0.1 秒（100ms）会以 SQL-TRACE 为标签在 DEBUG 级别记录，超过 0.1 秒的 SQL 会以 SQL-WARN-X 为标签在 WARN 级别记录，其中 X= 实际毫秒数/100。

4 JBO-QL 语法

4.1 JBO-QL 概述

JBO-QL 是 JBO 查询语言缩写，是可被 BizObjectQuery 执行的类 SQL 的语法。JBO-QL 和 SQL 的相同之处为：

- JBO-QL 使用和 SQL 相同的关键字；
- JBO-QL 使用和 SQL 相同的操作符号；
- JBO-QL 使用和 SQL 相同语法主体结构。

虽然从形式上 JBO-QL 尽量保持和 SQL 的一致，以减少学习困难，但作为以业务对象为核心的 JBO 查询语言，和面向数据库的 SQL 还是有不少差异。差异的根源有二：

- SQL 查询执行体（JDBC 的 Statement）本身无业务含义，是通用对象。而是在

SQL 语句内和结果中反映业务含义，相反 JBO-QL 的执行体是有特定的 BizObjectManager 创建的，有明确的主导业务对象和管理器，因此在 JBO 查询中存在主对象、主属性、主类的概念；

- JBO-QL 面向对象的查询，其中关于数据的一些特性是没有意义的，体现在查询目标是 BizObjectClass 而不是表、视图，SELECT 或者 UPDATE 的是属性列表不是字段列表，同样基于主对象的真实、无二意性，不允许对列表中的真实属性做转义和加工。

基于上面的两个根本差异，以及具体实现时的一些限制，JBO-QL 和 SQL 语法的明显区别如下：

1. 查询目标。JBO 查询目标有三种：主对象、扩展对象和虚拟对象，主对象用保留字“O”表示，扩展对象用对象类的全名称和别名定义，虚拟对象用“V”表示，只允许用作查询列表的虚拟字段前缀；
2. 属性明确。查询的属性列表中可包含主属性，扩展属性，虚拟属性三类。但三类属性的名字必须是唯一的。主属性和扩展属性不允许进行运算和修改，运算和表达式必须定义为虚拟字段使用；
3. 扩展对象定义，如果使用了扩展对象（JOIN、UNION、IN 子查询中）必须在 FROM 或者 JOIN 对象列表中使用完整对象类路径名，同时定义别名，而除此之外的地方必须用别名代表对象；
4. 非标准的 SQL 语法元素一律不被 JBO-SQL 所采用，如：+=、||等；
5. 双引号的使用。SQL 中的””限制数据库保留字为可用对象的做法不能使用，在 JBO-QL 中双引号用于 SELECT 属性列表中，表示是一个属性表达式；
6. 支持标准 FUNCTION 操作，通过注册机制可以加入新的 FUNCTION；
7. 参数的使用。JDBC 规定 SQL 中参数用占位的方法处理，用问号(?)表示参数，用位置数字设置参数。JBO-QL 中用命名参数的方式，用冒号（:）表示后面是一个参数，通过名字来设置参数；

8. 选择字段增强。SQL 中 SELECT 列表只能是单个枚举或者用“*”表示全部，JBO-QL 支持用正则表达式表示字段列表集合，支持用{label}形式为扩展和虚拟字段设置 label。

JBO-QL 被查询器执行，结果直接生效不需要经过保存或者其它操作，如果查询加入到事务之中，则按照事务的 commit 和 rollback 进行处理。

4.2 语法结构

JBO-QL 支持四种语法结构，QL 语法的第一个单词必须是“SELECT、DELETE、UPDATE”之一，由于 UINON 查询的特殊性，也把他作为一种语法结构，但本质归类还是 SELECT 查询。

4.2.1 SELECT

SELECT 查询语法用于获取一批对象结果，如果 SELECT 除了主对象外，加入其他的对象的属性或者虚拟对象属性，得到的对象是主对象的一个扩展对象。扩展对象继承主对象的所有属性，加入新增的属性。新增的属性只能用于查看，新增属性可以在查询语法中进行 label 的设置。

4.2.1.1 基本查询

SELECT {主属性列表} FROM O [WHERE 查询条件] [ORDER BY 排序属性列表 [ASC|DESC]]

- 主属性列表表示要查询的结果，支持直接属性和属性表达式混合的属性列表；
- WHERE 查询条件见 5.2.5 的专题说明；
- ORDER BY 列表是排序属性列表，只允许原始属性，排序可选使用 ASC|DESC 确定排序方案，缺省为正序 ASC。

兼容的简化查询语法：

- 空查询。查询语句 query 为空或无实际意义的空白字符串(即 StringX.isSpace(query)==true)时，自动创建一个等价查询：”SELECT \“O.*\” FROM O”；
- 缺少 SELECT 查询。为兼容原来语法当查询 query 第一个单词不是“SELECT,DELETE,UPDATE“之一时，假定为 WHERE 后面的部分，自动补齐为：”SELECT \”*\” FROM O WHERE ” + query；

基本查询要点

1. 基本查询的目标是主对象，主对象是内置对的，在整个查询中都以“O”代替；
2. 基本查询的属性都是来自主对象属性，使用属性的地方可选的加“O.”前缀；
3. 查询主属性列表中，可以使用属性表达式；
4. 属性表达式必须以双引号(“”)包括，是“*”或者正则表达式，同样可以用“O.”前缀；
5. **简化语法是为了兼容历史用的，不提倡使用；**

4.2.1.2 扩展查询

SELECT {主属性列表}{,扩展属性列表} FROM O {,扩展对象定义} WHERE {对象关联条件}[AND 查询条件] [ORDER BY 排序属性列表[ASC|DESC]]

SELECT {主属性列表}{,扩展属性列表} FROM O [LEFT|RIGHT] JOIN {扩展对象定义} ON {对象关联条件}[WHERE 查询条件] [ORDER BY 排序属性列表[ASC|DESC]]

扩展查询在基础查询上加入其他对象，从扩展对象中被选出的部分属性加入到主对象上形成了一个扩展类。

扩展对象必须以“对象类全路径”+”空白”+“别名”的形式先定义，使用的地方都

用别名，多个对象的定义不能使用同样的别名，对象之间用“，”连接。对象定义以两种形式加入到基本查询中：

- 隐性的 JOIN，扩展对象在 FROM 和主对象之后定义，关联条件在 WHERE 中定义；
- 显式的 JOIN，扩展对象在 JOIN 后定义，关联条件在 ON 中定义。

扩展属性必须以“扩展对象别名”+”.”+”属性名”出现，属性可以出现在查询列表和其他部分，用法和主属性一样。扩展属性在 SELECT 列表中可以用附加{label}的方法增加属性的 Label，这个 Label 会体现在最终的查询结果扩展类的属性中。

扩展查询要点

1. 扩展查询的扩展对象必须先定义后使用；
2. 扩展对象定义的别名不能重复，定义之外地方只能用别名；
3. 扩展属性必须来自定义过的扩展对象，属性必须加“对象别名.”前缀；
4. 扩展属性不能用属性表达式；
5. 除了上述限制外，扩展属性可以和主属性等同使用。
6. 虚拟属性的 label 可以在查询时设置，缺省的是通过 JBO 定义获取。

4.2.1.3 虚拟扩展

虚拟扩展在基础查询和扩展查询的基础上进一步扩展。上节的扩展查询是加入了主对象之外的真实对象，虚拟扩展则是加入了事实上并不存在的虚拟对象的扩展。和主对象相似虚拟对象是内置的一个对象“V“。

SELECT {主属性列表}[,扩展属性列表]{,虚拟字段} FROM {其他基础和扩展查询部分}

- 虚拟对象是内置的标准对象，不需要也不能定义；

- 虚拟对象的属性是动态产生的，只能用在查询列表中；
- 虚拟属性的定义使用 AS 语法，用虚拟对象 V 做前缀标识，格式为：“计算表达式”+“ AS “ +“V.”+ “虚拟属性名”；
- 计算表达式是一个合法的运算表达式，返回结果是一个合法属性类型。运算表达式中只能出现：JBO-QL 中的运算符、函数、实际选择出来的属性、可用的关键字；
- 虚拟属性同样可以用{lable}的形式设置 label。

虚拟扩展限制

1. 虚拟对象“V”是内置对象，不能定义；
2. “V”只能用做虚拟属性的前缀；
3. 虚拟属性只能出现在选择列表中，不可以出现在别的地方；
4. 虚拟属性不能出现在 UNION 查询中
5. 虚拟属性的 label 只能在查询时设置，无法通过 JBO 定义获取。

4.2.2 UPDATE

UPDATE 查询语法用于更新一批符合条件的对象实体。UPDATE 只能对主对象操作，没有扩展对象、扩展属性的概念。**更新语句中必须有 WHERE 条件。**

UPDATE [O] SET {更新项列表} WHERE 查询条件

- 更新对象总是为 O 可以不写；
- 更新项列表是要更新的内容，每个更新项为一个等式，等式左边是属性名称，右边是更新值，更新值允许使用参数，多个更新项用逗号分隔。形式如：a1=’www’,a2=1000,a3=:param1…

- WHERE 条件见 5.2.5 专题说明。

4.2.3 DELETE

DELETE 查询语法用于删除一批符合条件的对象实体。DELETE 只能对主对象操作，没有扩展对象、扩展属性的概念。**更新语句中必须有 WHERE 条件。**

DELETE [FROM O] WHERE 查询条件

- 删除对象总是为 O，可以和 FROM 一起不写，但不能只写一个；
- WHERE 条件见 5.2.5 专题说明。

4.2.4 UNION

UINON 查询本质上是 SELECT 查询的结果集的合并，从应用角度本身不是必须的。此语法用对合并一般查询的结果。

{主查询} UNION {子查询}[UNION 子查询][…]

- 主和子查询都是 SELECT 查询任何一种；
- 主查询决定了查询结果的对象结构，主查询必须直接以 SELECT 开头，不允许有其他附加东西；
- 子查询必须和主查询的属性个数、类型保持一致；
- 所有查询的属性列表都是原样属性，不需要也不允许做 AS 之类的处理。查询解析器以主查询为准自动进行转换处理。

UINON查询限制
1. UNION 查询中不能有虚拟字段。

4.2.5 WHERE 查询条件

WHERE 查询条件在 SELECT、DELETE、UPDATE 中都要用到，并且较为复杂，在此统一介绍。

JBO-QL 的基本查询条件语法和 SQL 一致，是逻辑运算为基础的逻辑表达式。

- 逻辑表达式支持的基础运算包括：**=、<>、<、>、<=、>=、IN、BETWEEN .. AND…、LIKE、IS NULL、EXISTS**
- 支持的联合运算为：**AND、OR、NOT、( )**
- 参与运算的对象包括：**JBO 属性、常量、参数**
- 参与运算的和比较的函数包括：LENGTH（）

4.3 构成元素

JBO-QL 的构成元素有八种：

1. 关键字，KEYWORD。用于构成语法基础，关键字都是英文单词，可以是单个或者双单词。关键字在查询中总是以空白或者操作符前后分割形成独立的单词；
2. 操作符，OPRERATOR。进行逻辑和算术运算的符号，包括用于做连接的逗号和改变运算优先级的括号，部分逻辑运算符是双符号的；
3. 常量，CONSTANT。数值或者字符常量，JBO-QL 中只识别两种常量，以‘’包括的字符串常量，数字序列构成的数字常量；
4. 参数，JBO-QL 中支持参数，一般出现常量的地方都可以用参数替代。参数的形式是：”冒号”+参数标识符;

5. 函数。JBO 查询中只支持 5 中标准函数“"COUNT","SUM","MAX","MIN","AVG","LENGTH"”，但应用可自行注册其他函数；
6. JBO 类。表示一个 JBO 对象的类的名称；
7. JBO 属性。JBO 对象的属性；
8. 空白。字符常量里面之外空白的字符串用做限定符号用于分割其他元素，有些元素不需要之间不需要通过空白分割，但加入空白总是不会影响原有语义。

查询语法中，操作符号除了具有本意的运算功能外，还起到限定符号的作用，和空白一起构成分隔符。

通常关键字、常量、和 JBO 对象的前后是空白或操作符形成的分割符号。

JBO 类、JBO 属性和函数中的标识部分要求必须是字符开头的单词，即符合：[a-zA-Z]\\w*

4.3.1 关键字

QL 的关键字全部为英文单词，在查询中关键字不区分大小写，但内部解析后全部转换为大写。关键字在查询中以空白字符(StringX.isSpace(wb)==true)和操作符号来作为前后界定。关键字定义为：

{"WHERE", "AS", "FROM", "SELECT", "AND", "OR", "UPDATE", "SET", "DELETE", "ORDER", "BY", "ASC", "DESC", "LIKE", "BETWEEN", "IN", "IS", "EXISTS", "NULL", "NOT", "GROUP", "ON", "JOIN", "LEFT", "RIGHT", "UNION", "GROUP", "CASE", "WHEN", "THEN", "ELSE", "END", "DISTINCT", "HAVING"}

4.3.2 操作符

QL 的操作符全部为符号，操作符包括逻辑云运算、算术运算逗号、括号等特殊操作符。

- 单一操作符：**{'(', ')', '*', '+', ',', '-', '/', '<', '=', '>'}**
- 复合操作符： **{“>=”, ”<=”, “<>”}**

4.3.3 函数

JBO 函数是 QL 查询的扩展能力，用于条件过滤中的运算和虚拟字段中。本版本内置函数包括：

{"COUNT","SUM","MAX","MIN","AVG","LENGTH"}

4.3.4 常量

常量是运算和比较的固定数值，在语法解析中不会做任何处理，直接传递个执行阶段。包括两类：

- 字符常量。查询语法中以单引号’’包括起来的部分是字符常量；
- 数字常量。数字字符构成的连续序列成构成数字常量，数字字符只包括 0-9 十个数字，在首个数字之后可以出现小数点“.“，除此之外不允许任何其他字符。当序列遇到其他元素时结束；

4.3.5 参数

JBO-QL 中支持参数，一般出现常量的地方都可以用参数替代。参数采用命名参数的形式，这和 SQL 中的位置参数是不同的。

参数的形式是：**”冒号”+参数标识符。**参数部分类型，格式一致，参数区分大小写。

程序中通过：BizObjectQuery.setParameter(“参数名”,”参数值”)的方法完成。

4.3.6 JBO 类（对象）

JBO 类（习惯称为对象）是 QL 查询的目标。QL 目前语法中支持三种类：

- 主对象类“O”。任何查询都会关联一个内置的主对象，主对象总是用单个字母“O“表示，查询语句中一般不出现主类的名字，内置主类不区分大小写，内部总是以大写表示；
- 虚拟对象类“V”。任何 SELECT 查询都可以适用一个内置的虚拟对象“V”，虚拟对象不经定义即可使用，但仅允许用于主查询的中表示和定义虚拟属性；
- 扩展对象类。扩展类必须先定义后使用，定义的唯一地方是 SELECT 查询（包括在条件中出现的子查询）FROM 或者 JOIN 之后的对象列表，语法格式是：“类全路径”+“空白”+“别名”。定义语法分为两部分：
  - 要引入的类的全路径名称。全路径必须是在 JBOFactory.getClass()方法可以找到的类，因为类全名称区分大小写，必须和 JBO 定义中一致；
  - 别名，是一个简单的字符串，复合 JBO 标识。不区分大小写；

特别注意：

JBO类对象要点

1. 任何查询都有内置的主对象“O”；
2. 任何 SELECT 查询都可以在查询列表中使用虚拟类“V”来定义虚拟属性；
3. 扩展类必须是存在的 JBO 对象实体，需要引入定义后使用；
4. 目前的 JBO 语法限制所有的扩展类必须在 jbo.包下面。

4.3.7 JBO 属性

JBO 属性是 SELECT 查询结果、UPDATE 更新目标，同时也是条件和其他子查询部分中的主要组成部分。所有属性都是来源于对类（对象），根据雷源属性包括了主属性、扩展属性和虚拟属性：

- 主属性。主属性源于查询主对象，主属性可用于任何需要的地方主属性可以用对象限定，也可以不用。在 QL 中以二种形式出现，：
  - 直接书写。即直接写出属性的名字；
  - 属性表达式。用双引号“ ”包括的部分被认为是属性表达式，表达式可以是全统配符号“*”或者正则表达式。显然属性表达式只出现在列表的情况下；
- 扩展属性。扩展属性只能直接写出，不允许用表达式。扩展属性必须使用扩展类别名前缀。形式为：”别名”.”属性名”。使用上
- 虚拟属性。不是来自于对象，通过计算得到的属性，包括：表达式和别名两部分，用 AS 定义。型如：“表达式”AS “V.属性名”。虚拟属性只能用于 SELECT 的选择列表中。

属性类别	来源	书写方式	语法示例	可用环境
主属性	主对象 “O”	属性名称 O+属性名 “”包括的表达式	CustomerID O.CustomerID “*”,”O.*” “O.Customer.*”	SELECT 列表（属性表达式仅用于此） UPDATE 目标 WHERE 条件中 ORDER BY 列表 GROUP BY 列表 函数参数 虚拟属性表达式
扩展属性	扩展对象	X+属性名称	X.ContractNo	SELECT 列表

	(例如“X”)		X.ContractName	WHERE 条件中 ORDER BY 列表 GROUP BY 列表 函数参数 虚拟属性表达式
虚拟属性	虚拟对象 ”V”	表达式+空白 +AS+V.属性名称	‘’ AS V.Attr1 SUM(Blanace) AS V.SumBalance COUNT(*) AS MyCount LENG(CodeName) as CodeLength CASE ….END AS MyAttr	SELECT 列表 **不能用于 UNION 的 SELECT

4.4 特殊字符

成对出现的{}是一个特殊字符，只能用于扩展属性或者虚拟属性的名字之后标识属性的 label。

4.5 书写惯例

JBO-QL 语法书写比较灵活，包括大小写很多情况下可以识别，但为了统一规范，一般遵守下面的惯例：

- 以完整的语法书写语句，避免使用简化查询；
- 关键字在同一个应用中应保持为全部大写或者全部小写；
- 对象别名全部大写；
- 对象属性按照 JBO 定义中的定义书写；

- JBO 函数名字大小写保持和关键字一致。

4.6 限制和扩展

JBO-QL 和 SQL 有相似之处，要注意其限制和扩展。

扩展：

- 支持属性表达式；
- 简化语法；
- 程序级别汇总支持；
- 扩展属性增加自定义 Label 的支持；
- 可扩展的解析器。

限制：

- 查询目标必须是对象，不是数据库的表；
- 查询结果属性在语法上明确区分主属性、扩展属性和虚拟属性；
- 虚拟属性只能在查询结果中，不允许使用；
- 只允许使用 JBO 定义的关键字、操作符和函数；
- UNION 查询中不能使用虚拟属性。

5 基础实现特性

JBO 框架中分为接口定义层和具体实现层，在标准的发布中包含了完整的 JBO 基础实现，丰富的基础功能既可以满足绝大多数应用的需求，同时也是进一步扩展和客户化

的基础。

5.1 工厂实现

5.1.1 实现说明

JBO 工厂抽象类接口 JBOFactory 的基本实现有两个类构成：

- com.amarsoft.are.jbo.impl.DefaultJBOFactory
- com.amarsoft.are.jbo.impl.DefaultXMLJBOFactory

DefaultJBOFactory 定义和实现了一个 JBOFactory 的所有功能，它的子类 DefaultXMLFactory 仅仅实现了从 XML 文件中加载数据的功能，这样的设计目的是提供尽可能多的重用和扩展特性，在这个基础上实现一个 JBOFactory 只需要改变一下初始化时读取外部数据的方法即可。

DefaultJBOFactory 是基于缓冲池技术，在初始化时加载所有的 JBO 对象和 Manager 的定义，在使用时直接从内存产生，保持很高的效率。DefaultJBOFactory 采用的是“定义-引用”的机制实现 JBO 类和 JBOManager 的对应关系，这样只需要很少的全局 Manager 定义，通过不同引用的参数化即可满足“每个类有自己的管理器”的 JBO 基本技术概念。

5.1.2 特性支持

基础 JBOFactory 实现支持以下的技术和功能特点：

- 易于管理的 XML 定义。既可手工编辑也可以用工具操作，同时也便于支持通用版本管理工具如 SVN 等方便协作开发；
- 管理器定义和实例复用。JBO 管理器定义和具体的类管理器分开，通过引用的方式实现公共管理器的复用；

- 智能管理器引用匹配。每个类引用的管理器不必一一定义，工厂通过一套路径匹配规则自动需找合适的管理器引用。其规则如下：
  - 有先指定。如果具体的类中指定了引用的管理器 ID，则该类使用此管理器；
  - 类名匹配。上面规则未成立时，类使用和自己的名字完全一样的管理器定义；
  - 包名路径匹配。的那个上述规则未果时，按照包路径从后向前逐节点进行匹配查找管理器定义；
  - 缺省管理器。如果按照上述规则都不能找到管理器，最终使用 id 为”default”的管理器。
- 支持多文件定义降低维护和管理难度。工厂支持同时对个文件定义 JBO 的 Manager 和 Class，避免定义文件多大造成的管理维护问题。多个文件定义中有重复，则以后加载的定义覆盖先加载的定义。

5.2 对象类扩展

基本 JBO 对象 BizObject 在具体实现中扩展为 com.amarsoft.are.jbo.impl.StateBizObject，此类增加状态标志特性配合对应的 Manger 和 Query 自动识别对象状态来实现对性的持久化管理。

SateBizObject 的状态包括：

- STATE_UNKNOWN = 0; //对象状态表示为未知，即不知道是什么状态;
- STATE_SYNC = 1; //对象状态表示为已同步，和实际存储的是一致的
- STATE_NEW = 2; //对象状态表示为新增，表示实际存储中不存在
- STATE_CHANGED = 3;// 对象状态表示为已改变，表示实际存储中不同

SateBizObjectManager 和 StateBizObjectQuery 对 StateBizObject 进行管理，自动设置和利用状态位进行持久化操作。

5.3 管理器实现

5.3.1 实现说明

JBO 管理器接口 BizObjectManager 的基本实现包括三个基本管理器：

- com.amarsoft.are.jbo.impl. StateBizObjectManager
- com.amarsoft.are.jbo.impl. SingleKeyStateBizObjectManager
- com.amarsoft.are.jbo.impl. ALSBizObjectManager

StateBizObjectManager 实现基本的 StateBizObject 的持久化管理是基本的管理器，另两个类是它的扩展。SingleKeyStateBizObjectManager 在 SateBizObjectManager 的基础上增加创建单 Key 的能力，ALSBizObjectManager 在 SateBizObjectManager 的基础上增加 ALS 标准 Key 创建能力和日期特殊处理能力。

注意 SingleKeyStateBizObjectManager 和 ALSBizObjectManager 是独立的扩展方法，没有继承关系。所有的管理器都实现了事务接口，可以加入事务控制中。

5.3.2 特性支持

基础 StateBizObject 实现以及两个子类支持以下的技术和功能特性如下表：

管理器	继承类和接口	特性支持	配置参数名字	取值
SateBizObject Manager	BizObjectManager BizObjectTableMapper	TableMapper 定义的基本 O-R Map 能力：		
		■ 独立定义的数据库	database	dbconfig.xml 中定义
		■ 对象到表的映射	table	表名字，**缺省为类名**
		■ 属性到字段的映射	attributeMap	{属性名 1,字段名 1}{属性名 2,字段名 2}

		■ 多个对象到单个表的映射，两种配置 filer 形式	filter filterColumn filterValue	字段名，值 字段名 过滤器名字
SingleKeySateBizObjectManager	SateBizObjectManager	扩展实现创建单个关键字的功能： ■ 指定是否创建关键字 ■ 创建关键字的格式，可含格式化日期前缀、格式化序号	createKey keyDatePrefix keyNumberLength	true|**false** **yyyyMMdd** **4**
ALSSateBizObjectManager	SateBizObjectManager	扩展实现 ALS 基本功能： ■ 指定是否创建关键字 ■ 创建标准 ALS 格式的 KEY ■ 日期格式和数据库字符型日期自动转换	createKey keyDatePrefix keyNumberLength	true|**false** **yyyyMMdd** **8**

注意所有属性都是通过具体类的 Managger 属性实现。全局定义的 Manger 属性作为缺省属性，具体类定义的可以覆盖缺省定义。

5.4 查询器实现

基本 JBO 查询在具体实现中扩展为两个实现类：

- com.amarsoft.are.jbo.impl.BaseicSqlQuery
- com.amarsoft.are.jbo.impl.SateBizObjectQuery

BasicSqlQuery 实现把 JBO-QL 转换为数据 SQL 查询的功能，实现 JBO-QL 的执行，同时封装查询结果为基本 BizObject。基本实现完成了一个查询器需要所有功能，是实现其他基于 SQL 查询的基础。

StateBizObjectQuery 并不是 BasicSQLQuery 的扩展类，它被设计为一个 Wrap 类，必须用另外的查询对象来构造本类。Wrap 类本身并不实现任何查询语句的功能，相反它利用构造时传入的查询对象执行查询，然后把查询结果封装为一个 SateBizObject 的实例，这些实例可以被 StateBizObjectManager 管理。

6 环境配置

本章说明的配置是针对实现而言，JBOFactory 是标准的 ARE 服务，其配置包括服务本身的配置和 JBO 对象及管理器的配置。

6.1 JBO 服务配置

JBO 服务配置遵从标准的 ARE 服务配置规则和约定。配置信息一般在 are.xml 中的<AREServices>节点下，包括：

- 服务 ID 定义:：包括服务 ID(id=”JBO”)、是否有效(enabled=true|false)、是否初始化启动（initOnStart=true|false）；
- 服务说明信息：Provider、Version、Describe；
- 服务类，缺省的是内置的工厂 XML 类

  <ServiceClass>com.amarsoft.are.jbo.impl.DefaultXMLJBOFactory</ServiceClass>
- 属性中则定义了工厂类要使用的信息，缺省的为：配置文件、缺省包。

```
<!-- Java业务对象管理服务。提供Java业务对象管理方法。
   应用程序可以通过JBO访问业务对象而不需要对数据库做操作。
-->
<Service id="JBO"  enabled="true" initOnStart="true">
   <Provider>Amarsoft</Provider>
   <Version>1.0</Version>
   <Describe>业务对象管理服务</Describe>
   <ServiceClass>com.amarsoft.are.jbo.impl.DefaultXMLJBOFactory</ServiceClass>
   <Properties>
      <Property name="com.amarsoft.are.jbo.impl.DefaultXMLJBOFactory.configFile"
         value="{$ARE.APP_HOME}/etc/jbo.xml,{$ARE.APP_HOME}/etc/jbo_customer.xml"/>
      <Property name="com.amarsoft.are.jbo.impl.DefaultXMLJBOFactory.defaultPackage" value="jbo.sample.customer"/>
   </Properties>
</Service>
```

配置文件可以配置为多个逗号分隔的文件，文件路径中可以使用标准的 ARE 变量。

6.2 基于 XML 的对象配置

DefaultXMLJBOFactory 配置文件的根节点是：<jbo>，分为<managers>和<pakage>两个部分实现。<managers>下面定义所有的全局<manager>，<package>可以有多个，每个<package>下定义多个<class>。基本文件结构如下：

```
<?xml version="1.0" encoding="GB18030"?>
<jbo>
    <managers>
        <manager id="default" managerClass="">
            <describe>缺省的通用状态 Object 管理器</describe>
            <managerProperties>
                <property name="database" value="sample_db"/>
            </managerProperties>
        </manager>
    </managers>
    <package name=”jbo.package1”>
        <class name=" " label=" " keyAttributes=" " briefAttributes="employeeId,PID">
            <attributes>
                <attribute name="employeeId" label="员工编号" type="STRING" length="4"/>
                …
            </attributes>
            <manager id="createKey">
                <managerProperties>
                    <property name="database" value="sample_db"/>
                </managerProperties>
            </manager>
        </class>
    </package>
    <package name=” jbo.package1”>
    </package>
</jbo>
```

mangers 部分和 package 部分可不必要同时出现，一般在比较大的应用中我们通常把 manager 的定义单独放一个文件，package 放在其他文件中。

6.2.1 全局 Manger 定义

全局 Manager 在文件中定义在<managers>节点下，每个 manager 有不同的 id，包括基本的定义和属性定义。

```
1  <?xml version="1.0" encoding="GB18030"?>
2  <jbo>
3    <managers>
4      <!--
5        管理器实现每个对象的查询、创建、保存、删除功能，每个对象都有对应的管理器。
6        在DefaultJBOFactory中所有的管理器实例必须在 <managers>节定义，具体每个类
7        使则在manager一节通过id引用此节定义的manager，通过managerProperties
8      -->
9
10     <!-- 缺省的通用管理器，整个JBO系统必须有一个ID为default的管理器处理一般的类的管理 -->
11     <manager id="default" managerClass="com.amarsoft.are.jbo.impl.StateBizObjectManager">
12       <describe>缺省的通用状态Object管理器</describe>
13       <managerProperties><property name="database" value="sample_db"/></managerProperties>
14     </manager>
```

管理器定义包括：

- 基本定义。是<manager>节点内置属性，包括表示 id 和管理类 managerClass (Line 11)；
- 描述。节点<describe>定义，可选(Line 12)；
- 属性。在<managerProperties>一节内定义，每个属性是一个<property>节点，内属性 name 和 value 定义了属性的名字和值(Line 13)。

所有 Manager 的定义形式都是一样的，基本的 id 和类决定了其实例，每个管理类需要的特别信息都是通过<managerProperties>下数量不限的<property>支持。基础 manager 支持的属性在 5.3.2 中的表格中定义。配置文件片段如下：

```
138        <managerProperties>
139          <property name="database" value="sample_db"/>
140          <property name="table" value="EMPLOYEE"/>
141          <property name="filter" value="Sex,0"/>
142          <property name="attributeMap" value="{employeeName,name}{depart,department}"/>
143          <property name="updateIncludes" value="*"/>
144          <property name="updateExcludes" value="Name"/>
145          <property name="createKey" value="true"/>
146          <property name="keyDatePrefix" value=""/>
147          <property name="keyNumberLength" value="4"/>
148        </managerProperties>
```

```
178         <managerProperties>
179           <property name="database" value="sample_db"/>
180           <property name="table" value="EMPLOYEE"/>
181           <property name="filterColumn" value="Sex"/>
182           <property name="filterValue" value="1"/>
183           <property name="attributeMap" value="{employeeName,name}{depart,department}"/>
184           <property name="updateIncludes" value="*"/>
185           <property name="updateExcludes" value="Name"/>
186           <property name="createKey" value="true"/>
187           <property name="keyDatePrefix" value=""/>
188           <property name="keyNumberLength" value="4"/>
189         </managerProperties>
```

6.2.2 JBO 类定义

每个 JBO 类定义在自己所属的 package 下面，类定义包括基本定义、属性定义和管理器定义三个部分。

```
207 <package name="jbo.sample.project">
208   <class name="Project" label="项目信息" keyAttributes="projectId" briefAttributes="projectId,customerId,projectName">
209     <attributes>
210       <attribute name="projectId" label="项目编号" type="STRING"  length="20"/>
211       <attribute name="customerId" label="客户号" type="STRING" length="20"/>
212       <attribute name="projectName" label="项目名称" type="STRING" length="60"/>
213       <attribute name="startDate" label="启动日期" type="DATE" length="8"/>
214       <attribute name="planFinishDate" label="预期完日期" type="DATE" length="8"/>
215       <attribute name="actualFinishDate" label="实际完成日期" type="DATE" length="8"/>
216       <attribute name="status" label="当前状态" type="STRING" length="1"/>
217       <attribute name="manager" label="项目经理" type="STRING" length="10"/>
218       <attribute name="budget" label="项目预算" type="DOUBLE" length="20" scale="0"/>
219       <attribute name="memo" label="备注" type="STRING" length="200"/>
220     </attributes>
221     <manager>
222       <managerProperties>
223         <property name="createKey" value="true"/>
224         <property name="keyDatePrefix" value="yyyyMMdd"/>
225         <property name="keyNumberLength" value="4"/>
226       </managerProperties>
227     </manager>
228   </class>
229
230   <class name="ProjectMember" label="项目成员表" keyAttributes="projectId,employeeId" briefAttributes="ProjectId,employeeId">
231     <attributes>
232       <attribute name="projectId" label="项目编号" type="STRING"  length="20"/>
233       <attribute name="employeeId" label="员工编号" type="STRING" length="18"/>
234       <attribute name="position" label="岗位" type="STRING" length="20"/>
235     </attributes>
236   </class>
237 </package>
```

定义说明如下：

- 基本定义。每个节点<class>定义一个类，基本定义包括 name、label、keyAttributes、

breidfAttributes，这些属性时<class>节点的内置属性(Line 208,230)；

- 属性定义。类属性在<attributes>节点下定义，每个节点定义一个属性。属性定义包括：name，label，type，length，scale(double 型)，它们都是以内嵌属性的方式定义在<attribute>节点内。type 值包限于：<INT|LONG|DOUBLE|STING|DATE|BOOLEAN>(Line 210-219,232-234)；
- 管理器定义。管理器定义在<class>节点下的<manager>节点定义，包括 ID 和属性。ID 由其内嵌属性定义<manager id=”myid”>，引用全局 Manager 一节定义的 id。如果每有定义 ID，则按照 5.1.2 节说明的只能匹配方式匹配。属性一节有<managerProperties>节点定义，含义和形式完全和 6.2.1 中 Manager 定义的描述一致。此处的定义值覆盖 Manager 原始的定义（Line223-225）。

对于简单的类管理，甚至可以用任何 manager 定义即可实现基本管理功能。如图中的 ProjectMember 类，完全使用只能寻找和缺省配置。

对 attribute，在应用中可以添加特别 uiHints 属性和扩展属性来满足特别应用的需要，这些属性可在程序中应用捕获以满足特别的用途。

```
<attribute name="employeeName" label="姓名" type="STRING" length="20">
  <uiHints required="true" sortingSuitable="true" filterSuitable="true" valueCharacter="" validator="" codeTable="" boundValues="" displayOnly="true" displayFormat=
    <customerHints>
      <hint name="myhint" value="hintvalue"/>
    </customerHints>
  </uiHints>
</attribute>
<attribute name="sex" label="性别" type="STRING" length="1">
  <uiHints required="true" sortingSuitable="true" filterSuitable="true" valueCharacter="codetable"  codeTable="inline,{M,男}{F,女}" boundValues="{M}{F}" />
</attribute>
<attribute name="birthday" label="生日" type="DATE" length="8"/>
<attribute name="education" label="学历" type="STRING" length="1">
  <uiHints  valueCharacter="codetable"  codeTable="datasource,datasource:db:als:select itemCode,ItemName from CODE_LIBRARY where TypeNo='EDUCATION'" />
</attribute>
<attribute name="workYear" label="工作年份" type="INT" length="4"/>
```

上图中的 employeeName，sex 和 education 定义了 uiHints

```
30    <class name="DMResponseHeader" label="反馈报文头" describe="DM通用反馈报文头">
31      <attributes>
32        <attribute name="TradeNo" label="交易代码" type="STRING" length="4"/>
33        <attribute name="SendTime" label="发送时间" type="DATE" length="14">
34          <extendProperties>
35            <property name="format" value="yyyyMMddHHmmss"/>
36          </extendProperties>
37        </attribute>
38        <attribute name="SubSystem" label="系统标识" type="STRING" length="2"/>
39        <attribute name="RspCode" label="错误代码" type="STRING" length="4"/>
40        <attribute name="RspMsg" label="错误信息" type="STRING" length="40"/>
41      </attributes>
42    </class>
```

上图的第 34-36 行为属性 SendTime 定义了一个 format 的属性，实时接口程序利用这个属性来生成报文头中的时间戳。