

## 第26章 在Eclipse中使用Hibernate

### 26.1 概述

#### 26.1.1 持久化框架产生的背景和现状

在Java数据库项目中，由于数据库是关系型而非面向对象的。很多时候，用面向对象方法完成了前期的设计和分析，到了数据库层编程的时候就会变得很别扭难受，其中最痛苦的就是写面向过程的SQL语句。

J2EE开发主要由JSP、Servlet、JavaBean、EJB四部份组成。其中EJB是J2EE中一个比较重要的部份，它提供了企业级开发的所需要的分布式支持。但现实中的大部份项目都是单服务器的轻量级项目，一般都不会涉及到分布式的开发环境，这时候用EJB就象大炮打蚊子，蚊子没打到，房子却被打破个洞。EJB的笨重、复杂是出了名的，这一直让开发者很不满，其中EJB中Entity Bean受到的批评最多，现实项目中鲜有使用Entity Bean的成功范例。

开发者急切的需要一种符合Java编程习惯的、适合轻量级开发的、易于使用的数据库持久化解决方案。在这个背景下就产生了轻量级的数据库持久化技术，其中最主要的就是Hibernate、JDO。

Hibernate是一个没有什么名分的民间开源项目，有点象一个草莽英雄，但Hibernate从实用出发的设计思路，使得它脱颖而出成为最流行的持久化技术。Hibernate的作者Gavin King也一举成名，现已成为EJB3.0专家组的成员，并且在EJB 3.0的Entity Bean部份，将采用和Hibernate类似的设计方案。因此，Hibernate是开发者学习和使用持久化技术的一个比较好的选择，即使以后EJB3.0一统天下，所学的Hibernate知识也不会浪费。

JDO1.1是已被JCP（Java规范管理委员会、由SUN、IBM、Oracle等共同参与）通过的一个规范，由于它是正式的规范，所以也受到了开发者很大的期望。然而，2005年1月19日在JDO2.0规范的投票中，它被IBM、Oracle、BEA等几家厂商投了反对票，由于得票数不够，它未能获得通过而成为正式规范。这使得JDO的前景堪忧。

#### 26.1.2 Hibernate简介

Hibernate把开发者从数据库编程中隔离开来，它在数据库外裹了一层面向对象的外衣，Java程序中所有进出数据库的操作都交给Hibernate来处理，它会为我们自动生成SQL语句操作数据库。

有了Hibernate后，程序员不再要写繁琐的SQL语句，也不再要把实体对象一个个字段拆开又组装。说通俗一点，Hibernate就象一位卖苦力的民工，脏活累活它全包了。

下图26.1是Hibernate的示意方式。Hibernate提供了一个和SQL类似的HQL语句，但结合Hibernate后功能更强大，而且Hibernate能够根据实体对象的状态来自动对数据库进行更新和插入，很智能化。

HQL语句操作数据库其实也是要转化成标准的SQL语句的，Hibernate根据用户所编写的XML映射文件来实现HQL到SQL的自动转化。XML映射文件是Hibernate中最关键的配置文件，它定义了实体类和数据库表之间的关系，架起了两者间的桥梁，掌握XML映射文件的编写是掌握Hibernate的关键。

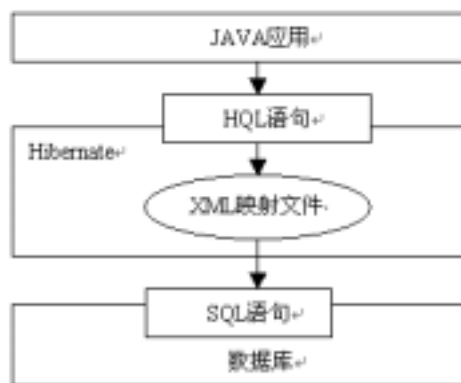


图26.1 Hibernate的示意

Struts提供视图和控制层的支持，Hibernate提供数据层的支持，两者是一个很不错的开发组合。Struts还只能用于WEB开发，而Hibernate还可以用于Application（包括Eclipse插件）的开发。

### 26.1.3 本章说明

Hibernate自带有中文文档，内容权威而全面，翻译得也还不错，是学习Hibernate的必看文档。不足的地方是，由于内容较多，主线不够突出，而且缺乏一个较为系统的实例。

Hibernate的使用很灵活，但有一些用法是常用的、主要的，有一些则是生僻的、很少用的。Hibernate中，应该掌握的重点内容是：Session概念、XML映射文件的配置、HQL的语法。本章用一个实例做主线来贯穿这些Hibernate的核心知识，读者可以顺着这条主线并结合Hibernate自带文档来学习。

本章的数据库表和数据模型借用了第22章的例子，具体请参阅“22.2.2节 面向对象的分析与设计”，和“22.2.3节 创建数据表”。

## 26.2 Hibernate的下载和安装（V0050）

### 26.2.1 下载

（1）访问Hibernate的官方网址<http://www.hibernate.org>，选择“Download”链接，如下图26.2所示：

（2）在下图26.3的新页面中，选择正式发布的Hibernate2.1.7c版。另外顺便把Hibernate Extensions2.1.3也下载了，此软件包里有一些Hibernate的辅助工具，在26.7节会用到。当然，Hibernate Extensions软件包并不是使用Hibernate开发所必须的。



图26.2 hibernate主页

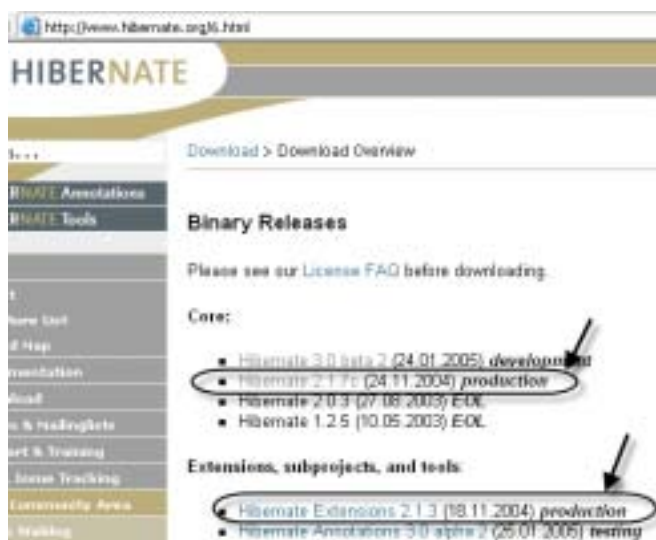


图26.3 下载版本选择

(3) 在下图26.4的新页面中，选择下载文件“ Hibernate-2.7.1c.zip ”

(4) 然后在列出的镜像中选择一个来下载。最后下载得到一个名hibernate-2.1.7c.zip的压缩包。解压后的目标结构如下图26.5所示：



图26.4 下载文件选择



图26.5 hibernate解压后的目标结构

主要目录及文件解释：

- hibernate2.jar文件是Hibernate的核心包。
- lib目录里有一些第三方支持包，安装时也要用到。
- src目录里是Hibernate（hibernate2.jar）的源文件。
- etc目录有一些可以参考的例子文件。
- doc目录包含Hibernate文档，在doc/reference/zh-cn目录下有其中文文档，分为pdf、多页面、单页面三种形式存放。

### 26.2.2 安装

(1) 将解压目录中的hibernate2.jar和lib目录下的所有\*.jar文件，复制到myweb项目的hello/WEB-INF/lib目录中。

其他说明：

- 其实并不需要复制lib目录下的所有\*.jar文件，本文只是为了安装上的方便。如果在

正式发布程序时,希望只包括真正用到包,可以参考解压目录lib中的README.TXT文件,里面有详细描述。或者参考Hibernate文档,里面也有部份描述。

- 注意不要将这些jar文件复制到TOMCAT/common/lib目录下,那是Tomcat全局库所在目录,有可能引起包冲突。
- Hibernate有一个包叫commons-logging-1.0.4.jar,以前安装struts时也复制了一个叫commons-logging.jar,两个包只保留一个即可,否则可能会引起包冲突。

(2)打开myweb的项目属性窗口,将新复制的hibernate2.jar包加入到项目引用“库”中,如下图26.6所示。

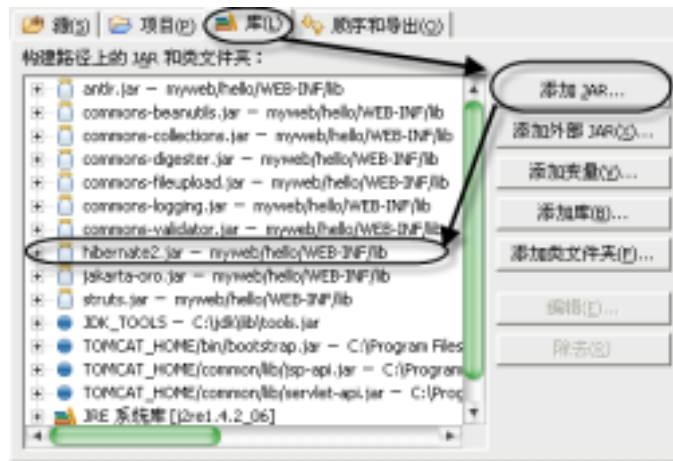


图26.6 项目引用“库”

其他说明：

- 库引用只是为了让得包中的类能在编程中被使用。虽然复制了很多第三方包,但在写代码的时候一般只用到hibernate2.jar包中的类。当然,将所有第三方包加入库引用,也不会出错。
- 如果你觉得“包资源管理器”视图显示的包太多,影响操作,可以用“过滤器”过滤掉一些,如下图26.7操作。



图26.7 过滤掉\*.jar的显示

(3)将Hibernate解压目录etc下的log4j.properties,复制到myweb项目的j2src目录下。

其他说明：

- log4j是一个日志输出软件包,前两步已经将它的包复制到项目lib目录中了。如果没有log4j.properties,控制台会出两条警告信息(下图26.8),log4j无法起作用。不过,这并不会和使开发无法前进,只是没有了一些输出信息。

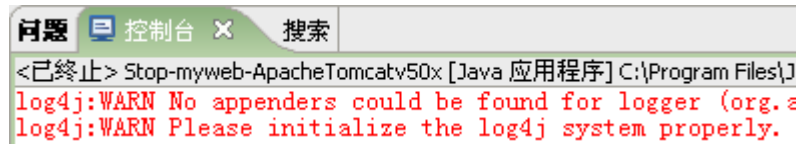


图26.8 没有log4j.properties

- Hibernate文档和某些文章都是说将log4j.properties复制到hello/WEB-INF/classes目录下。其实，本文复制到myweb/j2src目录，Eclipse也会自动将log4j.properties复制一份到hello/WEB-INF/classes，并且会保持两文件内容的同步。

(4) 在myweb/j2src目录创建一个hibernate.cfg.xml文件，这个文件是Hibernate的主配置文件。它分两大块，第一块是定义使用的数据库连接池，第二块是注册XML映射文件，具体内容如下：

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration
PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-2.0.dtd">

<hibernate-configuration>
  <session-factory>
<!-- database -->
<property name="connection.datasource">java:comp/env/jdbc/mysql</property>
<property name="show_sql">true</property>
<property name="dialect">net.sf.hibernate.dialect.MySQLDialect</property>

<!-- *.hbm.xml register here -->
<mapping resource="cn/com/chengang/sms/model/model.hbm.xml" />

  </session-factory>
</hibernate-configuration>
```

代码说明：

- connection.datasource设定所用的连接池。
- show\_sql设定在控制台是否显示Hibernate生成的SQL语句，开发期间用true，便于调试。
- dialect告诉Hibernate使用哪种SQL数据库方言(dialect)。不同的数据库都和SQL "标准"有一些出入，Hibernate会根据设置的方言来适应这些差异。要知道其他数据的方言名称，可以利用Eclipse的代码提示功能，在Java程序中键入“net.sf.hibernate.dialect.”然后按钮“Alt+/”键。
- model.hbm.xml是一个XML映射文件，这个文件创建在model目录下（内容以后将给出）。注意：这里用的是相对路径，cn字符串前面是没有“/”的。
- hibernate.cfg.xml还有一种hibernate.properties的写法，在Hibernate的解压目录etc可找到它的例子。两种写法选一种即可，本文选前一种。
- 某些属性对Hibernate的性能影响很大，比如batch\_size项设置成0和30，性能相差会有4倍以上。属性会有一个默认值，但如果所开发的项目需要优调性能，则可根据实际情况来重新设置。如果想了解hibernate.cfg.xml中更多的属性设置，可以查看Hibernate文档的“表 3.3. Hibernate配置属性”，那儿有属性的说明和建议值，本文不再复述。

## 26.3 一个简单的Hibernate实例

### 26.3.1 创建XML映射文件：model.hbm.xml

Hibernate之所以能够智能的判断实体类和数据表之间的对应关系，就是因为有XML映射文件。本步先在model目录下创建一个名为model.hbm.xml的XML映射文件，其内容如下：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping
  PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">

<hibernate-mapping>
  <class name="cn.com.chengang.sms.model.Grade" table="grade">
    <id name="id">
      <generator class="identity"/>
    </id>
    <property name="name"/>
  </class>

</hibernate-mapping>
```

代码说明：

- model.hbm.xml可以任意命名或放置于其他目录下，当然在hibernate.cfg.xml文件也要做相应的修改。笔者建议将它和它所对应的实体类放在一个包下，并用包名做文件名。
- <class>项定义了实体类和数据表之间的关系：name是实体类（用类全名），table是对应的数据表（表名不分大小写）。可以省略掉table属性，这时默认表名和实体类同名。在model.hbm.xml文件中可以设置多个<class>项，笔者建议将一个包中的所有实体类都集中在一个\*.hbm.xml文件中。
- <id>项定义了主键id字段所用的键值生成方法，identity是一种MSSQL、DB2、MySQL通用的主键值生成方法（Oracle不能用identity，可换成sequence）。要了解更多，可以查阅Hibernate文档的“5.1.4. id”。
- <property>子项定义了实体类和表字段的关联。本例只设置了定义类字段的name属性，还有一个column属性是定义数据库表字段名的，本例没有设置。Hibernate正是通过这里的设置建立起实体类和数据库表之间的字段对应关系。本例没有设置column，则Hibernate会默认它和name属性同名。假如，想将Grade实体类的name字段对应于数据库表的grade\_name字段，并将表字段的长度定义成16、不充许空值，则可以照如下设置：

```
<property name="name">
  <column name="grade_name" length="16" not-null="true"/>
</property>
```

<property>体现Hibernate的友好性：它可以设置得很详细，也可以很简洁，当设置简洁时，Hibernate会采用默认值。要了解更多关于<property>设置的内容，可以参阅Hibernate文档的“5.1.9. property”。

### 26.3.2 创建管理session的类：HibernateUtil

接下来创建一个关键的类：HibernateUtil，这个类用于管理session（生成、关闭）。session是Hibernate中最重要和使用最频繁的一个对象，实体对象都是通过它来和数据库交互。这个session和JSP的session不是一回事，倒是有一点点类似于JDBC的Connection，但Session比

Connection包含的内容更多，功能范围更广。在Hibernate的编程中将不会再使用Connection，而是通过session来和数据库交互。

HibernateUtil类的内容如下：

```
package cn.com.chengang.sms.db;
import net.sf.hibernate.HibernateException;
import net.sf.hibernate.Session;
import net.sf.hibernate.SessionFactory;
import net.sf.hibernate.cfg.Configuration;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
public class HibernateUtil {
    private static Log log = LogFactory.getLog(HibernateUtil.class);
    private static final SessionFactory sessionFactory;
    static {
        try {
            //实例化一个SessionFactory对象
            sessionFactory = new
Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            log.error("Initial SessionFactory creation failed.", ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static final ThreadLocal session = new ThreadLocal();

    public static Session currentSession() throws HibernateException {
        Session s = (Session) session.get();
        //当原session为空或已关闭时，打开一个新的Session
        if (s == null || !s.isOpen()) {
            s = sessionFactory.openSession();
            session.set(s);
        }
        return s;
    }

    public static void closeSession() throws HibernateException {
        Session s = (Session) session.get();
        session.set(null);
        if (s != null)
            s.close();
    }
}
```

程序说明：

- HibernateUtil可以任意取名。它是一个静态方法类，即类中的方法都是静态方法。
- SessionFactory是一个静态变量，它由static {...}静态代码块来初始化一个实例。注意，static {...}代码块比较特殊，它即不是方法也不是变量。

SessionFactory是线程安全的，在多线程下访问不会出问题。而Session不是线程安全的，所以使用了ThreadLocal对象把Session保存在当前工作线程中，以避免它被多个线程同时访问。这个机制相当重要，象WEB这种并发访问巨大的应用，Session用完后一定要及时关闭，否则会很快耗尽服务器的内存资源。

生成一个SessionFactory对象是很耗费时间和资源，所以在这里整个WEB系统共用一个SessionFactory。而生成一个Session对象的代价很小，在编程中千万不要把Session写成单例模式来进行实例共享，对Session的使用原则是用完就关闭，而且要

尽量早的关闭。

- 在currentSession()方法中，不仅要判断session为空，也要判断session是否已关闭（!s.isOpen()），在Hibernate文档中给出这个类的代码并没有对关闭做判断。如果没有对关闭做判断，则会在本实例的用户重复登录时导致错误。

### 26.3.3 创建一个用于测试的HibernateTest类

HibernateTest类类似于上一章的DbOperate类，主要提供数据库操作方法。在这里是编写了往Grade表插入一条记录的方法，以及取出Grade表中id值大于2的所有记录的方法。

```
package cn.com.chengang.sms.db;
import java.util.List;
import net.sf.hibernate.HibernateException;
import net.sf.hibernate.Query;
import net.sf.hibernate.Session;
import net.sf.hibernate.Transaction;
import cn.com.chengang.sms.model.Grade;
public class HibernateTest {
    public void insertGrade() throws HibernateException {
        Session session = HibernateUtil.currentSession();
        Transaction tx = session.beginTransaction();

        Grade grade = new Grade();//生成一个年级对象
        grade.setName("高四");

        session.save(grade); //将这个对象保存到数据库
        tx.commit();//提交
        HibernateUtil.closeSession();//关闭session
    }

    public List getGrades() throws HibernateException {
        Session session = HibernateUtil.currentSession();
        Transaction tx = session.beginTransaction();

        //创建一个条件查询语句
        String sql="select g from Grade as g where g.id > :id";
        Query query = session.createQuery(sql); //创建查询对象
        query.setInteger("id", 2); //设置查询参数
        List list = query.list(); //从数据库取出数据，并自动封装到List集合中

        tx.commit();//提交
        HibernateUtil.closeSession();//关闭
        return list; //返回数据集
    }
}
```

程序说明：

- 在Session中，每个数据库操作都是在一个事务(Transaction)中进行的，这样可以隔离不同的操作。Hibernate的事务(Transaction)是JDBC事务的更高层次的抽象，它提供了更好的灵活性和适应性。
- 以前都是将实体对象的字段一个个拆散并组合成SQL语句，或者从数据库取出数据后将字段值一个个封装到实体对象。现在用了Hibernate，就再也不必这么处理数据了，这是Hibernate有魅力的一个方面。
- getGrades方法中用到的sql字串不是JDBC的SQL语句，而是Hibernate自有的HQL语句。关于HQL的更多内容，请查阅Hibernate文档的“11. Hibernate查询语言”。

### 26.3.4 在hello目录下创建一个JSP文件：hibernateTest.jsp

hibernateTest.jsp使用HibernateTest类来获得数据，并将数据显示在页面上。

```
<%@ page contentType="text/html; charset=GBK" %>
<%@ page import="cn.com.chengang.sms.db.*"%>
<%@ page import="cn.com.chengang.sms.model.*"%>
<%@ page import="java.util.*"%>
<%
    HibernateTest db=new HibernateTest();
    db.insertGrade();//插入一个年级对象
    //取出所有年级对象，并显示在页面上
    for (Iterator it = db.getGrades().iterator(); it.hasNext();) {
        Grade g = (Grade) it.next();
        out.print(g.getId()+" "+g.getName()+"<br>");
    }
%>
```

### 26.3.5 总结

以上四步就完成了了一个简单的Hibernate实例，用浏览器运行hibernateTest.jsp的效果如下图26.9所示：



图26.9 hibernateTest.jsp的运行效果

本节用从底层到高层的次序来完成了一个实例的讲解，它虽然简单，但也反映了一个典型Hibernate程序的编写框架：

- HibernateUtil一经完成之后可以系统通用，以后很少改动。
- XML映射文件是项目前期要做的最重要的工作，在项目开发中后期就很少会改动它。对初学者来说，编写XML映射文件也是难点所在。
- HibernateTest.java是数据库操作类，这相当于以前的DbOperate类的功能。
- 最后，就是位于最高层的JSP文件：hibernateTest.jsp，这个文件主要是使用HibernateUtil来得到一个session，然后用HibernateTest类来操作数据库。

### 26.3.6 实践建议

- Lombok支持XML映射文件的热修改，当XML映射文件改动之后，Lombok会将它重新装入。不过这可能需要一两秒钟时间，具体时间要视读者所用电脑性能而定。
- 在Java编程中要时刻注意区分大小写，这是编程中出错较多的原因。

## 26.4 继续深入使用Hibernate ( V0060 )

### 26.4.1 概述

在Hibernate中最主要的就是XML映射文件和HQL查询语句。HQL和SQL相似，有过SQL经验的人可以很轻松的掌握HQL，所以Hibernate的难点集中在XML映射文件编写。为了让读者从实例中迅速掌握Hibernate的核心知识，本节将用Hibernate的知识来继续改写原有的用

户登录程序。

## 26.4.2 编写XML映射文件

在本步将把年级、班级、课程、用户四个实体类和表的映射关系设置清楚，这其中涉及到多对一关系、一对多关系、继承式实体类（用户类）等知识，关于用户类的设计方案及代码，参阅“22.2.2节 面向对象的分析与设计”。

给出XML映射文件model.hbm.xml的完整内容如下：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping
  PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">

<hibernate-mapping>
  <!--年级-->
  <class name="cn.com.chengang.sms.model.Grade">
    <id name="id">
      <generator class="identity"/>
    </id>
    <property name="name"/>
  </class>

  <!--班级-->
  <class name="cn.com.chengang.sms.model.SchoolClass">
    <id name="id">
      <generator class="identity"/>
    </id>
    <property name="name"/>
    <many-to-one name="grade" class="cn.com.chengang.sms.model.Grade"
column="grade_id" update="false" insert="false" />
  </class>

  <!--课程-->
  <class name="cn.com.chengang.sms.model.Course">
    <id name="id">
      <generator class="identity"/>
    </id>
    <property name="name"/>
  </class>

  <!--用户-->
  <class name="cn.com.chengang.sms.model.IUser" discriminator-value="I">
    <id name="id">
      <generator class="identity"/>
    </id>
    <discriminator column="type" type="character"/>
    <property name="userId"/>
    <property name="password"/>
    <property name="name"/>
    <property name="latestOnline"/>

    <!--用户.学生-->
    <subclass name="cn.com.chengang.sms.model.Student"
discriminator-value="B">
      <many-to-one name="schoolClass"
class="cn.com.chengang.sms.model.SchoolClass" column="schoolclass_id"/>
  </subclass>
  </class>
</hibernate-mapping>
```

```

        </subclass>

        <!--用户.老师-->
        <subclass name="cn.com.chengang.sms.model.Teacher"
discriminator-value="A">
            <set name="courses" table="iuser_course" inverse="true"
lazy="true">
                <key column="iuser_id"/>
                <many-to-many class="cn.com.chengang.sms.model.Course"
column="course_id"/>
            </set>
        </subclass>
    </class>
</hibernate-mapping>

```

代码说明：

(1) 初学者在设置XML映射文件时常常被一对多、多对一等关系搞得很迷糊，感觉这些关系的设置很难捉摸。这里给出一个重要的口诀：“以实体类的字段为依据来配置XML映射文件：类的字段有则映射有、类的字段无则映射无”。

(2) 年级实体类的设置中，把原来的table="grade"去掉了，没有table属性，则默认为表和类同名。这也是当初创建表时为什么用类名做表名的原因：一来名称相同方便记忆；二来配置XML映射文件时也图个方便简洁。

(3) 班级实体类的第三个字段“grade”是年级实体类的类型，一个年级有多个班级，所以班级对年级是多对一的关系。设置多对一关系的字段不再用<property>项，而是用<many-to-one>项来设定，如下：

```

<many-to-one name="grade" class="cn.com.chengang.sms.model.Grade"
column="grade_id" update="false" insert="false" />

```

其中name属性是班级实体类中的字段名grade；class属性是grade字段的类型（全类名）；column属性对应班级数据表的字段名。

update、insert两项属性的默认值是true，本处设为false。如果设成true，则Hibernate在更新班级实体对象所对应的数据库数据的同时，也会自动更新年级表数据。但我们一般都不会通过班级实体对象来自动更新年级表，因为年级表基本是不会动的，这种更新不仅没什么用，反而影响效率。注意：设为false之后，并不是指不能更新年级表，而是指不通过班级实体来自动更新年级表。

有些读者在这里也许会问：“班级对年级是多对一关系，反过来，年级对班级就是一对多关系。为什么只在设置班级实体时，指定了<many-to-one>关系，而在设置年级实体时，没有反过来指定<one-to-many>关系呢？”

年级对班级是一对多关系，是没错。但正如前面所说的口诀，年级实体类中并没有指向班级的字段，所以在xml映射中就不设置它的一对多关系。假设在年级实体类中增加一个字段“private List schoolClasses”，用来存放年级下的所有班级，这时就应该用<one-to-many>来设置这种一对多关系了，否则班级记录就不会自动通过Hibernate被抓取到schoolClasses字段中。

(4) 用户类是典型的继承式实体类，它的XML映射设置如下：

- <class>项定义接口IUser时多加了一个discriminator-value="I"。
- <discriminator column="type" type="character"/>是指增加一个名为type的character型字段，这个字段用来区分不同的子类。
- 在设置IUser时定义好共同字段。
- 在<subclass>项定义各子类的独有字段，其中discriminator-value属性要求各不相同，此值将存入表的type字段中。

关于这种继承式实体设置的更多内容，请参阅Hibernate文档的“第8章 继承映射”，此文档里面共有三种方案，本文选择的是“所有用户类合用一个表”的第一种方案。在本书第22章设计用户表时，也提到了这三种方案，大家可以参照阅读，以便加深理解。

(5) 学生类有一个“班级”字段，学生和班级是多对一关系，所以也用了<many-to-one>来设置此字段。学生类里有一个BUG：getSchoolclass方法的倒数第5个字符c，应该是大写，请大家更正过来。

关于这个BUG，这里再次说明一下：Hibernate是根据方法名来设置映射的。比如<many-to-one name="schoolClass"，对应的是getSchoolClass、setSchoolClass两方法，也就是把schoolClass第一个字符大写后，分别加上set、get字串。在上一章讲Struts时，也提到了这一点。不过，一般习惯还是说对应于类的字段名，其实真正对应的是方法名。

(6) 老师类的设置，如下：

```
<set name="courses" table="iuser_course" inverse="true" lazy="true">
  <key column="iuser_id"/>
  <many-to-many class="cn.com.chengang.sms.model.Course"
column="course_id"/>
</set>
```

- 老师类和课程是典型的多对多关系：一个老师可以教多门课程，一门课程也可以由多名老师教。
- 因为老师类中的课程字段的定义是“Set courses”，所以用<set>项来设置，除此之外还有<list><map><array>等。
- name="courses"对应于老师类字段courses。
- 多对多需要一个新表（仅两字段）来保存两者之间的关联，table="iuser\_course"就是这个新表的表名。
- inverse="true"，设置反转。因为多对多有两端，当两端同时都做了修改时，Hibernate需要根据inverse项的设置来判断应该依据那一端来做更新操作。如果两端都同时设为inverse="true"，或同时省略inverse，极可能导致更新冲突，所以一般是任选一端（且仅一端）来设置inverse="true"。
- lazy="true"（默认false），设置延迟。

延迟是Hibernate中非常重要的概念，主要在设置多对多、一对多关系中使用。比如，要显示一个仅有老师名称的列表，如果没有设置课程字段延迟获取，那么Hibernate会将老师对应的课程记录也一并从数据库取出，这样就有点浪费了；如果设置了延迟，那么Hibernate会在真正用到课程记录时，才会去数据库里取，这样就显得智能一些。

但延迟也不是没有缺点：1) 它要求在使用完延迟型数据（如课程数据）之前，Session不能关闭。2) 如果一个页面肯定要显示所有课程记录，这时不延迟一次取完，要比延迟分批取的效率高得多。

所以使用延迟要平衡利弊，根据实际情况做出选择。以本例来说明，如果老师对应的课程记录特别多，也就是说一次性取完相应课程记录的代价很大，那么你应该选择延迟。但象本例这种，课程表极小（对于中学来说，也就10种课程左右），每个老师对应的课程数很少（一个老师一般只教一种课程，除非是一人全包的乡村小学老师），这时完全可以不用延迟。本例使用延迟仅仅是为了演示。

如果已经设置了延迟，但某些特殊情况，又需要提前关闭session，则有如下解决方法：

```
Hibernate.initialize(user.getCourses()); //强行加载延迟字段的数据
session.close(); //关闭
Set set=user.getCourses();//在session关闭之后依然可以取得延迟数据
```

### 26.4.3 数据库操作类的实现

Hibernate的一个优点就是抽象于数据库，能够适应多数据库，所以过去数据层的设计也就没有必要存在了。把MysqlOperate、OracleOperate、SqlServerOperate、AbstractDbOperate、ConnectManager、SmsFactory五个类都删除掉，还有SMS类中的无用常量也可以删除掉，仅留CURRENT\_USER常量。然后将DbOperate由接口改成普通类，再将上一节写在HibernateTest.java中的数据库操作代码写在DbOperate类中。DbOperate的具体代码如下：

```
package cn.com.chengang.sms.db;
import java.util.Collections;
import java.util.List;
import net.sf.hibernate.HibernateException;
import net.sf.hibernate.Query;
import net.sf.hibernate.Session;
import net.sf.hibernate.Transaction;
import cn.com.chengang.sms.model.IUser;
public class DbOperate {
    /**
     * 根据用户名得到用户对象，如返回NULL则表示此用户不存在
     */
    public IUser getUser(String userId) throws HibernateException {
        Session session = HibernateUtil.currentSession();
        IUser user = getUser(session, userId);
        session.close();
        return user;
    }

    /**
     * 根据用户名得到用户对象，如返回NULL则表示此用户不存在
     */
    public IUser getUser(Session session, String userId) throws
HibernateException {
        Transaction tx = null;
        IUser user = null;
        try {
            tx = session.beginTransaction();
            Query q = session.createQuery("from " + IUser.class.getName() +
" where userId=:userId");
            q.setParameter("userId", userId);
            List list = q.list();
            if (!list.isEmpty())
                user = (IUser) list.get(0);
            tx.commit();
        } catch (HibernateException e) {
            if (tx != null)
                tx.rollback();
            throw e;
        }
        return user;
    }

    /**
     * 根据分页信息对象QueryInfo得到应用的用户记录
     * 要求QueryInfo中已有currentPage和pageSize的数据
     */
    public List getUsers(Session session, QueryInfo qi) throws
HibernateException {
```

```

// 得到总记录数
String sql = "select count(*) from " + IUser.class.getName();
qi.rsCount = ((Integer) session.iterate(sql).next()).intValue();
if (qi.rsCount == 0) // 等于0表示没有记录
    return Collections.EMPTY_LIST;
// 算出总页数
if (qi.rsCount % qi.pageSize == 0)
    qi.pageCount = qi.rsCount / qi.pageSize;
else
    qi.pageCount = (qi.rsCount / qi.pageSize) + 1;
// 算出起始位置 = (当前页号-1)*每页记录数
int start = (qi.currentPage - 1) * qi.pageSize;

Transaction tx = null;
List list = null;
try {
    tx = session.beginTransaction();
    Query q = session.createQuery("from " + IUser.class.getName());
    q.setFirstResult(start);
    q.setMaxResults(qi.pageSize);
    list = q.list();
    tx.commit();
} catch (HibernateException e) {
    if (tx != null)
        tx.rollback();
    throw e;
}
return list;
}
}

```

程序说明：

- 因为用户的老师类的课程字段采用了延迟，在前面已经讨论过，在获取延迟字段的数据之前不能关闭Session，所以取得用户的方法分成了两种：一个带Session参数，一个不带。如果不会用到课程数据，就用不带Session的方法，方便一些。
- 在第二个方法中的IUser.class.getName()得到的是IUser的全类名“cn.com.chengang.sms.model.IUser”，在HQL查询中，不用在查询之前加“select \*”字符串，而且表名是用类全名来代替，这也体现了Hibernate面向对象的风格。
- 第三个方法是分页式取数据的方法，q.setFirstResult(start)是设置起始记录位置，q.setMaxResults(qi.pageSize)是设置本次要取的记录数。
- 在这里不仅捕获了HibernateException异常，并且在处理完异常后，再次将此异常抛出。这是一种较标准的写法，是为了在类外程序中使用此方法时，能够再次捕获异常，并做一些处理。

#### 26.4.4 修改使用DbOperate类的程序

(1) LogonAction类。

此类代码基本不变，仅将原来取数据的两行代码：

```

DbOperate db = SmsFactory.getDbOperate();
IUser user = db.getUser(userId);

```

更改为：

```

IUser user=new DbOperate().getUser(userId);

```

(2) 对显示用户列表userList.jsp文件修改的示意如下：

.....

```

<%@ page import="net.sf.hibernate.*"%>
.....
<%
QueryInfo qi=new QueryInfo();
qi.currentPage=1; //显示第一页
qi.pageSize=100; //每页100条记录
Session hsession = HibernateUtil.currentSession();
List list =new DbOperate().getUsers(hsession,qi);
for (Iterator it = list.iterator(); it.hasNext();) {
    IUser user = (IUser) it.next();
%>
    <tr>
        <td><%=user.getId()%></td>
        .....
    <%
    }
    hsession.close();
%>
</table>
</BODY>
</HTML>

```

程序说明：

- 在文件头增加了一个<%@ page import来引用hibernate包。
- 将Session对象取名hsession是为了避免和JSP的默认对象session同名。
- hsession必须要在取数据循环结束后才能关闭，因为老师类的课程字段用的是延迟获取数据的方式。

## 26.5 实现用户的修改、删除功能（V0070）

### 26.5.1 程序界面效果及功能说明

这一节将同时需要Struts和Hibernate的知识，实例完成后的用户列表页面如下图26.10所示：



ID	用户名	密码	姓名	班级	课程	最后登录时间	修改	删除
1	chen	123	陈刚		数学	2005-01-01 00:00:00.0	<a href="#">修改</a>	<a href="#">删除</a>
3	tang	123	唐明	135		2005-01-03 00:00:00.0	<a href="#">修改</a>	<a href="#">删除</a>
4	jing	123	金春	136		2005-01-04 00:00:00.0	<a href="#">修改</a>	<a href="#">删除</a>
5	chai	123	蔡河	136		2005-01-05 00:00:00.0	<a href="#">修改</a>	<a href="#">删除</a>
6	Lieu	123	刘邦	137		2005-01-06 00:00:00.0	<a href="#">修改</a>	<a href="#">删除</a>

图26.10 用户列表页面

单击“删除”时，不提问，直接从数据库中删除记录，并再次返回用户列表页面。

单击“修改”时，打开下图26.11所示的页面。



图26.11 修改页面

单击“修改”按钮后，再次返回用户列表页面，如下图26.12。

ID	用户名	密码	姓名	班级	课程	最后登录时间	修改	删除
1	chen	123	陈刚		数学	2005-01-01 00:00:00.0	<a href="#">修改</a>	<a href="#">删除</a>
3	tang	123	唐明	135		2005-01-03 00:00:00.0	<a href="#">修改</a>	<a href="#">删除</a>
4	jing	123	金琴	136		2005-01-04 00:00:00.0	<a href="#">修改</a>	<a href="#">删除</a>
	chui	123	楚河	136		2005-01-05 00:00:00.0	<a href="#">修改</a>	<a href="#">删除</a>
6	gogo	555	嘎科	137		2005-01-06 00:00:00.0	<a href="#">修改</a>	<a href="#">删除</a>

图26.12 修改后的用户列表页面

## 26.5.2 修改DbOperate类

在DbOperate类分别加入删除数据、插入更新数据的方法，每种方法都提供带Session参数和不带Session参数的两种。saveOrUpdate同时具有插入和更新的功能，从这一点也可以体现Hibernate的高度智能化，的确是大大简化了操作数据表的工作。除了本例通过一个查询sql来删除，还有一种常用的传入实体类的删除方法：session.delete(Object obj)，Hibernate会根据实体类型去相应的表中将记录删除。

加入到DbOperate类的代码如下：

```
/**
 * 用HSQL语法来删除数据,参数sql是一个查询字符串
 */
public void delete(String sql) throws HibernateException {
    Session session = HibernateUtil.currentSession();
    delete(session, sql);
    session.close();
}
public void delete(Session session, String sql) throws HibernateException {
    Transaction t = null;
    try {
        t = session.beginTransaction();
        session.delete(sql);
        t.commit();
    } catch (HibernateException e) {
        if (t != null)
            try {
                t.rollback();
            } catch (Exception e2) {}
        throw e;
    }
}
/**
 * 插入或更新实体对象所对应的记录
 */
```

```

    public void saveOrUpdate(Object obj) throws HibernateException {
        Session session = HibernateUtil.currentSession();
        saveOrUpdate(session, obj);
        session.close();
    }
    public void saveOrUpdate(Session session, Object obj) throws
HibernateException {
        if (session != null && obj != null) {
            Transaction t = null;
            try {
                t = session.beginTransaction();
                session.saveOrUpdate(obj);
                t.commit();
            } catch (HibernateException e) {
                if (t != null)
                    try {
                        t.rollback();
                    } catch (Exception e2) {}
                throw e;
            }
        }
    }
}

```

### 26.5.3 修改用户列表userList.jsp文件

在原有的userList.jsp的每条记录后面加入“修改”、“删除”的超链接，如下：

```

<td><A HREF="/hello/user/userAction.do?method=showUser&userId=<%=user.g
etUserId()%>">修改</A></td>
<td><A HREF="/hello/user/userAction.do?method=removeUser&id=<%=user.getI
d()%>">删除</A></td>

```

(1) 以用户chen为例，给出相应的IE地址显示如下：

修改的IE地址显示：

```
http://127.0.0.1:8080/hello/user/userAction.do?method=showUser&userId=chen
```

删除的IE地址显示：

```
http://127.0.0.1:8080/hello/user/userAction.do?method=removeUser&id=1
```

(2) 在这里用的UserAction，稍后创建，它继承自struts的DispatchAction类(参阅“25.5.4节 使用DispatchAction类”)，并有三个方法removeUser、showUser、modifyUser分别对应于删除、显示、修改。

(3) 要修改用户，首先得把它的值显示出来，所以method参数值为showUser。修改是使用userId参数(用户名)，删除是使用id参数(自动递增主键)，当然，也可以改为通过userId来删除记录。

(4) 特别强调：

- 链接的HREF和表单的action在地址定位上有差别，所以表单action是用/user/action.do，而链接HREF是用/hello/user/action.do或action.do。这是容易忽略的一点。
- method参数的值必须是UserAction中的一个方法名
- userId、id两个参数必须是UserAction对应的UserForm类中的字段。

### 26.5.4 修改struts-config.xml

将下面的XML块，加入到<action-mappings>...</action-mappings>之间。

```

<action
    path="/user/userAction"
    type="cn.com.chengang.sms.user.UserAction"

```

```

parameter="method"
name="userForm"
scope="request"
validate="true"
input="/user/userList.jsp"
>
<forward name="modifyUserView" path="/user/modifyUser.jsp"/>
<forward name="modifySuccess" path="/user/userList.jsp"/>
</action>

```

代码说明：

- <action>项的path属性定义了UserAction。
- parameter属性是DispatchAction型Action所必须的。
- name="userForm"，此项设置说明UserAction也是用userForm来封装表单数据，当然，userForm还要加入一些字段，在后面会给出其代码。
- 这里设置了两个转发，一个是修改页面modifyUserView，一个是修改成功后的返回页面modifySuccess。

### 26.5.5 修改UserForm类

UserForm类需要做如下三处修改：

(1) 由于修改页面和登录页面合用UserForm类，所以要往类中再多加入修改页面用到的三个字段（同时还有相应的set/get方法），如下：

```

private Long id = null; //数据库ID
private String name = null; //姓名
private Date latestOnline = null; //最后登录时间
//-----set/get方法，省略----

```

(2) 因为UserAction继承自DispatchAction类的，这种类型的Action要求UserForm继承ValidatorActionForm，否则无法使用“25.6.2 方法二”中的验证方式。将UserForm的父类改为ValidatorActionForm

```

public class UserForm extends ValidatorActionForm {
    .....
}

```

(3) 最后还必须将UserForm中的validate方法删除，否则修改用户页面无法被打开。

从这里可以看到Struts程序的代码复用率很高，共用一个UserForm，不仅减少了创建ActionForm的个数，还能共用验证机制。Struts允许将系统中所有表单用到的字段合在一个ActionForm中，不过，不推荐这样做。也有人认为应该一个JSP表单对应一个ActionForm，但这种方案的代码复用率太低。

笔者建议将同类型表单所用到的字段合在一个ActionForm中，就象UserForm一样。这样即不会产生太多ActionForm，又有很好的代码复用。当然，这里面没有绝对的法则，读者应该根据自己实际的开发情况来选择。

### 26.5.6 创建UserAction类，其代码如下：

```

package cn.com.chengang.sms.user;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.actions.DispatchAction;

```

```

import cn.com.chengang.sms.db.DbOperate;
import cn.com.chengang.sms.model.IUser;
public class UserAction extends DispatchAction {
    private final static String USER = "modiUser";

    /**
     * 删除用户的Action
     */
    public ActionForward removeUser(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response) throws Exception {
        UserForm actionForm = (UserForm) form;
        Long id = actionForm.getId();
        String sql = "from " + IUser.class.getName() + " where id=" + id;
        new DbOperate().delete(sql);
        return (mapping.getInputForward());
    }

    /**
     * 修改用户的Action
     */
    public ActionForward modifyUser(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response) throws Exception {
        UserForm actionForm = (UserForm) form;
        IUser user = (IUser) request.getSession().getAttribute(USER);
        request.getSession().removeAttribute(USER);
        user.setUserId(actionForm.getUserId());
        user.setPassword(actionForm.getPassword());
        user.setName(actionForm.getName());
        new DbOperate().saveOrUpdate(user); //更新数据库
        return (mapping.findForward("modifySuccess"));
    }

    /**
     * 显示一个用户的Action
     */
    public ActionForward showUser(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response) throws Exception {
        UserForm actionForm = (UserForm) form;
        String userId = actionForm.getUserId();
        IUser user = new DbOperate().getUser(userId);
        actionForm.setUserId(user.getUserId());
        actionForm.setPassword(user.getPassword());
        actionForm.setName(user.getName());
        actionForm.setLatestOnline(user.getLatestOnline());
        request.getSession().setAttribute(USER, user); //保存用户状态
        return (mapping.findForward("modifyUserView"));
    }
}

```

#### 程序说明：

(1) 在显示用户的方法showUser中，先取得的用户对象，然后再将要显示在界面上的值转给UserForm。在修改用户的页面中，UserForm的字段值会自动显示在相应的文本框中。

(2) 在程序中定义了一个字符串常量USER。如果要经常用到某个字符串，最好定义成常量，否则容易因为书写错误而产生难以发现的BUG。

(3) 可以通过new DbOperate().saveOrUpdate(user)来直接更新user代表的数据库记录，但这要求让user对象从showUser中取出后，到modifyUser中还能用。本例采用HttpSession来维

持user的状态。

HttpSession是保存在服务器端的，通常HttpSession是30分钟左右失效（这是可设置的），HttpSession本身所占内存并不大，但在此期间，因为它保持着对user对象的引用，这使得user对象及user引用的其他对象都无法被JVM垃圾回收器回收。所以在使用HttpSession完之后，要记得用removeAttribute(USER)方法将HttpSession对user的引用清除。

对于user对象状态的保持，可能还会有以下几种行不通的想法：

- 用request来保持状态。  
分析：这是不行的，user状态可从showUser方法保持到modifyUser.jsp页面，但在modifyUser.jsp页面提交修改后，转到modifyUser方法中时，user对象已失效。
- 在UserForm创建一个user字段。  
分析：user状态一样无法保持到modifyUser方法里，当提交表单转到modifyUser时，所有在表单里没有被设置的UserForm字段值都会被清空。
- 在UserAction类创建一个user字段来保存user对象。  
分析：这是错误的。Struts的机制是：UserAction在整个系统只有一个实例，因此UserAction类的user字段会被所有用户线程共用。

### 26.5.7 创建modifyUser.jsp

modifyUser.jsp的作用是用文本框显示用户数据，当修改完成后，单击“修改”按钮就可以将这些修改提交到数据库里。

```
<%@ page contentType="text/html; charset=GBK"%>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ page import="cn.com.chengang.sms.model.*"%>
<HTML>
<HEAD>
<TITLE>修改用户</TITLE>
<META http-equiv=Content-Type content="text/html; charset=GBK">
</HEAD>
<BODY>
<html:form action="/user/userAction.do?method=modifyUser" focus="userId"
onsubmit="return validateUserForm(this)">
<table width="100%">
<tr>
<td>
<td>
用户名：<html:text property="userId" maxlength="10" />
<font color="red"><html:errors property="userId"/></font>
</td>
</tr>
<tr>
<td>
<td>
密 码：<html:text property="password" maxlength="20"/>
<font color="red"><html:errors property="password"/></font>
</td>
</tr>
<tr>
<td>
<td>
姓 名：<html:text property="name" maxlength="20" />
<font color="red"><html:errors property="name"/></font>
</td>
</tr>
<tr>
<td>
</td>
</tr>
</table>
</html:form>
```

```

        最后登录时间 :
        <bean:write name="userForm" property="latestOnline" />
    </td>
</tr>
<tr>
    <td>
        <html:submit>修改</html:submit><html:reset>重填</html:reset>
    </td>
</tr>
</table>
</html:form>
<html:javascript dynamicJavascript="true" staticJavascript="true"
formName="userForm" />
</BODY>
</HTML>

```

程序说明：

- 在页头加入了<tags/struts-bean>标签的声明，并在最后登录时间值的输出时使用了<bean:write>标签。在<bean>标签里的name属性引用了userForm对象，userForm对象是struts在JSP页面里隐式生成的。
- 此页面和logon.jsp是非常相似的，特别是对值进行验证机制完全一样。

## 26.6 解决Tomcat的中文问题

在本例修改用户的姓名时，会出现中文乱码问题。过去很多人在WEB编程中都是用字符编码转换函数来解决中文问题，其实没有必要这么麻烦。本节给出Tomcat中文问题的一个方便的解决方案，不必使用编码转换函数，就可以将其中文问题彻底解决（假设读者已经做好了MySQL数据库的字符集设置，参阅“22.1.5节 解决Java的中文问题”）。

(1) 将Tomcat的webapps\jsp-examples\WEB-INF\classes\filters目录中的两个源文件RequestDumperFilter.java、SetCharacterEncodingFilter.java复制到cn.com.chengang.sms包下（放在其他包也可以），然后将两源程序的包定义改为“package cn.com.chengang.sms;”

(2) 在hello/WEB-INF/web.xml里的</web-app>之前加入以下代码段。之后，如果是用XML插件（如XMLBuddy）来编辑这个文件，可能会提示一些警告信息，这是因为<filter>等项没有在XML文件前头的DTD文件里定义。要去掉这些警告信息，只要将前面的DTD定义项删除就可以了。

```

<filter>
    <filter-name>Set Character Encoding</filter-name>
    <filter-class>cn.com.chengang.sms.SetCharacterEncodingFilter</filter-
class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>GBK</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>Set Character Encoding</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

```

(3) 以后编写JSP文件时，记得加入这么两句

```

<%@ page contentType="text/html; charset=GBK"%>
<META http-equiv=Content-Type content="text/html; charset=GBK">

```

## 26.7 Hibernate的自动生成工具

在26.4节给出的XML映射文件设置口诀：“以实体类的字段为依据来配置XML映射文件：类的字段有则映射有、类的字段无则映射无”，实际上是“实体类 < XML映射 < 数据库表”这三者之间对应关系的一个反映。通过本节的学习，读者会发现，这三者之间是可以互相用工具来转化的。

### 26.7.1 由XML映射文件生成数据库表

Hibernate可以由XML映射文件自动生成数据库表，这简化了创建表格的工作。具体实现方法如下：

(1) 将以下两方法添加到HibernateUtil类中。

```
/**
 * 创建数据库表。如果表存在将会被删除重建，同时sql语句输出到c:\sms.sql
 */
public static void createDbTable() throws HibernateException {
    Configuration conf = new Configuration().configure();
    SchemaExport dbExport = new SchemaExport(conf);
    dbExport.setOutputFile("c:\\sms.sql");
    dbExport.create(true, true);
}

/**
 * 增量式更新数据库表。
 * (1) 会将XML映射文件中新增的字段加入到表中，表的原数据不会被抹掉。
 * (2) 这种更新是通过字段来判断的，如果表字段和XML映射文件字段同名，
 *     但类型和长度不同，表中的字段类型和长度也不会被更新。
 * (3) 如果表中的字段在XML映射文件中没有定义，此表字段也不会被删除。
 */
public static void updateDbTable() throws HibernateException {
    Configuration conf = new Configuration().configure();
    new SchemaUpdate(conf).execute(true, true); //增量
}
```

(2) 在hello目录下创建一个叫createDbTable.jsp的JSP页面来调用上面的两个方法，其代码如下：

```
<%
cn.com.chengang.sms.db.HibernateUtil.createDbTable();
//cn.com.chengang.sms.db.HibernateUtil.updateDbTable();
%>
```

注意：运行 createDbTable.jsp 之前要保证已经存在了 sms 库，否则会出错。本操作会自动将 sms 库的所有表都删除重建。

打开地址“<http://127.0.0.1:8080/hello/createDbTable.jsp>”，网页执行后无显示，但控制台会有输出，而且数据库已经产生变化，如下图26.13所示。

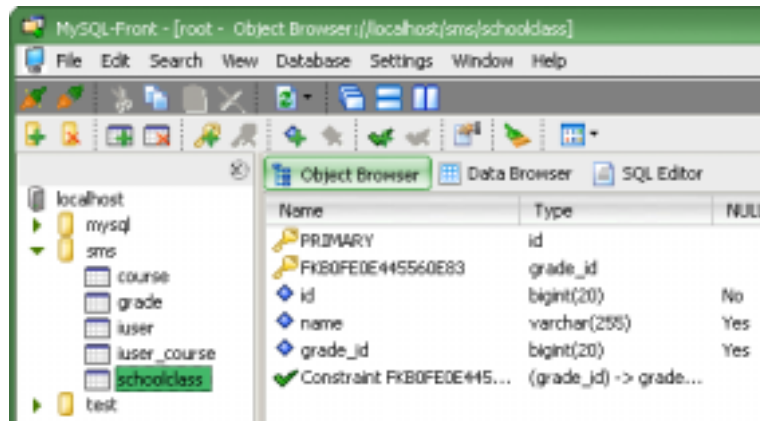


图26.13 Hibernate自动创建的表

自动生成的表和原来手工创建的表不同，我们可以重新修改XML映射文件，加入更多属性来定义一个字段，否则Hibernate对没有定义的属性(比如字段长度)会自动使用默认值。不过一般来说，由XML映射文件生成表之后还是需要进行一些手工修整。

### 26.7.2 由实体类自动得到XML映射文件：class2hbm

在hibernate-extensions的工具包中提供了根据实体类生产XML映射文件的功能，具体的操作步骤如下：

(1) 将以前下载的hibernate-2.1.7c.zip、hibernate-extensions-2.1.3.zip分别解压缩。前者解压后hibernate2.jar的路径为：D:\Downloads\hibernate-2.1.7c\hibernate-2.1\hibernate2.jar，后者解压后的批处理文件的路径为：D:\Downloads\hibernate-extensions-2.1.3\tools\bin，并将mysql的JDBC连接包复制到此hibernate-extensions-2.1.3\tools\bin目录下。

(2) 修改setenv.bat，这是成败关键的一步。setenv.bat里负责环境变量的设置，里面的路径和某些包名都要根据实际情况来重新设置。本文对setenv.bat做了一些调整，其内容如下：

```
@echo off
set HIBERNATE_HOME=D:\Downloads\hibernate-2.1.7c\hibernate-2.1
set HIBERNATETOOLS_HOME=%~dp0..
set
JDBC_DRIVER=%HIBERNATETOOLS_HOME%\bin\mysql-connector-java-3.0.16-ga-bin.jar
set CORELIB=%HIBERNATE_HOME%\lib
set LIB=%HIBERNATETOOLS_HOME%\lib
set
CP=%CLASSPATH%;%JDBC_DRIVER%;%HIBERNATE_HOME%\hibernate2.jar;%CORELIB%\commons-logging-1.0.4.jar;%CORELIB%\commons-lang-1.0.1.jar;%CORELIB%\cglib-full-2.0.2.jar;%CORELIB%\dom4j-1.4.jar;%CORELIB%\odmg-3.0.jar;%CORELIB%\xml-apis.jar;%CORELIB%\xerces-2.4.0.jar;%CORELIB%\xalan-2.4.0.jar;%LIB%\jdom.jar;%CORELIB%\commons-collections-2.1.1.jar;%LIB%\..\hibernate-tools.jar
```

(3) 将hello\WEB-INF\classes目录下的整个cn目录(如下图26.14)复制到D:\Downloads\hibernate-extensions-2.1.3\tools\bin目录。

特别强调：

- 复制的是整个cn目录，而非仅仅类文件。
- class2hbm程序需要的是编译好的class文件，而不是java源代码。



图26.14 复制整个cn目录

(4) 进入DOS窗口，并转到D:\Downloads\hibernate-extensions-2.1.3\tools\bin目录中，然后键入以下命令，将SchoolClass类对应的映射设置输出到当前目录下的abc.txt文件中，结果如下图26.15所示。

```
class2hbm cn.com.chengang.sms.model.SchoolClass --output=abc.txt
```



图26.15 生成的结果文件

特别强调：

- SchoolClass.class等类文件不能图方便放在和class2hbm.bat同一目录下，然后想这样class2hbm Teacher --output=abc.txt。这是错误的，得不到结果。
- abc.txt也可以改名成Teacher.hbm.xml或其他名称，它只是个文本文件。

如果用记事本打开abc.txt，因为没有分行，会很难看，最好用Editplus或者复制到Eclipse中打开。abc.txt的内容如下。

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 2.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping>
  <!-- cn.com.chengang.sms.model.SchoolClass root -->
  <class name="cn.com.chengang.sms.model.SchoolClass"
table="SchoolClass">
    <id name="id" type="long" column="id">
      <generator class="native"/>
    </id>
    <property name="name" column="name" type="string"/>
    <component name="grade" class="cn.com.chengang.sms.model.Grade">
      <property name="name" column="name_1" type="string"/>
      <property name="id" column="id_1" type="long"/>
    </component>
  </class>
</hibernate-mapping>
```

```
</component>
</class>
</hibernate-mapping>
```

上面的文件和以前手工设置的有些不同，因为Hibernate的XML映射文件设置是很灵活的，实现同一目的，会有不同的手段和途径。

### 26.7.3 由XML映射文件得到实体类：hbm2java

hbm2java.bat和class2hbm.bat的同一个目录下，它可以由XML映射文件得到实体类。举例说明：由上面生成的XML映射文件abc.txt来得相应的实体类，并将实体类源代码输出到当前目录下的d123目录中，命令如下：

```
hbm2java abc.txt -output=d123
```

命令执行后，在当目录的子目录“d123\cn\com\chengang\sms\model”中生成了两个文件：SchoolClass.java、Grade.java。本文将SchoolClass.java文件中的注释删除并做简单整理后，得到内容如下。

```
package cn.com.chengang.sms.model;
import java.io.Serializable;
import org.apache.commons.lang.builder.ToStringBuilder;
public class SchoolClass implements Serializable {
    private Long id;
    private String name;
    private cn.com.chengang.sms.model.Grade grade;

    public SchoolClass() {}
    public SchoolClass(cn.com.chengang.sms.model.Grade grade) {
        this.grade = grade;
    }
    public SchoolClass(String name, cn.com.chengang.sms.model.Grade grade){
        this.name = name;
        this.grade = grade;
    }

    public Long getId() { return this.id; }
    public void setId(Long id) { this.id = id; }

    public String getName() { return this.name; }
    public void setName(String name) { this.name = name; }

    public cn.com.chengang.sms.model.Grade getGrade(){return this.grade;}
    public void setGrade(cn.com.chengang.sms.model.Grade grade) {
        this.grade = grade;
    }

    public String toString() {
        return new ToStringBuilder(this)
            .append("id", getId())
            .toString();
    }
}
```

### 26.7.4 由数据库导出成XML映射文件：ddl2hbm.bat

ddl2hbm.bat和hbm2java.bat、class2hbm.bat在同一个目录下，它可以由数据库表得到XML映射文件，具体操作步骤如下：

(1) 双击ddl2hbm.bat，弹出一个GUI窗口，然后照下图26.16所示输入相应值。

(2) 照下图26.17所示流程操作。如果第二步单击“ tables...” 时，出现找不到JDBC的错误，请检查setenv.bat文件中set JDBC\_DRIVER这一项是否设置了正确的JDBC连接包的路径。



图26.16 设置数据库连接

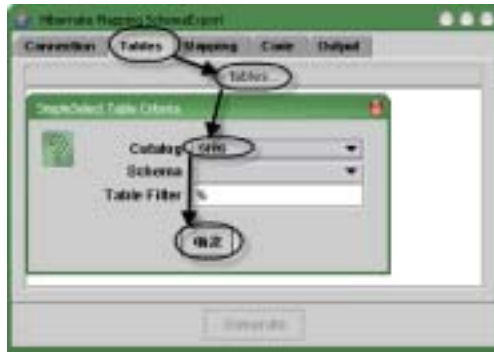


图26.17 选择数据库

(3) 最后得到sms库中的所有表，如下图26.18所示，将这些表全部选上。

(4) 单击“ Mapping ”，照下图26.19所示选择

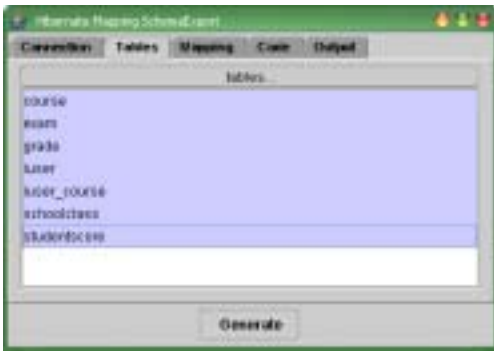


图26.18 选择数据表



图26.19 导出设置

(5) 转到“ Code ”项，在Package Name框输入“ cn.com.chengang.sms.model ”，此字符串将加入到生成的XML映射文件里。“ Base Class Name ”框让它空着。

(6) 转到Output项，接受默认的“ c:\temp ”为生成XML映射文件的存放目录，并检查此目录是否存在，不存在就手工创建它，否则ddl2hbm会报错。

(7) 单击“ Generate ”按钮，XML映射文件将被创建在C:\temp\cn\com\chengang\sms\model目录下。打开生成的XML映射文件，发现大部份设置还是比较合用的，但用户类的继承关系没有反映出来，这需要我们手工修改一下。

还有一个比ddl2hbm更强大的工具：Middlegen。本书不再介绍Middlegen，有兴趣的读者可以到“ <http://boss.bekk.no/boss/middlegen/> ”去下载。

## 26.8 本章小结

在开源日盛的今天，Eclipse作为一个开源的集成开发环境，对开源社区有着举足轻重的地位。本章是“ WEB开发篇 ”的最后一章，这四章以步步深入的方式介绍了Tomcat + JSP + JavaBean + Struts + Hibernate在Eclipse环境下的安装、设置，并用贯穿全篇的实例演示了综合开发的具体过程。