EasyDBO 简易教程 1.0 版

草稿

(官方网站：www.easyjf.com)

简易、实用才是硬道理!打造超轻量级的简捷 ORM 系统！

EasyDBO 最后一个测试版本隆重发布！感谢您对 EasyJF 开源及 EasyDBO 项目的关注与支持。

“ 国产开源，我们为梦而生 ”，革命尚未未成功，同志仍需努力。就像张靓颖在《梦想》中唱的一样 :“ 拥有梦想，就有可能! ”，EasyJF 开源才刚刚开始,EasyDBO 的发展需要您的支持及参与！

EasyJF 开源
2006 年 10 月

修 订 页

编号	章节名称	修订内容简述	修订日期	修订前版本号	修订后版本号	修订人	批准人
1	序言、总目录	草稿	2006-10-10		0.9	大峡	
2	与 strut 集成		2006-10-16		1.0	天意	
3	与 spring 集成、关系映射修改		2006-10-15		1.0	stef_wu	
4	结束语及校对		2006-10-17		1.0	天一	
5	多对多及一对多等		2006-10-17		1.0	大峡	
6	1.0 版审稿		2006-10-17		1.0	大峡	

EasyDBO 简易教程 1.0 版

(www.easyjf.com)

目 录

序言......3
一、EasyDBO 简介......4
1.1 对象-关系映射简介......4
1.2 如何实现对象-关系映射......4
1.3 为什么要设计 EasyDBO......4
1.4 EasyDBO 简介......5
二、EasyDBO 快速上手及示例......5
2.1 获取 EasyDBO SDK 及源代码......5
2.2 EasyDBO 文件组成......5
2.3 编译安装 EasyDBO......5
2.4 使用 EasyDBO......6
2.5、开始简单的映射......6
2.6、使用 EasyDBO 总体流程......8
三、EasyDBO 使用详解......8
3.1 ORM 核心引擎 EasyJDB 的创建......8
3.1.1 映射处理核心引擎 EasyJDB 类简介......8
3.1.2 创建 EasyJDB 对象实例......9
3.1.3 指定映射配置文件......10
3.1.4 指定缓存......10
3.2 数据源设置......11
3.3 数据库查询方言......11
3.4 实(域)对象 PO 及泛对象 FO(Fan Object)......12
3.5 使用接口实现简单映射......12
3.6 使用配置文件配置映射关系......14
3.7 使用 Java5 注解来设置映射......15
3.8 主键配置......16
3.9 缓存设置......16
3.10 一对一映射......17
3.11 一对多映射......19
3.12 多对多映射......21
3.13 延迟加载......25
3.14 事务处理......26
3.15 使用存储过程......26
四、EasyJDB 中常用方法介绍......27
4.1 对象添、删、改......27
4.2 从持久层读取单个域对象 DOM(PO)......27
4.3 从持久层读取多个域对象 DOM(PO)......28
4.4 从持久层读取多个泛对象 FO(Fan Object)......29

4.5 从持久层返回唯一对象 30
4.6 执行自定义 sql 语句 30
4.7 存储过程 30
五、EasyDBO 以其它框架的集成运用 31
5.1 EasyDBO 与 EasyJWeb 集成 31
5.2 EasyDBO 与 Spring 集成 33
5.3 在 Struts 中使用 EasyDBO 37
六、使用 EasyDBO 的开源项目介绍 38
6.1 EasyJF 开源 Blog 系统 38
6.2 简易 java 框架开源论坛系统(最新版本 0.5,更新时间 10 月 1 日) 38
6.3 简易 java 框架开源订销管理系统(最新更新:2006-4-3) 38
6.4 EasyJF 开源网上会议系统 iula-0.1.0(最新更新:2006-7-13) 39
七、结束语 39
八、联系我们 39

序言

EasyDBO 是一个非常适合中小型软件数据库开发的数据持久层框架，系统参考 hibernate、JDO 等，结合中小项目软件的开发实际，实现简单实用的对象-关系数据库映射。本文主要介绍 EasyDBO 的使用方法，作为初学者的快速上手指南。由于 EasyDBO 是一个不断更新的开源项目，本教程中一些新特性、功能的介绍及应用将不定期的在 EasyDBO 官方网站上提供。

该教程由 EasyJF 开源团队－EasyDBO 项目组编写,免费在各大 Java 专业网站及其它专业媒体上发表。

当前参与该文档编写的 EasyJF 成员有(排名不分先后)：
大峡、stef_wu、天意、天一及其它 EasyJF 团队成员
欢迎更多的朋友加入到我们的写作当中!

EasyDBO 项目开发人员

大峡、piginzoo、william、stef_wu、天意、散仙、qgz0910、clyyu 等

一、EasyDBO 简介

1.1 对象-关系映射简介

对象-关系映射（Object/Relation Mapping，简称 ORM），是随着面向对象的软件开发方法发展而产生的。面向对象的开发方法是当今企业级应用开发环境中的主流开发方法，关系数据库是企业级应用环境中永久存放数据的主流数据存储系统。对象和关系数据是业务实体的两种表现形式，业务实体在内存中表现为对象，在数据库中表现为关系数据。内存中的对象之间存在关联和继承关系，而在数据库中，关系数据无法直接表达多对多关联和继承关系。因此，对象-关系映射(ORM)系统一般以中间件的形式存在，主要实现程序对象到关系数据库数据的映射。

在开发关系数据库的系统时，可以通过 SQL 语句读取及操作关系数据库数据。在 Java 领域，可以直接通过 JDBC 编程来访问数据库。JDBC 可以说是 JAVA 访问关系数据库的最原始、最直接的方法。这种方式的优点是运行效率高，缺点是在 Java 程序代码中嵌入大量 SQL 语句，冗余是不可避免的，开发人员常常发现自己在一次又一次地编写相同的普通代码，如获得连接、准备语句、循环结果集以及其他一些 JDBC 特定元素，使得项目难以维护。特别是当涉及到非常多的关系数据表、需要在多个不同类型的关系数据库系统中使用时，通过在程序中使用 JDBC 开发实施起来更加困难。

在开发基于数据应用为主的软件系统时，引入对象－关系映射中间件是提高开发效率、提升软件产品的可维护、扩展性的现实需要。实践表明，在基于数据处理为主的企业级应用程序开发中，通过引入对象-关系映射中间件，可以节省与对象持久化相关的差不多 35%的编程工作量，同时提升软件产品可维护及易扩展性，提升软件产品质量。

1.2 如何实现对象-关系映射

在开发企业级应用时，有必要通过引入对象-关系映射系统中间件，实现数据库的快速开发。企业可以通过 JDBC 编程来开发单独的持久化层，把数据库访问操作封装起来，提供简洁的 API，供业务层统一调用，实现自己的 ORM 系统中间件。

当然，一个成熟的对象-关系映射中间件产品，不只是简单的把内存中的对象持久化到数据库、把数据库中的关系数据加载到内存中，还要保证系统频繁地访问数据库的性能，降低访问数据库的频率，需要引入多线程、缓存、事务管理等很多细节，涉及到的技术比较复杂，因此，我们更多是使用市场上优秀的 ORM 系统中间件产品。

当前主流的 ORM 中间件产品主要有：

Hibernate
EasyDBO
iBatis
JDO
EJB Entities 3
EJB Entity Beans 2.x
TopLink

1.3 为什么要设计 EasyDBO

要使用 java 访问数据库，我们都知道使用 JDBC，然而 jdbc 相对来说毕竟是一个比较底层的数据库访问 API，在实际应用中需要写非常多的 SQL 语句，管理数据源、处理数据

访问异常等诸多工作，造成开发的极为不便。而且，由于不同数据库之间的 sql 语句存在一定的差异，也使得我们写的程序难免捆绑到某一种数据库上，无法实现程序灵活的在不同的数据库之间进行移植。

近年来，对象关系映射系统得到广泛的应用。随着以 iBatis、hibernate 为代表的轻量级 ORM 系统普及，我们基本上可以告别 JDBC，告诉烦琐的 SQL 语句，同时也解决了前面提出的问题。然而，在软件的领域，永远没有完美的系统解决方案，随着我们在大量的项目中运用这些 ORM 系统，我们又发现了很多新的问题与苦恼。如：烦琐的配置文件、机器语言一样 sql 语句、数据访问的效率、性能问题以及难于发现的错误等等。

在一些中小型的应用软件中，一方面由于存在诸多因素，使得需求往往存在不确定性，另一方面软件正向更加专业化方向发展，软件的需求越来越复杂。如果每次变动都需要经过修改域模型、改配置文件、改业务逻辑等多个机械的步骤及环节，显得难以适应。因此，要求我们有一个能快速适应变化的超轻量级 ORM 系统，适应我们快捷开发的需求。

1.4 EasyDBO 简介

EasyDBO 是基于 java 技术，应用于 Java 程序中实现快速数据库开发的对象-关系映射 (ORM)系统框架。从本质上说，EasyDBO 是一个对 JDBC 的简单封装，通过借鉴当前的主流 ORM 系统，引入了更加简单实用的方式来实现对象及关系数据库的映射，从而使得我们在开发中可以把精力更多的集成中在域建模及软件业务逻辑上面。

EasyDBO 是一个超轻量级的 ORM 系统，其定位于解决中小型系统项目中的对象关系映射。提供更加简便、灵活的映射方式，把实际应用中的最佳实践融入到 ORM 的设计中，从而满足快捷开发的要求，即快速、简捷的完成应用软件开发。

EasyDBO 对外提供一用于处理对象－关系映射等的核心引擎，我们通过使用该引擎即可实现关系数据库的相关操作。若结合 EasyJF 所提供的其它的开源框架如 EasyJWeb 等使用，则可以在实际开发中大大提高开发效率。

二、EasyDBO 快速上手及示例

2.1 获取 EasyDBO SDK 及源代码

EasyDBO 作为国内的一个 Java 开源项目，可以通过其开发团队的官方网站 www.easyjf.com 中下载，下载地址:http://www.easyjf.com/easydbo/download.htm。当然，也可以直接从 SVN 上直接 Check out 该项目最新的源代码，EasyDBO 的 SVN 地址：http://svn.easyjf.com/repository/easyjf/easydbo/

2.2 EasyDBO 文件组成

下载的文件主要包括

lib 为编译 EasyDBO 所需要的支持库；

src 目录为 EasyDBO 框架的全部源代码、测试代码及示例代码；

bin 目录为 EasyDBO 的发布命令及脚本等。

2.3 编译安装 EasyDBO

一般情况下，我们直接下载整个 EasyDBO 项目的源代码，然后在自己的机器上根据 JDK 重新编译一次。通过执行 bin 里面的 build.bat jar，或者双击 build.bat，然后选择 jar，即可执行 EasyDBO 的编译工作。

如下图所示：

```
EasyJF
Buildfile: ..\build.xml

usage:
     [echo] EasyJF Build 文件
     [echo] -----------------------------------------------
     [echo] 可选编译选项
     [echo] compile          --> 编译Java源文件
     [echo] jar              --> 生成jar包
     [echo] javadoc          --> 生成API文档
     [echo] maven-jar        --> 用maven管理jar包
     [echo] test             --> 运行Junit测试
     [echo] test-report      --> 生成Junit测试报告
     [echo] eclipse          --> 生成Eclipse项目文件
     [echo] myeclipse        --> 生成MyEclipse项目文件
     [echo] idea             --> 生成Idea5项目文件
     [echo] jbuilder         --> 生成Jbuilder2006项目文件
     [echo] netbean          --> 生成NetBean5项目文件
     [echo] clean            --> 清理已编译的文件
    [input] 请输入一个您要执行的任务, 可选择的有: (compile,jar,war,javadoc,maven
-jar,test,test-report,eclipse,myeclipse,idea,jbuilder,netbean,clean)
jar


initial:
    [mkdir] Created dir: D:\easyjf\easydbo\release
```

编译完成后，若输入的是 jar 命令。则会生成一个 release 目录，其中有一个名为 easyjf-dbo-1-0-0.jar 的文件，其中后面的数字表示版本号。

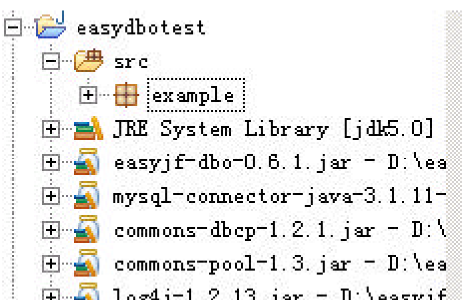
（注：由于 EasyDBO 中支持使用 Java5 注解来配置对象关系映射，因此，需要在 Java5(jdk1.5)以上的环境进行编译，才能使用全部的功能。）

2.4 使用 EasyDBO

要在项目中使用 EasyDBO，把 easyjf-dbo.jar、cglib-2.1.3.jar 与 log4j 日志的 jar，添加到你项目的类路径或 classpath 中即可。

当然，由于涉及到数据库开发，还必须保证您所用的数据库驱动包、连接池驱动包也需要存放于类路径上。

下图是我们使用 My SQL 数据，使用 apache-dbcp 连接池来处理数据库的项目中，使用 EasyDBO 所需要的最少的 jar 文件。



2.5、开始简单的映射

下面是我们使用 EasyDBO 的一个简单例子。我们以一个留言板表 Message 为例，首先定义一个表示留言板信息的持久层对象 PO，内容是一个简单 JavaBean，由于我们使用比较

简单的接口映射方式来实现映射关系，所以这个 Java Bean 还实 IObject 接口。Message.java 的内容大致如下：

```
package example;
import java.util.Date;
import com.easyjf.dbo.IObject;
public class Message implements IObject {
    private int cid;
    private String title;
    private String content;
    private String inputUser;
    private Date inputTime;
    private Boolean publish;
    private Integer status;
    public String getTableName() {
        return "message";
    }
    public String getKeyField() {
        return "cid";
    }
    public String getKeyGenerator() {
        return "com.easyjf.dbo.NullIdGenerator";
    }
    public int getCid() {
        return cid;
    }
    public void setCid(int cid) {
        this.cid = cid;
    }
...省略 getter 及 setter 方法。
}
```

下面，我们写一个简单的演示代码，看看在程序中如何非常简单的把留言板信息保存到数据库中，也即使用 EasyDBO 自动实现对象及关系表之间的映射。示例代码如下：

```
package example;
import org.apache.commons.dbcp.BasicDataSource;
public class MessageTest {
    public static void main(String[] args) {
        //首先准备一个数据源
        BasicDataSource datasource = new BasicDataSource();
        datasource.setDriverClassName("org.gjt.mm.mysql.Driver");
        datasource.setUrl("jdbc:mysql://127.0.0.1:3306/easyjf");
        datasource.setUsername("root");
        datasource.setPassword("mysql");
        //使用数据源创建一个EasyDBO映射处理引擎EasyJDB对象
```

```
        com.easyjf.dbo.EasyJDB easyjdb=new
com.easyjf.dbo.EasyJDB(datasource);
        Message m=new Message();
        m.setTitle("留言标题");
        m.setContent("留言内容");
        m.setInputTime(new java.util.Date());
        m.setInputUser("easyjf");
        m.setPublish(Boolean.TRUE);
        m.setStatus(new Integer(0));
        //使用EasyDBO映射处理引擎执行相关的数据持久化操作
        boolean ret=easyjdb.add(m);
        if(ret)System.out.println("成功写入数据！");
        //从数据库中读取对象
        java.util.List list=easyjdb.query(Message.class,"1=1");
        Message m2=(Message)list.get(0);
        System.out.println(m2.getTitle());
        System.out.println(m2.getContent());
    }
}
```

输出结果：

成功写入数据！
留言标题
留言内容

2.6、使用 EasyDBO 总体流程

1、选择三种映射方式中的一种，书写域对象(DOM或PO)，即普通的Java Bean。若选择的映射方式是接口方式，则要求Java Bean实现IObject接口；若使用Java5注解来定义映射，则要求在JavaBean中使用相关的标签来指定映射方式；若采用配置文件方式来定义映射，则要求在easyjf-dbo.xml文件中定义对象－关系表的映射。

2、在应用程序中创建一个EasyJDB实例，设置数据源、加载配置文件(如果需要的话)，然后设置EasyJDB相关参数，如缓存、是否在控制台回显Sql语句、默认事务处理方式等。

3、使用EasyJDB来执行对象的添、删、改、查等操作。

4、在需要的时候，也可以通过EasyJDB来执行传统的SQL语句、自定义复杂的SQL查询以及存储过程调用等。

三、EasyDBO 使用详解

3.1 ORM 核心引擎 EasyJDB 的创建

3.1.1 映射处理核心引擎 EasyJDB 类简介

在 EasyDBO 中，直接负责映射处理的类是 EasyJDB，我们在应用程序中只需要取得一

个映射引擎的实例，即可使用该对象实现对象－关系数据库的映射操作。

EasyJDB 中的方法大致可以归为三种类型：

1、第一种是把对象持久化到数据库中的相关方法，也是直接执行一定数据库操作的方法，如 add、update、del、saveOrUpdate、execute 等。比如使用 easyjdb.add(m)，实现把域对象 m 中的数据保存到持久层存储设备数据库中。

2、第二种是把从持久层存储设备(数据库)中查询域对象的方法。如 query、uniqueResult、read 等。比如 easyjdb.read(Mesage.class,"2")，可以从数据库中读取主键值为 2 的 Message 对象。

3、最后一种是关于映射处理引擎参数设置的方法，包括 setShowSql、setEnableCache、setAutoCommit、`setMaxSize` 等方法。比如 setEnableCache 方法可以开取或关闭 EasyDBO 中的缓存。

3.1.2 创建 EasyJDB 对象实例

为了能够使用 EasyJDB 来处理对象到关系数据库的映射，我们需要在应用程序中取得一个 EasyJDB 的实例。在 EasyDBO 中，主要有两种方法来构造 EasyJDB 实例。

第一种方法：通过 EasyJDB 构造子来创建 EasyJDB 实例

在构造 EasyJDB 实例的时候，需要提供至少一个可能的数据源作为参数。EasyJDB 提供了如下几个:

EasyJDB(javax.sql.DataSource dataSource)

根据一个数据源来创建一个处理默认方言为 MySQL 数据类型方言的 EasyJDB 实例；

EasyJDB(javax.sql.DataSource dataSource, java.lang.String cacheName)

根据一个数据源来创建一个 EasyJDB 实例，并指定所要使用使用缓存的缓存名称；

EasyJDB(javax.sql.DataSource dataSource, ISqlQuery sqlQuery)

根据数据源及方言来创建一个 EasyJDB 实例。

EasyJDB(javax.sql.DataSource dataSource, ISqlQuery sqlQuery, java.lang.String cacheName)

根据数据源、数据库方言、缓存名称等参数，创建 EasyJDB 实例。

EasyJDB(javax.sql.DataSource dataSource, ISqlQuery sqlQuery, java.lang.String cacheName, boolean showSql)

根据数据源、数据库方言、缓存名称、是否在后台回显 EasyDBO 生成的 sql 语句等参数，创建 EasyJDB 实例。

EasyJDB()

默认构造函数，构造出的实例后需要通过手动设置正确的数据源及数据库方言，EasyJDB 引擎才能正常工作。

下面是几种正确的构造 EasyJDB 实例的示例：

```
BasicDataSource datasource = new BasicDataSource();
…
EasyJDB easyjdb=new com.easyjf.dbo.EasyJDB(datasource,new com.easyjf.dbo.sql.MSSqlServerQuery());
Message m=(Message)easyjdb.read(Message.class,cid);

EasyJDB easyjdb=new com.easyjf.dbo.EasyJDB(datasource);
```

```
    easyjdb.setSqlQuery(new MSSqlServerQuery())
    Message m=(Message)easyjdb.read(Message.class,cid);

    EasyJDB    easyjdb=new    com.easyjf.dbo.EasyJDB(datasource,new
MSSqlServerQuery(),"EasyDBO",true);
    Message m=(Message)easyjdb.read(Message.class,cid);
```

第二种方法，使用 EasyJDB 的工厂方法创建 EasyJDB 实例

EasyJDB 中有一个名为 getInstance 的静态方法，该方法可以直接从 EasyJDB 的默认配置文件中加载数据源及相关配置属性，创建一个 EasyJDB 对象。在只有一个数据库的情况下，通过在配置文件 easyjf-dbo.xml 中配置相关的 EasyDBO 参数，然后使用 getInstance 方法得到 EasyJDB 实例。

使用方法：

```
EasyJDB easyjdb=EasyJDB.getInstance();
Message m=(Message)easyjdb.read(Message.class,cid);
```

3.1.3 指定映射配置文件

在程序构造完 EasyJDB 对象以后，可以通过 setConfFiles 来指定 EasyDBO 的映射配置文件，这样 EasyDBO 会首先加载这些配置文件中的映射关系。如下面代码所示：

```
    public void testConfigFile()
{
    BasicDataSource datasource = new BasicDataSource();
    datasource.setDriverClassName("org.gjt.mm.mysql.Driver");
    datasource.setUrl("jdbc:mysql://127.0.0.1:3306/easyjf");
    datasource.setUsername("root");
    datasource.setPassword("mysql");
    EasyJDB easyjdb=new EasyJDB(datasource);
    java.util.List configFile=new java.util.ArrayList();
    configFile.add("/easyjf-dbo.xml");
    configFile.add("/easyjf-dbo2.xml");
    easyjdb.setConfigFiles(configFile);
    System.out.println(easyjdb.getMapping().getMap().size());
}
```

通过使用 easyjdb.setConfigFiles 方法，把 /easyjf-dbo.xml 及 /easyjf-dbo2.xml 两个配置文件的内容都加载到 EasyJDB 引擎中。

3.1.4 指定缓存

也可以构造完 EasyJDB 引擎后，在程序中手动构造缓存。直接设置 EasyJDB 的 innerCache 属性值，然后再把 enableCache 设置成 true 即可。innerCache 是一个 DBOCache 类型，DBOCache 中持有一个 ICache 作为真正的 Cache。因此，程序中指定缓存的步骤如下：

```
    public void testCustomCache()
```

```
{
    EasyJDB easyjdb=new EasyJDB(datasource);
    com.easyjf.cache.ICache cache=new com.easyjf.cache.EasyCache();
    com.easyjf.dbo.DboCache dcache=new com.easyjf.dbo.DboCache();
    dcache.setCache(cache);
    easyjdb.setInnerCache(dcache);
    easyjdb.setEnableCache(true);
    easyjdb.setShowSql(true);
    easyjdb.setConfigFiles(configFile);
    for(int i=0;i<5;i++)
    {
        easyjdb.query(Message.class,"1=1");
    }
}
}
```

3.2 数据源设置

在实际应用中，一个 EasyJDB 实例与一个数据源进行绑定，这里指的 EasyJDB 的数据源也即 javax.sql.DataSource 的一个实例。要能正确使用 EasyDBO 访问数据库，必须有一个有效的数据源。这些数据源可以是一个普通的 Apache DBCP 数据库连接池，也可以是一个存在其它容器中的 JNDI 数据源，还可以是用户自己实现的数据库连接池。

例 1、使用 Apache DBCP 连接池数据源

```
import org.apache.commons.dbcp.BasicDataSource;
BasicDataSource mssql = new BasicDataSource();
mssql.setDriverClassName("net.sourceforge.jtds.jdbc.Driver");
mssql.setUrl("jdbc:jtds:sqlserver://127.0.0.1:1433;DatabaseName=easyjf;SelectMethod=cursor");
mssql.setUsername("easyjf");
mssql.setPassword("easyjf");

EasyJDB easyjdb=new EasyJDB();
easyjdb.setDataSource(mssql);
easyjdb.setSqlQuery(new MSSqlServerQuery());
```

例 2、使用 JNDI 数据源

```
Context ctx=null;
ctx=new InitialContext();
DataSource ds=(DataSource)ctx.lookup("java:comp/env/jdbc/easyjf");
EasyJDB easyjdb=new EasyJDB(ds,new MSSqlServerQuery());
```

3.3 数据库查询方言

由于每一种数据库的 SQL 语句及语法存在一定的差异。因此，为了能同时兼容不同的

数据库，EasyDBO 中专门设计了一个方言类来支持不同的数据库。在构造 EasyJDB 实例的时候，需要针对不同的数据库，设置与其相对应的数据查询方言，EasyJDB 才能完全正常地工作。默认情况下，EasyJDB 使用 My SQL 方言 MySqlQuery 作为 EasyDBO 的方言。

方言是通过 com.easyjf.dbo.sql.ISqlQuery 接口定义的，若要支持不同的数据库，则用户可以根据该接口实现自己的方言即可。

当前，我们已经提供了 MSSqlServerQuery、MySqlQuery、OracleQuery 等几种方言，分别用于支持 MS SQL(ACCESS)、MY SQL、Oracle 等几种不同类型的数据库。

设置 EasyJDB 的引擎方言的方法如下：

1、直接在构造函数指定使用的方言

```
EasyJDB easyjdb=new EasyJDB(datasource,new MSSqlServerQuery());
```

2、使用 EasyJDB 类的 setSqlQuery 设置查询方言

```
EasyJDB easyjdb=new EasyJDB(dataSource);
easyjdb.setSqlQuery(new MSSqlServerQuery());
```

3、在 easyjf-dbo.xml 文件中配置方言，然后使用 getInstance 方法取得 EasyJDB 实例。

easyjf-dbo.xml 中配置方言

```
<property
name="easydbo.dialect">com.easyjf.dbo.sql.MSSqlServerQuery</prope
rty>
EasyJDB easyjdb=EasyJDB.getInstance();
```

3.4 实(域)对象 PO 及泛对象 FO(Fan Object)

在 EasyDBO 中，关系表中的数据可以跟映射成两种类型的对象，一种是实对象，也即经常所说的域对象(DOM)或程序对象 PO。比如，一个 User 表与一个 User 类映射，一个 Message 表与 Message 类映射，这里的 User 及 Message 对象我们都称为域对象或程序对象 PO。

泛对象，也即类型为 EasyDBO 中的 DBObject 类型的对象。凡是不能直接映射到某一个域对象的，都可以在查询的时候，直接映射到 DBObject。DBOject 类是 EasyDBO 的一个核心类，用于代表关系数据表中的数据，其使用一个 Map 来存放查询到的数据。

在实际应用中，能直接映射成域对象的，我们都尽量使用域对象。若数据结构比较复杂、涉及多个相在关联的比较随意的查询时，我们就可以使用泛对象来轻松简单表示映射数据。

比如：select a.cid as cid,b.title as title,c.userName as userName from IDS a,Message b,User c inner join.....

3.5 使用接口实现简单映射

为了使 User 类能支持对象-关系映射，我们需要根据 User 类属性在相应的对象-关系映射中间件产品中作一些配置。用过 Hibernate 的都知道，在 hibernate 中需要配置 hibernate.cfg.xml 文件及与 User 类相应的 User.hbm.xml 文件来定义映射关系。

在 EasyDBO 中，可以通过简单的实现 EasyDBO 中的 com.easyjf.dbo.IObject 接口，达到最简单的对象-关系映射操作。接口映射建立在下面的假设：一个对象对应一个表，并且对象的属性于表字段名也一一对应。

IObject 接口的内容如下：

```
package com.easyjf.dbo;
public interface IObject {
String    getTableName();
String getKeyField();
String getKeyGenerator();
}
```

其中 getTableName()方法可以得到对象对应的关系表名称；getKeyField()方法返回关系中标识对象 ID 的主键字段名；getKeyGenerator()方法返回对象标识 ID 的生成处理器。

下面是实现了 IObject 接口的 User 类，User.jav 代码示例如下:

```
package com.easyjweb.business;

import java.util.Date;
import com.easyjf.dbo.IObject;
public class User implements IObject ,Serializable {
	private String cid;
	private String userName;
	private String password;
	private String email;
	private String tel;
	private Date birthday;
	private String intro;
	public String getTableName() {
		return "UserTable";
	}
	public String getKeyField() {
		return "cid";
	}
	public String getKeyGenerator() {
		return "com.easyjf.dbo.RandomIdGenerator";
	}
	public String getCid() {
		return cid;
	}
	public void setCid(String cid) {
		this.cid = cid;
	}
	..
	其它 getter 及 setter 方法
}
```

从上面的代码中，我们可以看出，我们 User 类在关系表中对应的表名为 User，主键字段是 cid，主键生成算法是 com.easyjf.dbo.RandomIdGenerator 类，生成的是一个 16 位的随机字符串。

使用接口映射方式可以实现最简单的、最常用的单表映射，在一些以数据表为主的系统或很多中小型的数据库应用中，由于表之间的关系不复杂，可以使用这种方式来实现映射，

不需要写配置文件，非常实用及方便。

由于只在接口中定义了表的名称、主键字段及主键生成器三项信息。因此无法进行诸如有选择性的缓迟加载，也无法定义表之间的关系等。因此，对于关系比较复杂的表，映射成对象的时候需要用定义 1 对 1、1 对 N、N 对 M 等多种关系，还有可能需要配置一些字段使用缓迟加载等。另外，接口定义要求属性的名称与数据表字段的名称一一对应，对于表字段名与对象属性名不相同的时候，这种方法就无法正确工作了。

3.6 使用配置文件配置映射关系

配置文件映射是指通过在配置文件中定义对象与关系表的映射关系。在配置文件中，除了定义对象对应的表、表之间的关系、对象属性与表字段的对应关系以外，还可以定义对象一些属性的使用缓迟加载，定义 1 对 1、1 对 N 等关系。这是 EasyDBO 为了满足比较复杂的对象－关系映射而提供的一种映射方式。

EasyDBO 的映射配置文件默认名称为 easyjf-dbo.xml，在前面的 Message 表的映射中，若改成配置文件映射，则域对象变成一个标识为 Serializable 的普通 JavaBean，不再需要实现 IObject 接口。其内容如下：

```
    public class Message implements Serializable {
    private String cid;
    private String title;
    private String content;
    private String inputUser;
    private Date inputTime;
    private Boolean publish;
    private Integer status;
    public String getCid() {
        return cid;
    }
    public void setCid(String cid) {
        this.cid = cid;
    }
...
    其它的 getter 及 setter 方法
}
```

在映射配置文件 easyjf-dbo.xml 中，通过下面的内容来定义对象关系映射。

```
<class name="com.easyjf.dbo.example.Message" table="Message"
schema="dbo" catalog="easyjf" lazy = "false">
        <id name="cid" type="string">
            <column name="cid" length="16" not-null="true" />
            <generator class="com.easyjf.dbo.IdGenerator" />
        </id>
        <property name="title" type="string">
            <column name="title" length="50" not-null="true" />
        </property>
        <property name="content" type="string">
            <column name="content"  />
```

```
        </property>
        <property name="status" type="integer">
            <column name="status" length=""  />
        </property>
        <property name="inputUser" type="string">
            <column name="inputUser" length="23" />
        </property>
        <property name="inputTime" type="date">
            <column name="inputTime" length="8" />
        </property>
        <property name="publish" type="bool">
            <column name="publish" length="1" />
        </property>
    </class>
```

在配置文件中，使用<class>标签来定义一个类，在<class>标签中，name 属性表示域对象 PO 完整的类名，属性 table 表示对应的主表；<id>标签定义对象的主键，<id>标签下的<generator>标签定义主键生成器；<property>标签定义对象的属性，其下的<column>定义属性对应的表字段等。

在当字段名称与对象属性名称无法一一对应，以及在需要配置比较复杂的关系时，使用该方式可以实现比较复杂、实用的对象－关系映射。

3.7 使用 Java5 注解来设置映射

由于配置文件的书写及管理比较复杂，EasyDBO 还提供了使用 Java5 注解的形式来定义对象关系映射。其可以实现跟使用配置文件完全相同的功能，满足复杂关系映射的配置。对于 Java5 以上的版本，我们建议使用该方式来定义映射。

下面是使用注解来配置映射的例子，上列中的 Message 类书写如下所示：

```java
import java.io.Serializable;
import java.util.Date;

import com.easyjf.dbo.annotation.*;
@Table(tableName="message")
public class Message implements Serializable{
@TableField(name="cid")
private String cid;
@TableField(name="title")
private String title1;
@TableField(name="content")
private String content1;
@TableField(name="inputUser")
private String inputUser;
@TableField(name="inputTime")
private Date inputTime;
@TableField(name="publish")
```

```
    private Boolean publish1;
    @TableField(name="status")
    private Integer status1;
    public String getCid() {
        return cid;
    }
    public void setCid(String cid) {
        this.cid = cid;
    }
    。。。
    其它的 getter 及 setter 方法
}
```

在上面的域对象 PO 中，@Table 标签用来定义表所对应的关系表。@TableField 标签用来定义对象属性所对应的表字段。另外还可以通过@OneToOne 标签、@ManyToOne 等标签来定义 1 对 1、1 对多的关系，我们将在后面详细讲述。

3.8 主键配置

在 EasyDBO 中，我们假设每一个表都有一个主键，这也是一种实践证明比较科学的关系表设计方式。有了主键，就可以通过主键值直接从持久层即数据库中读取对象。因此，需要在映射的时候定义对象的主键，同时定义对象主建值的生成器。

主键的定义通过 IObject 的 String getKeyField()方法，或者是配置文件中的<id>标签，或者是注解中的@Table 标签的 keyField 属性来指定。主键值生成器是指在第一次把对象保存到数据库中时，EasyDBO 使用的主键生成算法处理器，通过 IObject 的 String getKeyGenerator()方法，或者配置文件中的<generator>标签，或者是@Table 标签 keyGenerator 属性来指定主键值生成器。生成器是一个实现了 IIdGenerator 接口的对象。

IIdGenerator 接口的内容如下：

```
    public interface IIdGenerator {
    Object generator(Class classType);
    Object generator(Class classType, String column);
}
```

在 EasyDBO 中，内置了几种比较常用的主键值生成器。一种是用于支持自动增量的 NullIdGenerator，另外两种是用于生成 16 位随机字符串的主建值生成器 IdGenerator 及 RandomIdGenerator。用户可以根据实际的应用需要定义自己的主键生成器，我们建议在一套系统中使用统一的主键生成器。

3.9 缓存设置

在默认的情况下，EasyDBO 不使用缓存。有两种方法可以开取缓存功能！

方法一：

直接给 EasyJDB 的 innerCache 设置一个合法的 DBOCache，并把 enableCache 设置成 true，则在使用的过程中就会开取缓存功能。这种方法在基于 IOC 的配置中比较适用，也可以在自己的程序中手动创建 DBOCache 对象来设置缓存。代码如下：

```
com.easyjf.cache.ICache cache=new com.easyjf.cache.EasyCache();
```

```
com.easyjf.dbo.DboCache dcache=new com.easyjf.dbo.DboCache();
dcache.setCache(cache);
easyjdb.setInnerCache(dcache);
easyjdb.setEnableCache(true);
```

方法二：

若要开取缓存功能，可以在创建 EasyJDB **实例的时候，设置** EasyJDB **的** cacehName **为非** null **或**""**即可。若给** EasyJDB **对象设置了** cacheName**，则会自动尝试通过** easyjf-cache.xml **文件中加载指定配置的** cache**。也可以在配置文件** easyjf-dbo.xml **文件中配置中设置是否使用缓存，以及缓存的名称。**

仅仅让 EasyJDB **有一个缓存名称，或把** EasyJDB **的** enableCache **属性设置为** true **只是表示** EasyJDB **运行过程中，将尝试使用缓存，能否真正让缓存正确工作，还要取决于** Cache **详细参数的配置。**

Cache**的详细参数配置在另外一个名为**easyjf-cache.xml**的文件中，要在**EasyDBO**中使用缓存功能，必须确保该文件正确设置。**easyjf-cache.xml**文件的内容比较简单，大致内容如下：**

```
<?xml version="1.0" encoding="utf-8"?>
<easyjf-cache>
    <!-- storePolicy主要有LRU、LFU、FIFO三种,备注LRU缓存还有问题-->
    <cache name="EasyDBO" storePolicy="LFU" maxElements="50"
        expiredInterval="1000" type="com.easyjf.cache.EasyCache" />
</easyjf-cache>
```

在配置文件中，<cache>**标签用来定义详细的缓存内容，**name **表示缓存的名称，**storePolicy **属性定义缓存使用的策略，**maxElements **定义缓存中的最大元素个数，**expiredInterval **表示缓存超时的时间，**type **表示缓存类型。在** EasyDBO **中，type 默认是** EasyCache**。**

在 easyjf-dbo.xml **中配置缓存：**

```
<property name="easydbo.cache_name">EasyDBO</property>
<property name="easydbo.enable_cache">true</property>
```

在创建 EasyJDB **的过程中设置缓存：**

```
EasyJDB easyjdb=new EasyJDB(dataSoruce,"EasyDBO");
```

3.10 一对一映射

我们知道，在基于 OO **的程序设计中，对象与对象之间存在着继承、聚合等关联关系；而在关系表中，表与表之间也存在着一定的关联关系。因此，在** ORM **系统中，我们也希望反映这种对象与对象之间的关系。在** EasyDBO **中，一对一关系映射是通过**<one-to-one>**标签或者**@OneToOne **标签实现。**

下面看一个配置文件：

```
<class name="com.easyjf.dbo.example.OrderDetail"
table="OrderDetail" schema="dbo" catalog="easyjf" lazy = "false">
      <id name="cid" type="string">
          <column name="cid" length="16" not-null="true" />
```

```
            <generator class="com.easyjf.dbo.RandomIdGenerator" />
        </id>
         <one-to-one name="order"
type="com.easyjf.dbo.example.Order" column="orderId" key="cid"
lazy="true">
         </one-to-one>
         <one-to-one name="product"
type="com.easyjf.dbo.example.Product" column="productId"
key="cid">
        </one-to-one>
...
        </class>
```

对于<one-to-one>标签：

name属性：表示与类本身对应的类在声明时使用的属性名，比如order就是在OrderDetail对象中定义的private Order order;

type属性：表示对应类的类型（Class）

column属性：表示在类本身的数据表中标识对应类主键的数据表字段，比如column="orderId"中，order项就是在OrderDetail表中使用orderId字段来对应order表的cid，一般这个是一个外键对应。

lazy属性：表示这个对象要延迟加载，关于延迟加载请看相关小节介绍。

在上面的配置中，定单详细信息中包含了两个一对一关系。即从 OrderDetail 的角度来看，一个定单详细信息有一个 Order 及一个 Product 以之相对应。因此，我们可以在映射配制中使用<one-to-one>标签，来配置 OrderDetail 的 order 属性。

而这个时候对于 OrderDetail 对象来说，在他的 bean 的定义里面，就会是这样：

```
public class OrderDetail {
    private String cid;

    private Order order;

    private Product product;

    private Integer num;

    private java.math.BigDecimal price;
    //…
}
```

可以直接使用

```
    OrderDetail detail=(OrderDetail)db.get(OrderDetail.class, id);
    String orderTitle=detail.getOrder().getTitle();
```

来访问order对象了。

这里只是单向一对一的情况，如果要想order和orderDetail对应，就是使用

```
Order order=(Order)db.get(Order.class, id);
OrderDetail detail=order.getOrderDetail();
```

来访问detail对象，实现双向一对一，那么我们还要在Order一端配置对OrderDetail的映射：

```
<one-to-one name="orderDetail" type="com.easyjf.dbo.example.OrderDetail" column="orderDetailId" key="cid" lazy="true">
```

当然，根据我们的经验，一个order不会只和一个orderDetail对应，Order与OrderDetail应该是一对多关系，这就要使用下面小节介绍的一对多的情况。

当然，若不喜欢使用配置文件，则可以使用 Java 注解来配置这种关系，如下所示：

```
@Table(tableName="OrderDetail")
public class OrderDetail {
@TableField(name="cid")
private String cid;
@OneToOne(column="orderId",tableName="",type=Order.class)
private Order order;
@OneToOne(column="productId",tableName="",type=Product.class)
private Product product;
@TableField(name="num")
private Integer num;
@TableField(name="price")
private java.math.BigDecimal price;
public String getCid() {
    return cid;
}
public void setCid(String cid) {
    this.cid = cid;
}
...
}
```

3.11 一对多映射

一对多是指一个类的一个实例与另外一个类多个实例存在着对应关系。比如上面所说的对于一个订单 Order，有多个订单详细列表 OrderDetail。可以通过配置文件<many-to-one>标签或使用 Java 注解@ManyToOne 来配置这种关系，如下面是 Order 对象与 OrderDetail 的一对多映射的配置文件：

```
    <class name="com.easyjf.dbo.example.Order" table="OrderInfo" schema="dbo" catalog="easyjf" lazy = "false">
        <id name="cid" type="string">
            <column name="cid" length="16" not-null="true" />
            <generator class="com.easyjf.dbo.RandomIdGenerator" />
```

```
        </id>
         <property name="title" type="string">
            <column name="title" length="50" not-null="true" />
        </property>
       <many-to-one name="children" fieldType="java.util.List"
type="com.easyjf.dbo.example.OrderDetail" column="orderId"
key="cid">
         </many-to-one>
        </class>
```

对于<many-to-one>**标签：**

name **属性：一对多中，一一方的类中定义的来保存多一方类的属性名；**

fieldType **属性：一对多中，一一方类用来保存多一方类的集合类型。比如这里的** ArrayList **或者** List、Set **等；**

type **属性：一对多中，对应的多一方的类的类型（**Class**）；**

column **属性：一对多中，多一方用来标示一一方的数据表的字段名，比如这里就是在** orderDetail **表中有一个** orderId **来对应一个** Order **对象；**

key **属性：一对多中，多一方的主键名，这里对应的是** orderDetail **表中的** cid **字段。**

在这样的配置下，Order **类就会是这样：**

```
public class Order {
    private String cid;
    private String title;
    private java.math.BigDecimal money;
    private List children;
```

同样，我们就可以使用：

```
    Order order=(Order)db.get(Order.class, id);
    List allItem=order.getChildren();
    for(...){
        //遍历所有的OrderDetail操作；
    }
    //...
```

若不喜欢写配置文件的朋友，也可以使用下面 Java **注解标识的方式：**

```
@Table(tableName="orderInfo")
public class Order {
@TableField(name="cid")
private String cid;
@TableField(name="title")
private String title;
@TableField(name="money")
private java.math.BigDecimal money;
@ManyToOne(column="orderId",tableName="",type=OrderDetail.class)
private List children=new java.util.ArrayList();
```

```
public String getCid() {
    return cid;
}
public void setCid(String cid) {
    this.cid = cid;
}
……
}
```

测试代码：

```
    EasyJDB db=EasyJDB.getInstance();
    db.setAutoCommit(false);
    Order order=new Order();
    order.setTitle("测试");
    order.setCid("1111");
    java.util.List details=new java.util.ArrayList();
    for(int i=0;i<10;i++){
    OrderDetail detail=new OrderDetail();
    detail.setOrder(order);
    detail.setNum(15);
    detail.setPrice(new java.math.BigDecimal(16));
    details.add(detail);
    }
    order.setChildren(details);
    db.add(order);
    db.commit();
```

在上面的代码中，我们在使用 EasyJDB 的 add 方法保存 order 对象的时候，也将会把其 children 属性中的 OrderDetail 对象保存到数据库对应的表中。

在一对多的关系映射中，默认情况下 EasyDBO 使用了延迟加载功能，若要关掉延迟加载，可以把属性的 lazy 设置成 false 即可。

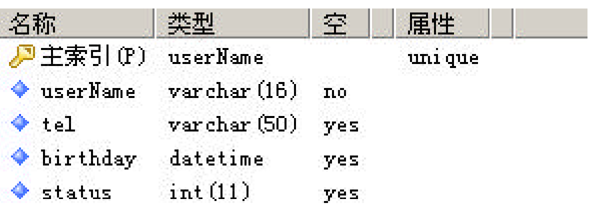
3.12 多对多映射

在现实对象关系中，还存在多对多关系，也即一个类实例与另一个类的多个实例相互彼此对应。在实际应用中，这种多对多关系很多时候都可以转换成一对多关系。因此，若能转换成一对多的时候，我们尽量转换成一对多来处理。实在不行，可以使用 EasyDBO 的多对多映射功能，来表示这种关系。在 EasyDBO 的映射配置中，使用配置文件中的 <many-to-many>标签或 Java 注解@ManyToMany 标签来配置这种多对多的关系。在多对多关系中，经常要引入一个中间表。下面看一个多对多的实例：

产品的 Product 表结构

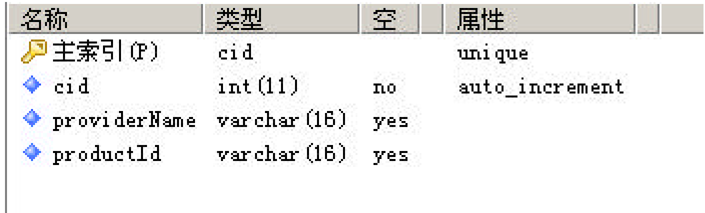
名称	类型	空	属性
主索引(P)	cid		unique
cid	varchar(16)	no	
title	varchar(50)	yes	
intro	varchar(200)	yes	
price	decimal(10,2)	yes	
status	int(11)	yes	

供应商的 Provider 表结构



名称	类型	空	属性
主索引(P)	userName		unique
userName	varchar(16)	no	
tel	varchar(50)	yes	
birthday	datetime	yes	
status	int(11)	yes	

存放多对多关系的 ProvidrProduct 表结构



名称	类型	空	属性
主索引(P)	cid		unique
cid	int(11)	no	auto_increment
providerName	varchar(16)	yes	
productId	varchar(16)	yes	

在上面的所示的关系中，一个 Provider 可以供应多种产品，同一个产品 Product 也可以由多个供应商供应。因此，产品 Product 与供应商 Provider 之间是多对多的关系。在 EasyDBO 中，我们通过下面配置文件来表示这种关系！

```
<class name="com.easyjf.dbo.example.Product" table="product"
schema="dbo" catalog="easyjf" lazy = "false">
       <id name="cid" type="string">
          <column name="cid" length="16" not-null="true" />
          <generator class="com.easyjf.dbo.IdGenerator" />
       </id>
       <many-to-many name="providers" fieldType="java.util.List"
type="com.easyjf.dbo.example.Provider" tableName="providerproduct"
column="productId" key="cid" tagColumn="providerName"
tagKey="userName">
        </many-to-many>
       ...
       </class>
    <class name="com.easyjf.dbo.example.Provider" table="Provider"
schema="dbo" catalog="easyjf" lazy = "false">
      <many-to-many name="prdoucts" fieldType="java.util.List"
type="com.easyjf.dbo.example.Product" tableName="providerproduct"
column="providerName" key="userName" tagColumn="productId"
tagKey="cid">
       </many-to-many>
```

```
    ...
    </class>
```

在上面的配置文件中：

name—表示存在多对多关系的属性名；

filedType—表示属性的类型，即集合或数组等微容器类型；

type—表示关联对象的实际类型，即微容器中的元素类型；

tableName—表示中间表名；

columan—表示关联表中跟本对象对应的字段名称；

key—表示关联表中代表本对象的属性名称；

tagColumn—表示关联目标对象的字段名称；

tagKey—表示关系目标对象的属性名称。

在以上属性中，除了 filedType 可以缺省以外，其它的属性都是必须设置的。

跟一对一、一对多关系一样，我们也可以使用 Java5 注解来表示配置这种第，只要使用@ManyToMany 标签即可，如下所示：

Product.java

```
import com.easyjf.dbo.annotation.ManyToMany;
import com.easyjf.dbo.annotation.Table;
import com.easyjf.dbo.annotation.TableField;
@Table(tableName="Product")
public class Product {
@TableField(name="cid")
private String cid;
@TableField(name="title")
private String title;
@TableField(name="intro")
private String intro;
@TableField(name="price")
private java.math.BigDecimal price;
@ManyToMany(fieldType=java.util.List.class,type=Provider.class,co
lumn="productId",tableName="providerproduct",key="cid",tagColumn=
"providerName",tagKey="userName")
private java.util.List providers;
@TableField(name="status")
private Integer status;
public String getCid() {
    return cid;
}
…
}
```

Provider.java

```
package com.easyjf.dbo.example;
import com.easyjf.dbo.annotation.ManyToMany;
import com.easyjf.dbo.annotation.Table;
```

```
import com.easyjf.dbo.annotation.TableField;
@Table(tableName="Provider")
public class Provider {
    @TableField(name="userName")
    private String userName;
    @TableField(name="tel")
    private String tel;
    @TableField(name="birthday")
    private java.util.Date birthday;
    @TableField(name="status")
    private Integer status;
    @ManyToMany(fieldType=java.util.List.class,type=Product.class,
column="providerName",tableName="providerproduct",key="userName",
tagColumn="productId",tagKey="cid")
    private java.util.List products;
    public java.util.List getProducts() {
        return products;
    }
    public void setProducts(java.util.List products) {
        this.products = products;
    }
...
}
```

测试代码：

```
    EasyJDB db=EasyJDB.getInstance();
    db.setAutoCommit(false);
    Provider p=new Provider();
    p.setUserName("EasyJF");
    java.util.List products=new java.util.ArrayList();
    for(int i=0;i<5;i++)
    {
        Product pro=new Product();
        pro.setTitle("产品"+i);
        pro.setIntro("好产品啊"+i);
        products.add(pro);
    }
    p.setProducts(products);
    db.add(p);
    db.commit();
```

则上面的代码将 Provider 表中插入一条供应商信息，在 Product 表中插入五条产品信息，在中间表 ProviderProduct 中插入五条关联数据信息。

3.13 延迟加载

在基于 ORM 系统的软件开发中，我们并不需要每时每刻都加载持久对象的所有属性，有一些属性的所占的内存比较大，有一时属性加载时间也比较长，而我们又并非每次都要用到对象的全部属性。因此，为了让用户能有选择性的加载持久化对象属性，EasyDBO 提供了一个延迟加载功能，通过这个功能，把一些占用内存比较大或加载时间长的不常用的对象属性设置成延迟加载，这样就可以实现持久层对象加载的优化处理，提高了持久化对象的加载速度及效率。

默认情况下，除了一对多，多对多的属性以外，一个持久层对象其它所有的属性都是即时加载的。若要把一个对象属性设置成缓迟加载，需要在持久层对象的映射配置文件或者是注解中进行定义。

在配置文件中定义缓迟加载，直接把属性映射中的 lazy 属性设置成 true 即可，如下面的示例：

```
<class name="com.easyjf.dbo.example.OrderDetail"
table="OrderDetail" schema="dbo" catalog="easyjf">
        <id name="cid" type="string">
            <column name="cid" length="16" not-null="true" />
            <generator class="com.easyjf.dbo.RandomIdGenerator" />
        </id>
         <property name="num" type="integer" >
            <column name="num" length=""  />
        </property>
         <property name="price" type="java.math.BigDecimal">
            <column name="price" length=""  />
        </property>
         <one-to-one name="order"
type="com.easyjf.dbo.example.Order" column="orderId" key="cid"
lazy="true">
         </one-to-one>
        <one-to-one name="product"
type="com.easyjf.dbo.example.Product" column="productId" key="cid">
        </one-to-one>
        </class>
```

在上面的示例中，属性 order 就是使用了缓迟加载。即当使用 EasyJDB 的 get(Class cls, Object id)来从数据库中加载一个 OrderDetail 对象的时候，不会立即加载相应的 oder 属性的值，只有当要使用这个持久对象的 order 属性的时候，才会从数据库中加载。

另外，若不喜欢使用烦琐的配置文件，也可以通过在注解中定义缓迟加载。如下所示：

```
@Table(tableName="OrderDetail")
public class OrderDetail {
@TableField(name="cid")
private String cid;
@OneToOne(column="orderId",tableName="",type=Order.class,lazy=tru
e)
```

```
private Order order;
@OneToOne(column="productId",tableName="",type=Product.class)
private Product product;
@TableField(name="num")
private Integer num;
@TableField(name="price")
private java.math.BigDecimal price;
....
```

在上面的 OrderDetail 类定义中，我们使用@TableField 标签来配制对象映射关系，其中属性 order 使用了缓迟加载。

3.14 事务处理

默认情况下，EasyDBO 是不支持事务的（自动提交），要使用事务，需要通过把 EasyJDB 的自动提交标志设置成为 false，然后再用 commit()来提交数据，最后使用 close 方法来释放数据源，这个原理跟 JDBC 的默认操作是一致的。

示例如下：

```
EasyJDB db=EasyJDB.getInstance();
db.setAutoCommit(false);//开取事务标志
Message m=new Message();
Message m2=new Message();
m.setTitle("标题");
m2.setTitle("标题 2");
m.setInputTime(new java.util.Date());
db.add(m);
db.add(m2);
db.commit();      //提交
db.close();//释放数据源，结束事务
```

从代码中可以看到，add(m),add(m2)是在一个事务中，其中任何一条 add 方法出错，则都会造成数据加整个提交失败。

3.15 使用存储过程

在 EasyDBO 中，可以直接通过存储过程名称及参数调用存储过程。EasyDBO 支持两种类型的存储过程，一种是直接对数据库进行相关操作的存储过程，如数据更新、数据处理、数据删除等,这种存储过程不返回任何结果;另外一种是通过存储过程返回查询结果列表的，返回的结果以泛对象(FO)的形式存放于 List 中。

在 EasyJDB 核心映射处理类中，我们有以下几个方法用于支持存储过程，用户直接调用这些方法即可。这几个方面分别如下：

java.util.List callQuery(java.lang.String procedureName)

调用指定的存储过程，并返回一个类型为 DBObject 的结果列表。

java.util.List callQuery(java.lang.String procedureName, java.lang.Object[] parameter)

根据所给的参数，调用指定的存储过程，并返回一个类型为 DBObject 的结果列表。

void callUpdate(java.lang.String procedureName)

执行指定名称的存储过程。

void callUpdate(java.lang.String procedureName, java.lang.Object[] parameter)
根据所给的参数，执行指定名称的存储过程。

下面是一个简单列子：

```
EasyJDB easyjdb=new EasyJDB(dataSource);
List list=easyjdb.callQuery("getPM",new Object[]{"四川省"});
for(int i=0;i<list.size;i++)
{
DBObject obj=(DBObject)list.get(i);
System.out.println(obj.get("title"));
System.out.println(obj.get("num"));
}
```

其中 getPM 是一个存储过程名，这个存储过程需要给其提供一个地区作为参数。

四、EasyJDB 中常用方法介绍

4.1 对象添、删、改

boolean add(java.lang.Object obj)
把对象 obj 保存到持久层数据库中；

boolean update(java.lang.Object obj)
把对象 obj 的内容更新的持久层数据库中；

boolean saveOrUpdate(java.lang.Object obj)
把对象 obj 添加或更新到持久层数据库中；

boolean del(java.lang.Object obj)
把对象从持久层对象中删除。

在上面的几个方法中，参数 obj 可以是一个域对象 DOM(或 PO)，若是一个域名对象，则 EasyDBO 会把其相关属性插入或修改到对应表的相应列中，若对象还包括一对一、一对多或对多多等属性，则在添加的时候还会在相应的表中插入或修改数据记录；同理，若是执行删除操作，若对象包含多对多或者一对多属性，则还会删除其下属的对象。

参数 obj 也可以是一个泛对象(FO)，即类型为 DBObject 的对象。此时会根据泛对象中定义的表以及列名称，生成一个对应操作的 sql 语句，然后执行该 sql 语句进行数据操作。

上面四个方法，是我们在使用 EasyDBO 进行数据操作过程中最常用到的方法。saveOrUpdate 方法是一个比较灵活的方法，其首先要查找一次系统中是否存在所要操作的对象，若存在则执行 update 操作，若不存在则执行 add 操作，在实际的应用需要灵活使用。

除了上面四个方法可以达到数据库内容的改变以外，还可以通过存储过程或者是直接使用 EasyJDB 的 execute 方法执行数据操作 sql 语句达到数据库内容的更改。

4.2 从持久层读取单个域对象 DOM(PO)

java.lang.Object get(java.lang.Class cls, java.lang.Object id)
根据对象类型 cls 及主键值从特久设备中读读取对象，若找不到对象则返回 null.。这里要求 id 必须为一个对象类型。

示例：

Message message=(Message)db.get(Message.class,"1111");

java.lang.Object read(java.lang.Class cls, java.lang.String scope)

根据查询条件从数据表中读出一个对象，若符合条件的记录有多个，则该方法只返回第一个对象，其它的将被忽略。

示例：

Message message=(Message)db.get(Message.class,"inputUser='**大峡**'");

这种查询方式主要采用拼凑 sql **语句的方式，由于可能存在** sql **注入攻击，因此我们不建议使用。**

java.lang.Object read(java.lang.Class cls, java.lang.String scope, java.util.Collection params)

根据查询条件及具体的参数从数据表中读出一个对象，若符合条件的记录有多个，则该方法只返回第一个对象，其它的将被忽略。此处的参数存放于集合类型中，取个数及值与查询条件 scope **中的问号**"?"**号对应。**

示例：

java.util.Collection paras=new java.util.ArrayList();
paras.add("**大峡**");
Message message=(Message)db.get(Message.class,"inputUser=?",paras);

在实际应用中，这种方法可以避免 sql **注入漏洞攻击，另外还可以灵活支持** jdbc **的各种数据类型，因此，我们推荐使用这种方法使用查询。**

4.3 从持久层读取多个域对象 DOM(PO)

java.util.List query(java.lang.Class cls, java.lang.String scope)

执行条件查询，返回类型为 cls **的对象列表**!

示例：

java.util.List list=db.query(Message.class,"inputUser='**大峡**'");

这个查询将从持久层数据库中查出所有输入人 inputUser **值为"大峡"的** Message **对象。同理，在实际应用中，这种方法可以避免** sql **注入漏洞攻击，另外还可以灵活支持** jdbc **的各种数据类型，因此，我们推荐使用这种方法使用查询。**

java.util.List query(java.lang.Class cls, java.lang.String scope, java.util.Collection params)

根据查询条件及参数执行查询，返回类型为 cls **的对象列表。**

示例：

java.util.Collection paras=new java.util.ArrayList();
paras.add("**大峡**");
java.util.List list=db.query(Message.class,"inputUser=?",paras);

这个查询将从持久层数据库中查出所有输入人 inputUser **值为"大峡"的所有** Message **对象。**

java.util.List query(java.lang.Class cls, java.lang.String scope, java.util.Collection params, int begin, int max)

根据用户自定义的 sql **语句执行查询操作，返回从** begin **开始**,max **条记录，类型为** cls **的对象列表。在我们只需要加载查询结果中的几个对象时，使用该方法将提高处理速度，提升系统性能。**

示例：

```
java.util.Collection paras=new java.util.ArrayList();
 paras.add("大峡");
 java.util.List list=db.query(Message.class,"inputUser=?",paras,100,10);
```

查询输入人 inputUser **值为“大峡”对象，从第** 100 **条符合的数据开始，最多取** 10 **个对象。**

java.util.List query(java.lang.Class cls, java.lang.String scope, java.util.Collection params, int begin, int max, boolean cache)

根据用户自定义的 sql **语句执行查询操作，返回从** begin **开始**,max **条记录，类型为** cls **的对象列表，并把查询结果存入缓存中。**

这个查询的功能跟上面一样，只是这个查询将会把查询结果存入缓存中，若下次再遇到跟这个一样的查询，则直接使用缓存中的查询结果。

示例：

```
java.util.Collection paras=new java.util.ArrayList();
 paras.add("大峡");
 java.util.List list=db.query(Message.class,"inputUser=?",paras,100,10,true);
```

4.4 从持久层读取多个泛对象 FO(Fan Object)

java.util.List query(java.lang.String sql)

执行用户自定义 sql **语句，直接返回泛数据表** DBObject **对象列表。**

java.util.List query(java.lang.String sql, java.util.Collection params)

根据用户自定义的 sql **语句及查询参数执行查询，返回泛数据表** DBObject **对象列表**

java.util.List query(java.lang.String sql, java.util.Collection params, int begin, int max)

根据用户自定义的 sql **语句执行查询操作，返回从** begin **开始**,max **条记录的泛数据表** DBObject **对象列表**

java.util.List query(java.lang.String sql, java.util.Collection params, int begin, int max, boolean cache)

根据用户自定义的 sql **语句执行查询操作，返回从** begin **开始**,max **条记录的泛数据表** DBObject **对象列表。**

在读取多个泛对象的查询中，需要我们提供完整的 SQL **语句。如** select * from Message where inputUser='**大峡**'**。与读取多个域对象不同的是，由于没有指定返回结果集的对象类型，因此，查询出来的结果集合中是类型为** DBObject **泛对象。其它的参数与前面查询域对象的参数意义及用法相同。**

示例：

```
java.util.Collection paras=new java.util.ArrayList();
 paras.add("大峡");
 java.util.List list=db.query("select * from Message where inputUser=?",paras,0,10,true);
for(int i=0;i<list.size();i++)
{
DBObject obj=(DBObject)list.get(i);
System.out.println(obj.get("title"));
```

}

4.5 从持久层返回唯一对象

java.lang.Object uniqueResult(DBTable table, DBField field, java.io.Serializable value)

根据主键加载一个表中的某一个列；

这个方法适合用于在知道表配置关系 DBTable，DBField 的情况下。

java.lang.Object uniqueResult(java.lang.String sql)

执行一条 sql 语句，返回唯一的对象，不缓存查询结果。

java.lang.Object uniqueResult(java.lang.String sql, boolean cache)

执行一条 sql 语句，返回唯一的对象，根据 cache 的值决定是否缓存结果。

java.lang.Object uniqueResult(java.lang.String sql, java.util.Collection params)

根据自定义 sql 语句及查询参数执行查询，返回唯一的对象，不缓存查询结果。

java.lang.Object uniqueResult(java.lang.String sql, java.util.Collection params, boolean cache)

根据自定义 sql 语句及查询参数执行查询，返回唯一的对象，根据 cache 的值决定是否缓存结果。

uniqueResult 查询主要是用来查询只返回一个对象的 sql 查询，这个对象是一般的 Java 类型如 Number、String，或者是表字段对应的类型，而不是域对象 DOM(PO)或泛对象 FO。uniqueResult 中的 sql 是一条完整的 sql 语句，适用于查询数据记录总数、数据求合、统计结果、查询数据表中某一行的某一个字段值的情况。

示例：

```
Number num=(Number)db.uniqueResult("select count(*) from Message");
```

返回 Message 表中的记录数；

```
java.util.Collection paras=new java.util.ArrayList();
 paras.add("1");
String title=(String)db..uniqueResult("select title from Message where cid=?",paras);
```

返回 Message 表中 cid 为"1"的行的 title 列的值。

4.6 执行自定义 sql 语句

int execute(java.lang.String sql)

执行指定的 sql 语句，返回受影响的记录数。

int execute(java.lang.String sql, java.util.Collection params)

根据参数 params 中的值，执行指定的 sql 语句，返回受影响的记录数。

在实际应用中，我们尽量避免直接使用硬编码的 sql 语句。然而，在一些特殊情况下，我们出于性能或其它方面的考虑，需要直接传于自己的 sql 语句来操作数据库，则可以使用 EasyJDB 提供的 execute 方法，执行具体的 sql 语句操作，可以避免我们写烦琐的数据源获取、定义 Statement、释放数据源等 JDBC 底层操作，大大简化了编程。

示例：

```
db.execute("update Message set status=2 where status<1");
```

4.7 存储过程

java.util.List callQuery(java.lang.String procedureName)

java.util.List callQuery(java.lang.String procedureName, java.lang.Object[] parameter)

void callUpdate(java.lang.String procedureName)

void callUpdate(java.lang.String procedureName, java.lang.Object[] parameter

上面几个方法是通过 EasyDBO 来调用存储过程的接口。详情请参考第三章第 15 节《关于存储过程的使用》部分内容。

五、EasyDBO 以其它框架的集成运用

5.1 EasyDBO 与 EasyJWeb 集成

EasyDBO 是 EasyJWeb 的一个基础框架，EasyJWeb 之所以能实现快速开发，很大一部分因素是使用了 EasyDBO。可以这么说，EasyJWeb 的快速数据库应用开发功能，全是基于 EasyDBO，另外 EasyJWeb 中的代码生成等也是依赖于 EasyDBO。因此，EasyDBO 与 EasyJWeb 可以说是天然集成的。当然，EasyJWeb 也可以选择其它的 ORM 系统框架。

在 EasyJWeb 的 CRUDAction 中，每一个 Action 都有一个 DAO 操作对象 IDAO，用于充当综合数据访问层。IDAO 接口的内容如下：

```
public interface IDAO {
    boolean save(Object obj);
    boolean update(Object obj);
    boolean del(Object obj);
    Object get(Class clz, Serializable id);
    Object getBy(Class clz, String fieldName, Serializable value);
    List query(Class clz, String scope);
    List query(Class clz, String scope, Collection paras);
    List query(Class clz, String scope, Collection paras, int begin,
int max);
    Object uniqueResult(String sql);
    Object uniqueResult(String sql,Collection paras);
    int execute(String sql);
    int execute(String sql,Collection paras); // 执行任意SQL语句
}
```

在 EasyJWeb 中有一个基于 EasyDBO 的 IDAO 实现，类名为 EasyDBODAO,其内容大致如下：

```
public class EasyDBODAO implements IDAO {
    private static final EasyDBODAO singleton = new EasyDBODAO();
    private EasyJDB db;
    public EasyDBODAO() {
    }
    public EasyDBODAO(EasyJDB db) {
        this.db = db;
    }
    public EasyJDB getDb() {
        return db;
    }
    public void setDb(EasyJDB db) {
        this.db = db;
```

```
    }
    public boolean save(Object obj) {
        return db.add(obj);
    }
    public Object get(Class clz, Serializable id) {
        return db.get(clz, (Object) id);
    }
...
}
```

在 AbstractCrudAction **中，有一个自动加载** IDAO **对象的方法** autoLoadDAO**，其中有一个自动把** EasyDBO **作为数据层处理的代码，如下：**

```
    protected IDAO autoLoadDAO(Module module) {
        return EasyDBODAO.getInstance();
    }
```

清楚了这个结构及关系，我们就不难理解在 EasyJWeb Tools **自代码生成工具中，生成的一个数据表添删改查、及分页显示的代码了。下面是一个** EasyJWeb **示例，由** EasyJWeb Tools **代码生成工具生成的具有添删改查及分页的** Action **代码。**

```
    /**
 * 使用添删改查action
 * @author easyjwebtools
 *
 */
public class MessageAction extends CrudAction {
    /**
     * 查询及分页，通过实现AbstractCrudAction中的抽象方法实现
     */
    public IPageList doQuery(WebForm form, int currentPage, int
pageSize) {
        String scope = "1=1";
        Collection paras = new ArrayList();
        String orderType =
CommUtil.null2String(form.get("orderType"));
        String orderField =
CommUtil.null2String(form.get("orderField"));
        if (!"".equals(orderField)) {
            scope += " order by " + orderField;
            if (!"".equals(orderType))
                scope += " " + orderType;
        }
        DbPageList pList = new DbPageList(Message.class, scope,
paras);
        pList.doList(currentPage, pageSize);
```

```
        return pList;
    }

    /**
     * 这个方法负责根据Form中的数据转换成java对象。
     *
     */
    public Object form2Obj(WebForm form) {
        String cid = (String) form.get("cid");
        Message obj = null;
        if (cid != null && (!cid.equals("")))
            obj =(Message)this.getDao().get(Message.class,cid);
        if (obj == null)
            obj = new Message();
        return form.toPo(obj);
    }
```

5.2 EasyDBO 与 Spring 集成

在上面的 EasyDBO **的简单介绍中，我们会发现，**EasyDBO **的构造和配置完全符合构造方法注入或者** Setter **方法注入的使用，这使得它能和** spring **非常容易结合使用。下面给一个简单的** spring **下使用** EasyDBO **的配置实例。**

首先，我们配置一个 EasyDBO bean。

先配置一个 DataSource**：**

```
<bean id="dataSource"
        class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
        <property name="driverClassName">
            <value> org.gjt.mm.mysql.Driver </value>
        </property>
        <property name="url">
            <value> jdbc:mysql://localhost:3306/easydbotest </value>
        </property>
        <property name="username">
            <value>root</value>
        </property>
        <property name="password">
            <value>stef</value>
        </property>
    </bean>
```

数据源配好了，我们来配置我们需要的 EasyDBO **核心引擎** EasyJDB bean。**如果只是配置最简单的** EasyJDB，**只需这样：**

```
<bean id="easydbo" class="com.easyjf.dbo.EasyJDB" singleton="true">
    <property name="dataSource" ref="dataSource"></property>
</bean>
```

因为一个数据库对应一个 EasyJDB，这里使用 singleton 就可以了。现在，我们的 EasyDBO 已经是可用的了，来做个测试，这里我们就使用了前面的 Message 那个例子：

Message 类不变，编写一个简单的业务接口如下：

```
public interface IMessageService {
boolean add(Message message);
boolean del(Message message);
boolean update(Message message);
List getAllMessages();
}
```

然后就是这个简单业务接口的实现：

```
public class MessageService implements IMessageService {
    private EasyJDB db;
    public EasyJDB getDb() {
        return db;
    }
    public void setDb(EasyJDB db) {
        this.db = db;
    }

    public boolean add(Message message){
        return this.getDb().add(message);
    }

    public boolean del(Message message){
        return this.getDb().del(message);
    }

    public boolean update(Message message){
        return this.getDb().update(message);
    }

    public List getAllMessages(){
        return this.getDb().query(Message.class, "1=1");
    }
    }
```

接下来就是测试类了：

```
public class MessageTest {
    private IMessageService messageService;

    /**
     * @return the messageService
     */
    public IMessageService getMessageService() {
        return messageService;
```

```
    }

    /**
     * @param messageService the messageService to set
     */
    public void setMessageService(IMessageService messageService) {
        this.messageService = messageService;
    }

    public void test(){
        Message m=new Message();
        m.setTitle("留言标题");
        m.setContent("留言内容");
        m.setInputTime(new java.util.Date());
        m.setInputUser("easyjf");
        m.setStatus(new Integer(0));
        if(this.getMessageService().add(m)){
            System.out.println("成功写入数据??");
        }
        java.util.List list=this.getMessageService().getAllMessages();
        Message m2=(Message)list.get(0);
        System.out.println(m2.getTitle());
        System.out.println(m2.getContent());
        m2.setTitle("新的标题");
        if(this.getMessageService().update(m2)){
            System.out.println("成功修改数据");
        }
    }

    public static void main(String[] args){
        ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
        MessageTest test=(MessageTest)context.getBean("messageTest");
        test.test();
    }
    }
```

把以上几个类都在 applicationContext.xml 文件里面配置好：

```
<bean name="messageService" class="impl.MessageService">
        <property name="db" ref="easydbo"></property>
    </bean>

    <bean name="messageTest" class="test.MessageTest">
        <property name="messageService"
ref="messageService"></property>
```

```
        </bean>
```

好了，现在可以运行下这个简单的应用了：结果如下(去掉了 spring 的众多 debug 信息)：
成功写入数据
留言标题
留言内容
成功修改数据

从上面这个简单的例子可以看到，EasyDBO 和 spring 可以非常容易集成在一起，就像使用 hibernate，ibaits 类似。下面来看一个使用了 cache 的配置：

```
<bean id="easydbo" class="com.easyjf.dbo.EasyJDB"
        singleton="true">
        <property name="dataSource" ref="dataSource"></property>
        <property name="configFiles">
            <list>
                <value>/mappings.xml</value>
            </list>
        </property>
        <property name="showSql" value="true"></property>
        <property name="enableCache" value="true"></property>
        <property name="innerCache" ref="dcache"></property>
    </bean>

    <bean name="dcache" class="com.easyjf.dbo.DboCache">
        <property name="cache" ref="easycache"></property>
    </bean>

    <bean name="easycache" class="com.easyjf.cache.EasyCache" />
```

在上面的配置片断中，我们把 Message 的 mapping 信息放在/mappings.xml 文件中，所以这时的 Message 对象只是一个单纯的 javabean 了。另外，我们使用 setter 注入了一个 cache，通过使用 showsql=true，和在代码中的重复查询操作：

```
    for(int i=0;i<5;i++){
        java.util.List list=this.getMessageService().getAllMessages();
        Message m2=(Message)list.get(0);
        System.out.println(m2.getTitle());
        System.out.println(m2.getContent());
    }
```

我们可以看到 debug 信息：

```
[main] INFO  com.easyjf.cache.store.MemoryStore  - defaultCache:
defaultMemoryStore中找到OBJECT:domain.Message:1
1359 [main] INFO  com.easyjf.cache.store.MemoryStore  - defaultCache:
defaultMemoryStore中找到OBJECT:domain.Message:2
```

Cache 已经成功地在运行了。

5.3 在 Struts 中使用 EasyDBO

一般来说，现在的struts都是和spring集成使用的，这种情况下，就使用上面的那个演示即可，还有一种方法就是门面模式，这种情况下，EasyDBO的使用同前面讲解得EasyDBO的普通使用方法。如果应用与数据库交道确实很简单，那么在这种情况下可以把EasyDBO的一个实例在一个抽象的公共类中提供，而要使用数据库的action都继承这个类就可以了。还有一种使用方法就是类似于hibernate为struts提供的plugin一样，可以在一个plugin中实例化一个EasyDBO的实例，并放在applicationContext或者session里面供action使用。

下面给一个例子，演示把 EasyDBO 的一个实例在一个抽象的公共类中提供，而要使用数据库的 action 都继承这个类的用法。例子仍然使用 message。首先创建一个 BaseDaoAction：

```
public class BaseDaoAction extends Action {
    private EasyJDB easyjdb;
    protected EasyJDB getEasyjdb() {
        if(easyjdb==null){
            easyjdb=initJdb();
        }
        return easyjdb;
    }

    private EasyJDB initJdb(){
        BasicDataSource datasource = new BasicDataSource();
        datasource.setDriverClassName("org.gjt.mm.mysql.Driver");
        datasource.setUrl("jdbc:mysql://127.0.0.1:3306/easydbotest");
        datasource.setUsername("root");
        datasource.setPassword("mysql");
        com.easyjf.dbo.EasyJDB db=new
com.easyjf.dbo.EasyJDB(datasource);
        return db;
    }
    }
```

然后创建一个 ActionForm:

MessageForm:

```
  public class MessageForm extends ActionForm {
     private String title;
     private String content;
    public String getContent() {
        return content;
    }
    //...
```

最后是用来进行持久化处理MessageAction:

```
public class MessageAction extends BaseDaoAction {
    public ActionForward execute(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
```

```
        HttpServletResponse response) throws Exception {
    MessageForm messageform=(MessageForm)form;
    Date date=new Date();
    String title=messageform.getTitle();
    String content=messageform.getContent();
    Message message=new Message();
    message.setTitle(title);
    message.setContent(content);
    if(this.getEasyjdb.add(message)){
        return mapping.findForward("successed");
     }
    return mapping.findForward("faild");
}
```

这时的Message的映射方式随便哪种都行。同样，像延迟加载等配制都如出一辙了。

六、使用 EasyDBO 的开源项目介绍

6.1 EasyJF 开源 Blog 系统

EasyJF 开源 Blog 系统是一个由 EasyJF 开源团队组织开发的基于 Java 平台的开源博客系统。当前 Blog 已经实现了基本的基本的博客的书写、流量统计、排名、个人像册、RSS、支持自定义模板、静态 html 文件生成、权限系统、积分系统等功能。另外还将加入博客圈、音乐、专题等功能及更强大的权限系统支持。系统使用基于 OO 的方法设计，采用多层 B/S 构架，数据库持久层使用 EasyDBO，Web 层使用 EasyJWeb 框架，java 代码与页面完全分离，易扩展。欢迎广大 Java 开源爱好者下载交流，并请多提宝贵意见及建议！

系统演示：http://blog.easyjf.com

系统 SVN：http://svn.easyjf.com/repository/easyjf/easyjfblog/

系统源码下载：http://dl.easyjf.com/downloads/easyjf/blog/easyjf-blog-0.1.1.zip

6.2 简易 java 框架开源论坛系统(最新版本 0.5,更新时间 10 月 1 日)

简易 java 框架开源论坛系统拥有常用论坛系统的基本功能，集前台后台代码为一体，支持 UBB。该论坛系统使用基于 OO 的方法设计，采用多层 B/S 构架，数据库持久层主要使用简易数据库开源框架 EasyDBO，Web 层使用 EasyJWeb 框架，java 代码与页面完全分离，易扩展。欢迎广大 Java 爱好者下载使用。

系统演示：http://ent.easyjf.com

系统 SVN：http://svn.easyjf.com/repository/easyjf/easyjfbbs/

系统源码下载：http://dl.easyjf.com/downloads/easyjf-bbs-0.5.0.zip

6.3 简易 java 框架开源订销管理系统(最新更新:2006-4-3)

该系统是一个使用 Java 语言开发,以国内开源 Web MVC 框架 EasyJWeb 作系统引擎的 Java Web 应用系统.系统主要实现的功能有订单录入、打印、销售汇总、原料管理、客户管理、生产配料计算、报表打印、汇总、系统数据管理及维护等功能,是一个使用非常简单的编码方式实现的 Web 开源应用系统。

系统采用面向对象的设计方法，页面设计及系统逻辑分离，具有较好的扩展性。系统使用数据库中间件技术，支持 My SQL、MS SQL Server 等多种数据库系统平台。系统涉及到复杂表单数据提交、AJAX 无刷新数据提交、WEB 打印等常用应用软件中涉及到的技术。

系统在线演示地址：http://asp.easyjf.com 用户名:test 密码:test

源码下载：http://www.easyjf.com/download/erp0.1.zip

6.4 EasyJF 开源网上会议系统 iula-0.1.0(最新更新:2006-7-13)

EasyJF 开源网上会议系统 iula 是一个使用 AJAX+EasyJWeb+EasyDBO 及多线程技术技术开发的网上信息交流及互动系统，主要供 EasyJF 开源团队的成员网上会议使用，会议系统模拟传统的会议形式，可以同时开设多个不同主题的会议室，每个会议室需要提供访问权限控制功能，会议中能够指定会议发言模式（分为排队发言、自由发言两种），系统能自动记录每个会议室的发言信息，可以供参会人员长期查阅。

作为一个开源项目，iula-0.1.0 版本当前已经实现了 AJAX 基本框架搭建，网上文字信息的基本交流等功能，可以作为网上会议或类似信息交流系统的一个基本框架。

系统演示：http://www.easyjf.com/chatRoom.ejf?easyJWebCommand=show

源码下载：http://dl.easyjf.com/downloads/easyjf-iula-0.1.0.zip

安装说明：http://www.easyjf.com/html/20060713/2208218216744714.htm

svn 地址：http://svn.easyjf.com/repository/easyjf/easyjfiula

更多实用项目介绍，请随时关注 EasyDBO 官方网站:www.easyjf.com

七、结束语

EasyDBO 从项目发起到现在，已经历将近 7 个月了，这期间我们遇到了很多困难，也曾经有很多次放弃的念头，然而在广大开源爱好者鼓励下，终于有了这一次比较大的更新。

感谢与 EasyDBO 项目组全体成员的辛勤劳动及付出，感谢 EasyJF 其它项目成员给们做了大量的系统测试工作，更要感谢广大的开源爱好者对我们的建议及帮助。

当然，由于 EasyDBO 项目组开发人员技术水平有限，再加上本项目还没有得到比较彻底的测试与应用实践，因此这个项目中肯定存在着或多或少的问题。因此，我们非常需要得到广大开源爱者的关注与支持，欢迎大家给我们的开源作品提出批评或者建议，你所做的一切努力，都将给我们更好地改进和完善框架提供莫大的帮助。

八、联系我们

官方网站：www.easyjf.com
各地办公室联系电话：(请留意官方网站上的更新)。
学习交流 QQ 群：22642962,21727290,2320233,3639313
EasyJF 团队工作群众： 18230172
EasyJF 开源团队邮箱：easyjf at 163.com 、 easyjf at 126.com