

## 第二十五章 Tomcat与Hibernate

Hibernate是一个面向Java环境的对象/关系数据库映射工具。对象/关系数据库映射（object/relational mapping（ORM））这个术语表示一种技术，用来把对象模型表示的对象映射到基于SQL的关系模型数据结构中去。

### 25.1 Hibernate技术简介

Hibernate对JDBC进行了非常轻量级的对象封装，使得Java程序员可以随心所欲的使用对象编程思维来操纵数据库。Hibernate可以应用在任何使用JDBC的场合，既可以在Java的客户端程序使用，也可以在Servlet/JSP的Web应用中使用，最具革命意义的是，Hibernate可以在应用EJB的J2EE架构中取代CMP（Container managed persistence，EJB的一种），完成数据持久化的重任。

Hibernate不仅仅管理Java类到数据库表的映射（包括Java数据类型到SQL数据类型的映射），还提供数据查询和获取数据的方法，可以大幅度减少开发时人工使用SQL和JDBC处理数据的时间。

Hibernate的目标是对于开发者通常的数据持久化相关的编程任务，解放其中的95%。对于以数据为中心的程序来说，它们往往只在数据库中使用存储过程来实现商业逻辑，Hibernate可能不是最好的解决方案；对于那些在基于Java的中间层应用中，它们实现面向对象的业务模型和商业逻辑的应用，Hibernate是最有用的。不管怎样，Hibernate一定可以帮助开发者消除或者包装那些针对特定厂商的SQL代码，并且帮助开发者把结果集从表格式的表示形式转换到一系列的对象去。

然而Hibernate不是一种强迫性的解决方案。开发者在写业务逻辑与持续性类时，不会被要求遵循许多Hibernate特定的规则和设计模式。这样，Hibernate就可以与大多数新的和现有的应用平稳地集成，而不需要对应用的其余部分作破坏性的改动。

Hibernate本身是个独立的框架，它不需要任何web server或application server的支持。在后面的几节中先简单介绍Hibernate能干什么，然后将Hibernate与Tomcat进行集成开发。

从这图25.1可以看出，Hibernate使用数据库和配置信息来为应用程序提供持久化服务（以及持久的对象）。

下面更详细地看一下Hibernate运行时体系结构。由于Hibernate非常灵活，且支持数种应用方案，所以这里只描述两种极端的情况。“轻型”的体系结构方案，要求应用程序提供自己的JDBC连接并管理自己的事务。这种方案使用了Hibernate API的最小子集，此时，Hibernate在应用程序中的作用如图25.2。



图25.1 Hibernate体系结构

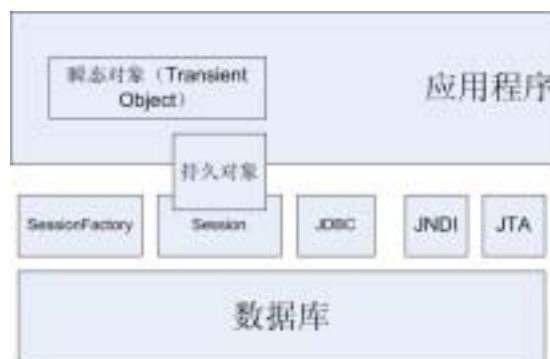


图25.2 轻型Hibernate运行体系结构

“全面解决”的体系结构方案，将应用层从底层的JDBC/JTA API中抽象出来，而让Hibernate来处理这些细节。如图25.3。



图25.3 “全面解决”的体系结构

图25.3中各个对象的定义如下：

- SessionFactory (org.hibernate.SessionFactory)：针对单个数据库映射关系经过编译后的内存镜像，它也是线程安全的（不可变）。它是生成Session的工厂，本身要用到ConnectionProvider。该对象可以在进程或集群的级别上，为那些事务之间可以重用的数据提供可选的二级缓存。
- Session(org.hibernate.Session)：表示应用程序与持久储存层之间交互操作的一个单线程对象，此对象生存期很短。它隐藏了JDBC连接，也是Transaction的工厂。会持有针对持久化对象的必选（第一级）缓存，在遍历对象图或者根据持久化标识查找对象时会用到。
- 持久的对象及其集合：带有持久化状态的、具有业务功能的单线程对象，此对象生存期很短。这些对象可以是普通的JavaBeans或者POJO（Plain Old Java Objects，就是POJOs，有时候也称作Plain Ordinary Java Objects，一个POJO很像JavaBean），比较特殊的是它们正与一个（仅仅一个）Session相关联。这个Session被关闭的同时，这些对象也会脱离持久化状态，可以被应用程序的任何层自由使用。（例如，用于跟表示层打交的数据传输对象data transfer object。）
- 瞬态（transient）以及脱管（detached）的对象及其集合：持久类的没有与Session

相关联的实例。它们可能是在被应用程序实例化后，尚未进行持久化的对象。也可能是因为实例化他们的Session已经被关闭而脱离持久化的对象。

- 事务Transaction ( org.hibernate.Transaction ) ( 可选的 ) : 应用程序用来指定原子操作单元范围的对象，它是单线程的，生存期很短。它通过抽象将应用从底层具体的JDBC、JTA以及CORBA事务隔离开。某些情况下，一个Session之内可能包含多个Transaction对象。尽管是否使用该对象是可选的，但是事务边界的开启与关闭（无论是使用底层的API还是使用Transaction对象）是必不可少的。
- ConnectionProvider ( org.hibernate.connection.ConnectionProvider ) ( 可选的 ) : 生成JDBC连接的工厂（同时也起到连接池的作用）。它通过抽象将应用从底层的Datasource或DriverManager隔离开。仅供开发者扩展/实现用，并不暴露给应用程序使用。
- TransactionFactory ( org.hibernate.TransactionFactory ) ( 可选的 ) : 生成Transaction对象实例的工厂。仅供开发者扩展/实现用，并不暴露给应用程序使用。
- 扩展接口：Hibernate提供了很多可选的扩展接口，可以通过实现它们来定制持久层的行为。

在一个“轻型”的体系结构中，应用程序可能绕过 Transaction/TransactionFactory 以及 ConnectionProvider 等API直接跟JTA或JDBC打交道。

## 25.4 Hibernate配置

从上一节中可以了解到Hibernate在数据库应用中的作用。Hibernate使用数据库和配置信息来为应用程序提供持久化服务，它实现与数据库连接的过程如图25.4。



图25.4 Hibernate获得数据库连接

由于Hibernate是为了能在各种不同环境下工作而设计的，因此存在着大量的配置参数，Hibernate得到配置信息存在很多种方式。

- 编程的配置方式。
- 传一个java.util.Properties实例给 Configuration.setProperties()。
- 将hibernate.properties放置在类路径（classpath）的根目录下（root directory）。
- 通过java -Dproperty=value来设置系统（System）属性。
- 在hibernate.cfg.xml中加入元素<property>。

其中使用hibernate.properties文件是最简单的方式，下面结合这几种方式讲述Hibernate是如何实现数据库连接的。

一个org.hibernate.cfg.Configuration实例代表了一个应用程序中Java类型到SQL数据库映射的完整集合。Configuration被用来构建一个（不可变的）SessionFactory。

### 25.4.1 可编程的配置方式

可编程的配置方式并不是最好的方式，但是，通过了解可编程的配置方式，可以对Hibernate的配置过程的实现有更好的理解，而且通过这种方式来了解Hibernate的一些配置属性的作用和使用方法。

#### 1. 获得配置属性

编程的配置方式是直接实例化Configuration来获取一个实例，并为它指定XML映射定义文件。如果映射定义文件在类路径（classpath）中，就使用addResource()方法把XML映射定义文件的信息加载从而进行Hibernate的配置。

```
Configuration cfg = new Configuration().addResource("Item.hbm.xml")
    .addResource("Bid.hbm.xml");
```

一种更好的方式是指定被映射的类，让Hibernate寻找映射定义文件：

```
Configuration cfg = new Configuration()
    .addClass(org.hibernate.auction.Item.class)
    .addClass(org.hibernate.auction.Bid.class);
```

Hibernate 将会在类路径中寻找名字为 /org/hibernate/auction/Item.hbm.xml 和 /org/hibernate/auction/Bid.hbm.xml映射定义文件。这种方式消除了任何对文件名的硬编码

Configuration也允许开发者编码指定配置属性：

```
Configuration cfg = new Configuration()
    .addClass(org.hibernate.auction.Item.class)
    .addClass(org.hibernate.auction.Bid.class)
    .setProperty("hibernate.dialect",
        "org.hibernate.dialect.MySQLInnoDBDialect")
    .setProperty("hibernate.connection.datasource",
        "java:comp/env/jdbc/test")
    .setProperty("hibernate.order_updates", "true");
```

Configuration实例是一个启动期间的对象，一旦它完成创建SessionFactory的任务，它就被丢弃了。

#### 2. 获得SessionFactory

当所有映射定义被Configuration解析后，应用程序必须获得一个用于构造Session实例的工厂。这个工厂将被应用程序的所有线程共享：

```
SessionFactory sessions = cfg.buildSessionFactory();
```

Hibernate允许应用程序创建多个SessionFactory实例。这对使用多个数据库的应用来说很有用。

#### 3. JDBC连接

通常希望SessionFactory来创建和缓存（数据库连接池）JDBC连接。如果采用这种方式，只需要如下所示那样，打开一个Session：

```
Session session = sessions.openSession(); // 打开一个新的 Session
```

一旦需要进行数据访问时，就会从连接池获得一个JDBC连接。为了使这种方式工作起来，需要向Hibernate传递一些JDBC连接的属性。所有Hibernate属性的名字和语义都在org.hibernate.cfg.Environment中定义。下面将描述JDBC连接配置中最重要的几项设置。

如果设置如下属性，Hibernate将使用java.sql.DriverManager来获得（和缓存）JDBC连接

表25.1 Hibernate JDBC属性

属性名	用途
-----	----

hibernate.connection.driver_class	JDBC驱动类
hibernate.connection.url	JDBC URL
hibernate.connection.username	数据库用户
hibernate.connection.password	数据库用户密码
hibernate.connection.pool_size	连接池容量上限数目

但Hibernate自带的连接池算法相当不成熟。它只是为了让初学者快些上手，不适合用于产品系统或性能测试中。出于最佳性能和稳定性考虑应该使用第三方的连接池。

连接池的特定设置替换hibernate.connection.pool\_size。这将关闭Hibernate自带的连接池。例如，可以考虑使用Tomcat的连接池。

为了能在应用程序服务器（application server）中使用Hibernate，应当将Hibernate配置成注册在JNDI中的Datasource处获得连接，至少需要设置下列属性（如表 25.2）中的一个：

表25.2 Hibernate数据源属性

属性名	用途
hibernate.connection.datasource	数据源JNDI名字
hibernate.jndi.url	JNDI提供者的URL（可选）
hibernate.jndi.class	JNDI InitialContextFactory类（可选）
hibernate.connection.username	数据库用户（可选）
hibernate.connection.password	数据库用户密码（可选）

下面是一个使用应用程序服务器JNDI数据源的hibernate.properties样例文件：

```
#数据源JNDI名字
hibernate.connection.datasource = java:/comp/env/jdbc/test
#工厂类
hibernate.transaction.factory_class = \
org.hibernate.transaction.JTATransactionFactory
hibernate.transaction.manager_lookup_class = \
org.hibernate.transaction.JBossTransactionManagerLookup
hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
```

从JNDI数据源获得的JDBC连接将自动参与应用程序服务器中容器管理的事务中去。任何连接配置属性的属性名要以"hibernate.connection"前缀开头。例如，可以使用hibernate.connection.charset来指定charset。

通过实现org.hibernate.connection.ConnectionProvider接口，可以定义属于自己的获得JDBC连接的插件策略。通过设置hibernate.connection.provider\_class，可以选择一个自定义的实现。

注意：除了以上属性外还有大量属性能用来控制 Hibernate 在运行期的行为，它们都是可选的，并拥有适当的默认值，这里不详细介绍。

## 25.4.2 XML 配置文件方式

另一个配置方法是在hibernate.cfg.xml文件中指定一套完整的配置。这个文件可以当成hibernate.properties的替代，它使用标准的XML语法，是比较受欢迎的一种方式。

下面是一个配置文件的例子：

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
```

```

    <session-factory>

        <!-- 数据库连接设置 -->
        <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
name="connection.url">jdbc:mysql://localhost/hibernate_test</property>
        <property name="connection.username">root</property>
        <property name="connection.password">ict</property>

        <!-- JDBC 连接池（使用内置的连接池） -->
        <property name="connection.pool_size">1</property>

        <!-- SQL dialect -->
        <property
name="dialect">org.hibernate.dialect.MySQLDialect</property>

        <!-- 显示所有的SQL语句到标准输出 -->
        <property name="show_sql">true</property>

        <!-- 启动时删除并新建数据库schema -->
        <property name="hbm2ddl.auto">create</property>

        <mapping resource="Student.hbm.xml" />

    </session-factory>

</hibernate-configuration>

```

注意：在 XML 配置文件中使用的属性与前面介绍的属性名字是基本一样的，唯一的差别就在于 XML 配置文件中的属性名省略了 hibernate 前缀。

使用XML配置，使得启动Hibernate变的非常简单，而且还具有XML语法的优势。如下所示，一行代码就可以获得一个SessionFactory实例：

```
SessionFactory sf = new Configuration().configure().buildSessionFactory();
```

另外可以使用如下代码来添加一个不同的XML配置文件

```
SessionFactory sf = new Configuration()
    .configure("catdb.cfg.xml").buildSessionFactory();
```

## 25.5 对象/关系数据库映射基础

对象和关系数据库之间的映射通常是用一个XML文档来定义的。这个映射文档被设计为容易阅读的，并且可以手工修改。映射语言是以Java为中心，这意味着映射文档是按照持久化类的定义来创建的，而不是表的定义。

一个映射文件中可以包含很多元素，其中有些在简单的开发过程中可以不用考虑（如果需要更深入的了解相关内容，可以参照Hibernate的帮助文档），下面是一个映射文件的定义，之后会通过分析这个文件的一部分来阐述对象/关系数据库映射是如何实现的。

```
<?xml version="1.0"?>
```

```

<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="eg">
<!-- 下面定义了持久化类的类名和这个类中的属性 -->
    <class name="cn.ac.ict.Cat" table="cats">
        <id name="id">
            <generator class="native"/>
        </id>

        <property name="weight"/>

        <property name="birthdate"
            type="date"
            not-null="true"
            update="false"/>

        <property name="color"
            type="eg.types.ColorUserType"
            not-null="true"
            update="false"/>

        <property name="sex"
            not-null="true"
            update="false"/>

        <property name="litterId"
            column="litterId"
            update="false"/>

    </class>
</hibernate-mapping>

```

### 25.5.1 class元素

使用class元素定义一个持久化类，下面是class元素的语法定义。

```

<class
    name="ClassName"
    table="tableName"
    discriminator-value="discriminator_value"
    mutable="true|false"
    schema="owner"
    catalog="catalog"
    proxy="ProxyInterface"
    dynamic-update="true|false"
    dynamic-insert="true|false"
    select-before-update="true|false"
    polymorphism="implicit|explicit"
    where="arbitrary sql where condition"
    persister="PersisterClass"

```

```

batch-size="N"
optimistic-lock="none|version|dirty|all"
lazy="true|false"
entity-name="EntityName"
check="arbitrary sql check condition"
rowid="rowid"
subselect="SQL expression"
abstract="true|false"
entity-name="EntityName"
node="element-name"

```

/>

所有属性都是可选的，下面简单介绍几个属性的作用和使用方法：

- name (可选)：持久化类 (或者接口) 的完整类名 (包括包名)。如果这个属性不存在，Hibernate将假定这是一个非POJO的实体映射。
- table (可选 - 默认是类的非全限定名)：对应的数据库表名。

如例中可以看到持久化类的名字是cn.ac.ict.Cat，在数据库中对应的表名是cats。

注意：类 (或者接口) 的 Java 完整类名是指包含类或接口所在包的名字，如例中 Cat 类在 cn.ac.ict 包中，则其 Java 全限定名是 cn.ac.ict.Cat，非全限定名是 Cat。

## 25.5.2 id元素

被映射的类必须定义对应数据库表的主键字段。大多数类有一个JavaBeans风格的属性，为每一个实例包含唯一的标识。<id>元素定义了该属性到数据库表的主键字段的映射。id元素的语法定义如下。

```

<id
  name="propertyName"
  type="typename"
  column="column_name"
  unsaved-value="null|any|none|undefined|id_value"
  access="field|property|ClassName"
  node="element-name|@attribute-name|element/@attribute|.">
  <generator class="generatorClass"/>
</id>

```

其中：

- name (可选)：标识属性的名字。
- type (可选)：标识Hibernate类型的名字。
- column (可选 - 默认为属性名)：主键字段的名称。

注意：如果 name 属性不存在，会认为这个类没有标识属性。

### 1. Generator子元素

可选的<generator>子元素是一个Java类的名字，用来为该持久化类的实例生成唯一的标识。如果这个生成器实例需要某些配置值或者初始化参数，用<param>元素来传递。例如：

```

<id name="id" type="long" column="cat_id">
  <generator class="org.hibernate.id.TableHiLoGenerator">
    <param name="table">uid_table</param>
    <param name="column">next_hi_value_column</param>
  </generator>

```

</id>

所有的生成器都实现net.sf.hibernate.id.IdentifierGenerator接口。这是一个非常简单的接口；某些应用程序可以选择提供自己特定的实现。当然，Hibernate提供了很多内置的实现。下面是一些内置生成器的快捷名字及其描述。

- Increment：用于为long, short或者int类型生成唯一标识。只有在没有其他进程往同一张表中插入数据时才能使用。在集群下不要使用。
- Identity：对DB2、MySQL、MS SQL Server、Sybase和Hypersonic SQL的内置标识字段提供支持。返回的标识符是long, short 或者int类型的。
- Sequence：在DB2、PostgreSQL、Oracle、SAP DB McKoi中使用序列（sequence），而在Interbase中使用生成器（generator）。返回的标识符是long, short或者int类型的。
- Hilo：使用一个高/低位算法高效的生成long、short 或者 int类型的标识符。给定一个表和字段（默认分别是hibernate\_unique\_key 和next\_hi）作为高位值的来源。高/低位算法生成的标识符只在一个特定的数据库中是唯一的。
- Seqhilo：使用一个高/低位算法来高效的生成long、short 或者 int类型的标识符，给定一个数据库序列（sequence）的名字。
- Uuid：用一个128-bit的UUID算法生成字符串类型的标识符，这在一个网络中是唯一的（使用了IP地址）。UUID被编码为一个32位16进制数字的字符串。
- Guid：在MS SQL Server 和 MySQL 中使用数据库生成的GUID字符串。
- Native：根据底层数据库的能力选择identity, sequence 或者hilo中的一个。
- assigned：让应用程序在save()之前为对象分配一个标识符。这是<generator>元素没有指定时的默认生成策略。
- select：通过数据库触发器选择一些唯一主键的行并返回主键值来分配一个主键。
- foreign：使用另外一个相关联的对象的标识符。通常和<one-to-one>联合起来使用。

注意：对于内部支持标识字段的数据库（DB2、MySQL、Sybase、MS SQL），可以使用 identity 关键字生成。对于内部支持序列的数据库（DB2、Oracle、PostgreSQL、Interbase、McKoi、SAP DB），可以使用 sequence 风格的关键字生成。

在本节开头的例子中使用了名为native的生成器，对于跨平台开发，native策略会从identity, sequence 和hilo中进行选择，选择哪一个，取决于底层数据库的支持能力。

### 25.5.3 property元素

<property>元素为类定义一个持久化的JavaBean风格的属性。<property>元素的语法定义如下。

```
<property
  name="propertyName"
  column="column_name"
  type="typename"
  update="true|false"
  insert="true|false"
  formula="arbitrary SQL expression"
  access="field|property|ClassName"
  lazy="true|false"
  unique="true|false"
  not-null="true|false"
  optimistic-lock="true|false"
```

- ```
node="element-name|@attribute-name|element/@attribute|."
```
- </>
- name：属性的名字，以小写字母开头，与持久化类中的属性名字一致
  - column（可选 - 默认为属性名字）：对应的数据库字段名。也可以通过嵌套的 <column>元素指定。
  - type（可选）：一个Hibernate类型的名字。

Hibernate类型可以是如下几种：

- Hibernate基础类型之一（比如：integer、string、character、date、timestamp、float、binary、serializable、object和blob）。
- 一个Java类的名字，这个类属于一种默认基础类型（比如：int、float、char、java.lang.String、java.util.Date、java.lang.Integer和java.sql.Clob）。
- （一个可以序列化的Java类的名字。
- 一个自定义类型的类的名字。（比如：cn.ac.ict.type.MyCustomType）；

如果不指定类型，Hibernate会使用反射来得到这个名字的属性，以此来猜测正确的Hibernate类型。Hibernate会按照规则2，3，4的顺序对属性读取器（getter方法）的返回类进行解释。

注意：在某些情况下仍然需要 type 属性。比如，为了区别 Hibernate.DATE 和 Hibernate.TIMESTAMP，或者为了指定一个自定义类型。

## 25.2 第一个可持久化类

Hibernate往往与其他的开发环境结合使用，但是对于初学者来说，首先要知道Hibernate是一个怎样的工具和能够完成什么样的工作，在下面的介绍中，以一个简单的例子来介绍Hibernate，让读者对Hibernate能够干什么有一个最直观的认识，这个例子不涉及除Ant外其他的开发平台或技术，如图读者需要了解Ant的相关知识可以参考本书《使用Ant管理Web应用》一章。

在这个例子中，需要使用MySQL数据库，所以要有MySQL的JDBC Driver，笔者使用的是MySQL4.1，驱动程序为mysql-connector-java-3.0.16-ga-bin.jar可从<http://www.mysql.com>上免费下载。

### 25.2.1 新建项目并配置环境

(1) 新建一个项目目录，例如：E:\HibernateWork\My2stHibernate，以后称此目录为项目目录。

(2) 在项目目录下新建3个子目录src、classes、lib，然后在lib目录下新建2个目录hibernate和db，此时的目录结构如下：

```
E:\HibernateWork\My2stHibernate\  
E:\HibernateWork\My2stHibernate\src  
E:\HibernateWork\My2stHibernate\classes  
E:\HibernateWork\My2stHibernate\lib  
E:\HibernateWork\My2stHibernate\lib\hibernate  
E:\HibernateWork\My2stHibernate\lib\db
```

其中src目录用于存放源文件，lib目录用于存放项目开发需要的库文件，下面又分为两个子目录，其中hibernate存放hibernate的库文件，db存放数据库驱动程序，这样做的好处就是把不同的库文件分开存放，方便管理。

(3) 把Hibernate包所在目录下的hibernate3.jar包拷贝到<Web应用目录>\lib\hibernate目录下,同时还需要把Hibernate包所在目录下lib目录中的如下包文件。

- > antlr-2.7.5H3.jar
- > asm.jar
- > asm-attrs.jar
- > cglib-2.1.jar
- > commons-collections-2.1.1.jar
- > commons-logging-1.0.4.jar
- > dom4j-1.6.jar
- > ehcache-1.1.jar
- > jta.jar
- > log4j-1.2.9.jar

拷贝到<Web应用目录>\lib\hibernate目录下,这个是Hibernate运行所需要的最小库文件集合(可以在Hibernate分发版本的lib/目录下查看README.txt,以获取更多关于所需和可选的第三方库文件信息),当然最简单的办法就是把所有包文件都拷贝到<项目目录>\lib\hibernate目录下,把数据库驱动程序(这里是mysql-connector-java-3.0.16-ga-bin.jar)拷贝到<Web应用目录>\lib\db下。

(4) 创建名为hibernate的MySQL数据库,并建立数据表StudentInfo,使用的SQL语句以及效果如下所示。

```
mysql>create database hibernate;
Query OK, 1 rows affected (0.11 sec)
mysql>create table StudentInfo(sid int(20) not null primary key ,sname
varchar(20) not null,LOGIN_DATE DATE);
Query OK, 0 rows affected (0.11 sec)
```

## 25.2.2 编写持久化类

在本例中第一个持久化类是一个简单的JavaBean class,带有一些简单的属性,下面是该文件的代码:

```
import java.util.Date;

public class Student{
    private Long sid;
    private String sname;
    private Date logindate;

    // sid属性的获取和设置方法
    public Long getSid()
    {
        return sid;
    }
    public void setSid(Long newsid)
    {
        sid=newsid;
    }
    // sname属性的获取和设置方法
    public String getSname()
    {
        return sname;
    }
}
```

```

    }
    public void setName(String newsname)
    {
        sname=newsname;
    }
    // logindate属性的获取和设置方法
    public Date getLogindate()
    {
        return logindate;
    }
    public void setLogindate(Date newlogindate)
    {
        logindate=newlogindate;
    }
}

```

这个class对属性的存取方法使用标准的JavaBean命名约定，同时把内部字段隐藏起来，推荐这样使用，但并不是必须这样做，因为Hibernate也可以直接访问这些字段，而使用访问方法的好处是提供了程序重构时候的健壮性。

sid属性为一个Student实例提供标识属性的值，如果希望使用Hibernate的所有特性，那么所有的持久性实体类（这里也包括一些次要依赖类）都需要一个标识属性。而事实上，大多数应用程序（特别是web应用程序）都需要识别特定的对象，所以开发者应该考虑使用标识属性而不是把它当作一种限制。然而，通常不会直接操作一个对象的标识符，因此标识符的setter方法应该被声明为私有的。这样当一个对象被保存的时候，只有Hibernate可以为它分配标识符。Hibernate可以直接访问被声明为public、private和protected等不同级别访问控制的方法和字段。所以选择哪种方式来访问属性是完全取决于开发者，可以使选择与程序设计相吻合。

所有的持久类都要求有无参的构造器。因为Hibernate必须要使用Java反射机制来实例化对象。构造器的访问控制可以是私有的，然而当生成运行时代理的时候将要求使用至少是package级别的访问控制，这样在没有字节码编入的情况下，从持久化类里获取数据会更有效率一些。

把这个Java源代码文件放到开发目录下面一个叫做src的目录里。

### 25.2.3 编写映射文件

Hibernate映射文件（mapping file）告诉Hibernate怎样去加载和存储程序的持久化类的对象。映射文件告诉Hibernate它应该访问数据库里面的哪个表（table）和应该使用表里面的哪些字段（column）。映射文件的基本结构是这样的：

```

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
[... ]
</hibernate-mapping>

```

在两个hibernate-mapping标签中间，包含了一个class元素。所有的持久性实体类都需要一个这样的映射，来映射到程序使用的SQL数据库。

```

<hibernate-mapping>
  <class name="Student" table="StudentInfo">

    </class>
  </hibernate-mapping>

```

下面将讲述有关唯一标识属性的映射。另外，如果不希望去考虑怎样产生这个标识属性，开发者将配置Hibernate的标识符生成策略来产生代用主键。

```

<hibernate-mapping>
  <class name="Student" table="StudentInfo">
    <id name="sid" column="Student_ID">
      <generator class="increment"/>
    </id>
  </class>
</hibernate-mapping>

```

sid元素是标识属性的声明，name="sid"声明了Java属性的名字（Hibernate将使用getSid()和setSid()来访问它）。column参数则告诉Hibernate使用StudentInfo表的哪个字段作为主键。嵌套的generator元素指定了标识符的生成策略，在这里是使用increment，这个是非常简单的在内存中直接生成数字的方法，多数用于测试（或教程）中。Hibernate同时也支持使用数据库生成，全局唯一性和应用程序指定（或者自己为任何已有策略所写的扩展）这些方式来生成标识符。

最后还必须在映射文件里面包括需要持久化属性的声明。缺省的情况下，类里面的属性都被视为非持久化的：

```

<hibernate-mapping>

  <class name="Student" table="StudentInfo">
    <id name="sid" column="Student_ID">
      <generator class="increment"/>
    </id>
    <property name="logindate" type="timestamp" column="LOGIN_DATE"/>
    <property name="sname"/>
  </class>

</hibernate-mapping>

```

property元素和sid元素类似，它的name参数告诉Hibernate使用哪个getter和setter方法。为什么logindate属性的映射包括column参数，但是sname却没有？

当没有设定column参数的时候，Hibernate缺省使用属性名作为字段名。对于sname，这样就可以了。

下一件事情是logindate属性的type参数。开发者声明并使用在映射文件里面的type，既不是Java data type，同时也不是SQL database type。这些类型被称作Hibernate mapping types，它们把数据类型从Java转换到SQL data types。如果映射的参数没有设置的话，Hibernate也将尝试去确定正确的类型转换和它的映射类型。在某些情况下这个自动检测（在Java class上使用反射机制）不会产生期待或者需要的缺省值。

这个例子中的logindate属性，Hibernate无法知道这个属性应该被映射成下面这些类型中的哪一个：SQL date, timestamp, time。通过声明属性映射timestamp来表示开发者希望保存所有的关于日期和时间的信息。

这个映射文件（mapping file）应该被保存为Student.hbm.xml，和Student.Java源文件放在同一个目录下。映射文件的名称可以是任意的，然而hbm.xml已经成为Hibernate开发者社

区的习惯性约定。

## 25.2.4 Hibernate配置

下面开始配置Hibernate。Hibernate是程序里连接数据库的应用层,它需要连接用的信息。连接是通过一个也由开发配置的JDBC连接池来获取的。

为了配置Hibernate,可以使用一个简单的hibernate.properties文件,或者一个稍微复杂的hibernate.cfg.xml,甚至可以完全使用程序来配置Hibernate。本例中采用XML文件的方式,下面是hibernate.cfg.xml文件的内容:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>

        <!-- Database connection settings -->
        <!-- 数据库驱动程序-->
        <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <!-- 数据库连接URL-->
        <property name="connection.url">jdbc:mysql://localhost/hibernate
</property>
        <!-- 数据库用户名-->
        <property name="connection.username">root</property>
        <!-- 数据库登陆密码-->
        <property name="connection.password">ict</property>

        <!-- 使用内置的JDBC 连接池 -->
        <property name="connection.pool_size">1</property>

        <!-- SQL dialect -->
        <property
name="dialect">org.hibernate.dialect.MySQLDialect</property>

        <!-- 把所有执行的SQL语句输出到标准输出 -->
        <property name="show_sql">>true</property>

        <!-- 在启动时删除并新建数据库schema -->
        <property name="hbm2ddl.auto">create</property>
        <!-- 使用的映射文件-->
        <mapping resource="Student.hbm.xml" />

    </session-factory>

</hibernate-configuration>
```

这个hibernate.cfg.xml文件也要保存在src目录下。

### 25.2.5 编写应用文件

在本例中使用了一个简单的Application，来完成调用Hibernate的工作，下面是StudentManager.java的源代码。

```
import org.hibernate.Transaction;
import org.hibernate.Session;

import java.util.Date;

public class StudentManager{
    public static void main(String args[]){
        StudentManager sm = new StudentManager();
        if(args[0].equals("store")){
            sm.createAndStoreStudent("Rambler",new Date());
            HibernateUtil.sessionFactory.close();
        }
    }
    // createAndStoreStudent方法根据学生的名字和注册日期构建一个持久化对象，并
    //使用Hibernate进行持久化
    private void createAndStoreStudent(String sname,Date loginDate){
        Session session = HibernateUtil.currentSession();
        Transaction tx = session.beginTransaction();

        Student newS = new Student();
        newS.setSname(sname);
        newS.setLogindate(loginDate);
        session.save(newS);
        tx.commit();
        HibernateUtil.closeSession();
    }
}
```

在这个类中的一个主要的方法是createAndStoreStudent，在这个方法中获得了Hibernate的Session和Transaction，然后创建了一个持久化类，并把这个类保存到数据库中，最后是关闭所有不再需要的资源。

### 25.2.6 Ant编译

下面是Ant需要的build.xml文件的内容：

```
<project name="My2stHibernate" default="compile">

    <property name="sourcedir" value="${basedir}/src"/>
    <property name="targetdir" value="${basedir}/bin"/>
    <property name="librarydir" value="${basedir}/lib"/>

    <path id="libraries">
        <fileset dir="${librarydir}">
            <include name="**/*.jar"/>
        </fileset>
    </path>
</project>
```

```

    </path>
    <!-- 删除临时目录 ---->
    <target name="clean">
        <delete dir="${targetdir}"/>
        <mkdir dir="${targetdir}"/>
    </target>
    <!-- 编译文件 ---->
    <target name="compile" depends="clean, copy-resources">
        <javac srcdir="${sourcedir}"
            destdir="${targetdir}"
            classpathref="libraries"/>
    </target>
    <!-- 把需要的资源拷贝到合适的位置 ---->
    <target name="copy-resources">
        <copy todir="${targetdir}">
            <fileset dir="${sourcedir}">
                <exclude name="**/*.java"/>
            </fileset>
        </copy>
    </target>
    <!-- 执行这个应用 ---->
    <target name="run" depends="compile">
        <java fork="true" classname="StudentManager" classpathref="libraries">
            <classpath path="${targetdir}"/>
            <arg value="${action}"/>
        </java>
    </target>
</project>

```

### 25.2.7 运行程序

按照上面的顺序准备好所有的文件后，这时整个程序的结构如图25.5。在开发目录下运行ant命令：ant run -Daction=store，可以得到如下输出：

```

Buildfile: build.xml

copy-resources:
    [copy] Copying 4 files to E:\HibernateWork\My2stHibernate\bin

compile:
    [javac] Compiling 3 source file to E:\HibernateWork\My2stHibernate\bin
run:
    [java] log4j:WARN No appenders could be found for logger
(net.sf.hibernate.cfg.Environment).
    [java] log4j:WARN Please initialize the log4j system properly.
    [java] Hibernate: insert into StudentInfo(LOGIN_DATE, sname, Student_ID)
values (?, ?, ?)

BUILD SUCCESSFUL
Total time: 5 second
这是Hibernate执行的INSERT命令，问号代表JDBC的待绑定参数。然后可以在MySQL

```

中查看结果，如图25.6所示。

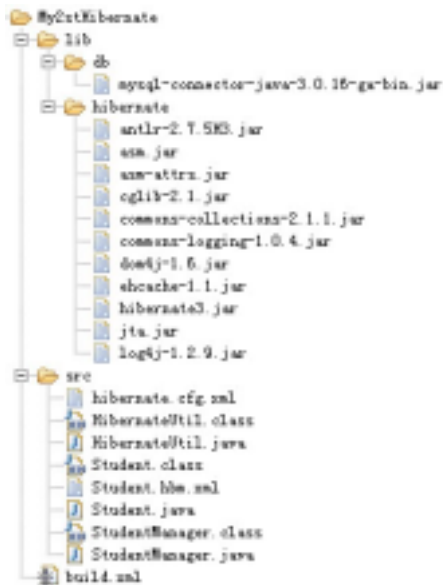


图25.5 应用的程序结构

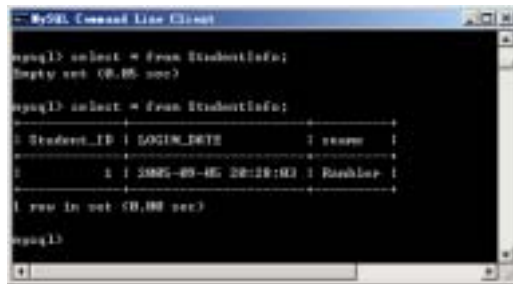


图25.6 第一个持久化类执行结果

数据库中多了一条记录，可是在程序中却没有一条SQL语句，这就是使用Hibernate的效果了。

## 25.3 Tomcat与Hibernate安装配置

本节讲述如何在Apache Tomcat servlet容器中为web应用程序配置使用Hibernate 3.0（使用Tomcat 5.0版本）。Hibernate在大多数主流J2EE应用服务器的运行环境中都可以工作良好，甚至也可以在独立Java应用程序中使用。使用的示例数据库系统是MySQL 4.1，只需要修改Hibernate SQL语言配置与连接属性，就可以很容易的支持其他数据库了。

### 25.3.1 新建Tomcat Web Context并配置环境

为了演示Tomcat与Hibernate的整合使用，新建立一个新的Tomcat Context，为TomcatHibernate，具体建立方式参考本章第二章。

拷贝所有需要的库文件到Tomcat安装目录中。在本例中，使用一个独立的web Context配置（TomcatHibernate）。确认全局库文件（TOMCAT/common/lib）和本web应用程序上下文的路径（对于jar来说是TomcatHibernate/WEB-INF/lib，对于class文件来说是TomcatHibernate/WEB-INF/classes）能够被类装载器检测到。这里把这两个类装载器级别分别称做全局类路径（global classpath）和上下文类路径（context classpath）。

把需要的库文件拷贝到两个类路径去：

- 把数据库的JDBC驱动程序拷贝到全局类路径中，这是Tomcat捆绑的DBCP连接池所需要的。Hibernate使用JDBC连接数据库方式执行SQL语句，所以开发者要么提供外部连接池中的连接给Hibernate，或者配置Hibernate自带的连接池（C3PO，Proxool）。对于本例来说，把mysql-connector-java-3.0.16-ga-bin.jar库文件（支持MySQL和JDK 1.4以上）到全局类装载路径下即可。如果希望使用其他的数据库，

- 拷贝其相应的JDBC驱动文件)。
- 不要拷贝任何其他东西到Tomcat的全局类路径下,否则可能在使用其他一些工具上遇到麻烦,比如log4j、commons-logging等等。让每个web应用程序使用自己的上下文类路径,就是说把自己需要的类库拷贝到WEB-INF/lib下去,把configuration/property等配置文件拷贝到WEB-INF/classes下面去。这两个目录都是当前程序缺省的上下文类路径。
  - Hibernate本身打包成一个JAR类库。将hibernate3.jar文件拷贝到程序的上下文类路径下,和应用程序的其他库文件放一起。在运行时,Hibernate还需要一些第三方类库,它们在Hibernate发行包的lib/目录下。参见表25.1“Hibernate 第三方类库”。把所需要的第三方库文件也拷贝到上下文类路径下。

表25.3 Hibernate第三方类库

类库	描述
antlr (必需)	Hibernate使用ANTLR来产生查询分析器,这个类库在运行环境下时也是必需的。
dom4j (必需)	Hibernate使用dom4j解析XML配置文件和XML映射元文件。
CGLIB, asm (必需)	Hibernate在运行时使用这个代码生成库增强类(与Java反射机制联合使用)。
CommonsCollections, Commons Logging (必需)	Hibernate使用Apache Jakarta Commons项目提供的多个工具类库。
EHCache (必需)	Hibernate可以使用不同cache缓存工具作为二级缓存。EHCache是缺省的cache缓存工具。
Log4j (可选)	Hibernate使用Commons Logging API,它也可以依次使用Log4j作为底层实施log的机制。如果上下文类目录中存在Log4j库,则Commons Logging使用Log4j和并它在上下文类路径中寻找的log4j.properties文件。可以使用在Hibernate发行包中包含的那个示例Log4j的配置文件。这样,把log4j.jar和它的配置文件(位于src/目录中)拷贝到程序的上下文类路径下,就可以在后台看到底程序如何运行的。

### 25.3.2 配置数据库连接池

配置在Tomcat和Hibernate中共用的数据库连接池。就是说Tomcat会提供经过池处理的JDBC连接(用它内置的DBCP连接池),Hibernate通过JNDI方式来请求获得JDBC连接。作为替代方案,也可以让Hibernate自行管理连接池。Tomcat把连接池绑定到JNDI,配置好数据源后, Tomcat的配置文件TomcatHibernate.xml如下:

```
<Context path="/TomcatHibernate"
docBase="E:\PublishBook\TomcatHibernate">
  <Resource name="jdbc/hibernate" scope="Shareable"
type="javax.sql.DataSource"/>
  <ResourceParams name="jdbc/hibernate">
    <parameter>
      <name>factory</name>
      <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
    </parameter>

    <!-- 数据库连接池信息 -->
    <parameter>
      <name>url</name>
      <value>jdbc:mysql://localhost/hibernate</value>
    </parameter>
  </ResourceParams>
</Context>
```

```

</parameter>
<!-- 数据库的JDBC驱动程序 ---->
<parameter>
  <name>driverClassName</name>
  <value>com.mysql.jdbc.Driver</value>
</parameter>
<!-- 下面是数据库的用户名和密码 ---->
<parameter>
  <name>username</name>
  <value>root</value>
</parameter>
<parameter>
  <name>password</name>
  <value>ict</value>
</parameter>

<!-- DBCP 连接池选项 -->
<parameter>
  <name>maxWait</name>
  <value>3000</value>
</parameter>
<parameter>
  <name>maxIdle</name>
  <value>100</value>
</parameter>
<parameter>
  <name>maxActive</name>
  <value>10</value>
</parameter>
</ResourceParams>
</Context>

```

在这里定义了名为jdbc/hibernate的数据源，它的JNDI名也为jdbc/hibernate；另外要使这个数据源能正常工作，还需要在MySQL中建立名为hibernate的数据库，数据库中有一个Cat数据表，其中表中的字段及其描述如表25.2。

表25.4 Cat数据表中的字段及其描述

字段	类型	描述
cat_id	char (32)	not null 、主键
name	varchar(16)	not null
sex	Char(1)	
weight	real	

在这个例子中要配置的Context是TomcatHibernate，这个Web应用的主目录是E:\PublishBook\TomcatHibernate（对于读者Web应用的位置要相应的在上面这个文件中修改）

Tomcat现在通过JNDI的方式：java:comp/env/jdbc/tomcathibernate来提供连接。如果遇到JDBC驱动所报的exception出错信息，应该在有Hibernate的环境下，先测试JDBC连接池本身是否配置正确。Tomcat和JDBC的配置方法可以在本书第七章中找到。

下面是一个用于测试这个连接池是否正确的一个简单的JSP页面，datasourceTest.jsp的代码如下：

```

<%@ page language="java" pageEncoding="GB2312" %>
<%@ page import="java.sql.*, javax.sql.*, javax.naming.*"%>
<%

    Connection conn=null;
    String url = "jdbc:mysql://localhost:3306/shopdb";
    String user = "root";
    String passw = "ict";
    try{
        InitialContext ctx = new InitialContext();
        DataSource ds = (DataSource)
ctx.lookup("java:comp/env/jdbc/hibernate");
        conn = ds.getConnection();
        if(conn!=null){
            out.println("数据源jdbc/hibernate配置正确!");
        }
    }catch(Exception ex){
        out.println("数据库驱动加载出现错误!" +ex);
    }
}
%>

```

配置好后，把这个文件拷贝到Web应用的根目录下（这里是E:\PublishBook\TomcatHibernate），然后在浏览器地址栏中输入如下地址：

http://localhost:8080/TomcatHibernate/datasourceTest.jsp

这时的页面显示如图25.7。



图25.7 数据源配置成功测试

### 25.3.3 配置Hibernate

在本例中使用基于XML格式的Hibernate配置文件让Hibernate知道如何获得JDBC连接。使用properties文件的进行配置也可以，但缺少一些XML语法的特性。这个XML配置文件必须放在上下文类路径（WEB-INF/classes）下面，命名为hibernate.cfg.xml（固定命名）。

```

<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration
PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>

        <property

```

```

name="connection.datasource">java:comp/env/jdbc/hibernate</property>
    <property name="show_sql">false</property>
    </property>
name="dialect">org.hibernate.dialect.MySQLDialect</property>

    <!-- 映射文件 -->
    <mapping resource="Cat.hbm.xml"/>

</session-factory>

</hibernate-configuration>

```

在这个配置文件中关闭了SQL命令的log，同时告诉Hibernate使用MySQL数据库用语（Dialect），以及如何得到JDBC连接（通过Tomcat声明绑定的JNDI地址）。Dialect是必需配置的，因为不同的数据库都和“SQL标准”有一些出入。Hibernate会根据这个Dialect处理这些差异，Hibernate支持所有主流的商业和开放源代码数据库。

SessionFactory是Hibernate中的一个概念，表示对应一个数据存储源。通过创建多个XML配置文件并在你的程序中创建多个Configuration和SessionFactory对象，就可以支持多个数据库了。

在hibernate.cfg.xml中的最后一个元素声明了Cat.hbm.xml，这是一个Hibernate XML映射文件，对应于持久化类Cat。这个文件包含了把Cat POJO类映射到数据库表（或多个数据库表）的元数据。

### 25.3.4 编写持久化类

Hibernate使用简单的Java对象（Plain Old Java Objects）这种编程模型来进行持久化。一个POJO很像JavaBean，它通过getter和setter方法访问其属性，对外则隐藏了内部实现的细节（假若需要的话，Hibernate也可以直接访问其属性字段）。

```

package cn.ac.ict;

public class Cat {
    private String id;
    private String name;
    private char sex;
    private float weight;
    public Cat() {
    }

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

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

```

```

    }

    public char getSex() {
        return sex;
    }

    public void setSex(char sex) {
        this.sex = sex;
    }

    public float getWeight() {
        return weight;
    }

    public void setWeight(float weight) {
        this.weight = weight;
    }
}

```

Hibernate对属性使用的类型不加任何限制。所有的Java JDK类型和原始类型（比如String、char和Date）都可以被映射，也包括Java集合框架（Java collections framework）中的类。可以把它们映射成为值、值集合，或者与其他实体类相关联。id是一个特殊的属性，代表了这个类的数据库标识符（主键），对于类似于Cat这样的实体类建议使用。Hibernate也可以使用内部标识符，但这样会失去一些程序架构方面的灵活性。

持久化类不需要实现什么特别的接口，也不需要从一个特别的持久化根类继承下来。Hibernate也不需要任何编译期处理，比如字节码增强操作，它独立的使用Java反射机制和运行时类增强（通过CGLIB）。所以不依赖于Hibernate，就可以把POJO的类映射成为数据库表。

### 25.3.5 编写映射文件

Hibernate XML映射文件，对应于持久化类，它的作用就是把持久化类的属性和数据库表中的字段进行映射。

```

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping
    PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

    <class name="Cat" table="CAT">

        <!-- 把Hibernate使用UUID模式自动产生的32位16进制字符作为主键-->
        <id name="id" type="string" unsaved-value="null" >
            <column name="CAT_ID" sql-type="char(32)" not-null="true"/>
            <generator class="uuid.hex"/>
        </id>

        <!-- Cat的名字 -->
        <property name="name">
            <column name="NAME" length="16" not-null="true"/>
        </property>
    </class>
</hibernate-mapping>

```

```

        <property name="sex" />

        <property name="weight" />
    </class>
</hibernate-mapping>

```

每个持久化类都应该有一个标识属性，这个属性用来区分持久化对象：如果 `catA.getId().equals(catB.getId())` 结果是 `true` 的话，这两个 `Cat` 就是相同的。这个概念称为数据库标识。

Hiernate附带了几种不同的标识符生成器，用于不同的场合（包括数据库本地的顺序（sequence）生成器、hi/lo高低位标识模式和程序自己定义的标识符等）。在这里使用UUID生成器（只在测试时建议使用，如果使用数据库自己生成的整数类型的键值更好），并指定CAT表中的CAT\_ID字段（作为表的主键）存放生成的标识值。

Cat的其他属性都映射到同一个表的字段。对name属性来说，把它显式地声明映射到一个数据库字段。如果数据库schema是通过由映射声明使用Hibernate的SchemaExport工具自动生成的（作为SQL DDL指令）的话，这就特别有用。所有其他的属性都用Hibernate的默认值映射，大多数情况开发者都会这样做。

### 25.3.6 编写应用文件

现在可以开始Hibernate的Session了。Session（与jsp和sevlet中的Session不一样）是一个持久化管理器，通过它来从数据库中存取Cat。首先，要从SessionFactory中获取一个Session（Hibernate的工作单元）。

```

SessionFactory sessionFactory =
    new Configuration().configure().buildSessionFactory();

```

通过对 `configure()` 的调用来装载 `hibernate.cfg.xml` 配置文件，并初始化成一个 `Configuration` 实例。

在创建 `SessionFactory` 之前（它是不可变的），可以访问 `Configuration` 来设置其他属性（甚至修改映射的元数据）。应该在哪儿创建 `SessionFactory`，在程序中又如何访问它呢？`SessionFactory` 通常只是被初始化一次，例如通过一个 `load-on-startup` servlet 的来初始化。这意味着开发者不应该在 `serlvet` 中把它作为一个实例变量来持有，而应该放在其他地方。进一步的说，需要使用单例（Singleton）模式，开发者才能更容易的在程序中访问 `SessionFactory`。下面的方法就同时解决了两个问题：对 `SessionFactory` 的初始配置与便捷使用。编写一个辅助的类文件 `HibernateUtil.java`：

```

package cn.ac.ict;

import org.hibernate.*;
import org.hibernate.cfg.*;
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 {
                // 创建 SessionFactory
                sessionFactory = new
Configuration().configure().buildSessionFactory();
            } catch (Throwable ex) {
                //输出日志信息
                log.error("Initial SessionFactory creation failed.", ex);
                throw new ExceptionInInitializerError(ex);
            }
        }

        public static final ThreadLocal session = new ThreadLocal();

        public static Session currentSession() {
            Session s = (Session) session.get();
            // 打开一个新的会话
            if (s == null) {
                s = sessionFactory.openSession();
                session.set(s);
            }
            return s;
        }
    }
    //关闭一个Hibernate会话
    public static void closeSession() {
        Session s = (Session) session.get();
        if (s != null)
            s.close();
        session.set(null);
    }
}

```

这个类不但在它的静态初始器中使用了SessionFactory，还使用了一个ThreadLocal变量来保存Session做为当前工作线程。

SessionFactory是安全线程，可以由很多线程并发访问并获取到Sessions。单个Session不是安全线程对象，它只代表与数据库之间的一次操作。Session通过SessionFactory获得并在所有的工作完成后关闭。

在一个Session中，每个数据库操作都是在一个事务（transaction）中进行的，这样就可以隔离不同的操作（甚至包括只读操作）。使用Hibernate的Transaction API来从底层的事务策略中（本例中是JDBC事务）脱身出来。这样，开发者不需要更改任何源代码，就可以把程序部署到一个由容器管理事务的环境中去（使用JTA）。

这样可以随心所欲的多次调用HibernateUtil.currentSession()，而且每次都会得到当前线程的同一个Session。不管是在servlet代码中，或者在servlet filter中还是在HTTP结果返回之前，开发者都必须确保这个Session在数据库访问工作完成后关闭。这样做的一个好处就是可以容易的使用延迟装载（lazy initialization）。

下面是获得一个Session然后进行数据更新的一个简单JSP页面。

```

<%@ page language="java" pageEncoding="GB2312" %>
<%@ page import="org.hibernate.*,org.hibernate.cfg.*,cn.ac.ict.*"%>

```

```

<body>
<p class="style1">
<%
    if(request.getParameter("catname")!=null){
        String catname=request.getParameter("catname");
        char sex=request.getParameter("sex").trim().charAt(0);
        float weight=Float.parseFloat(request.getParameter("weight"));
        try{
            Transaction tx=null;
//根据客户的请求信息构建一个持久化对象
            Cat cat = new Cat();
            cat.setName(catname);
            cat.setSex(sex);
            cat.setWeight(weight);
            Session Hsession = HibernateUtil.currentSession();
            tx= Hsession.beginTransaction();
            Hsession.save(cat);
            tx.commit();
            out.print(cat);

        }catch(Exception e){out.print("insert error!"); }

    }
%>
</p>
<p>Add a New Cat:</p>
<form action="add.jsp" method="post" name="form1" target="_self">
    <p>CatName:
        <input name="catname" type="text" id="catname">
    </p>
    <p>Sex:
        <input name="sex" type="text" id="sex">
    </p>
    <p>Weight:
        <input name="weight" type="text" id="weight">
    </p>
    <p>
        <input type="submit" name="Submit" value="提交">
    </p>
</form>

```

### 25.3.7 发布运行Web应用

按照上面的步骤建立好所有需要的文件，并按照规定要求存放后，这个Web应用的程序结构如图25.8。

运行Tomcat，在浏览器地址栏中输入<http://localhost:8080/TomcatHibernate/add.jsp>，页面显示如图25.9。

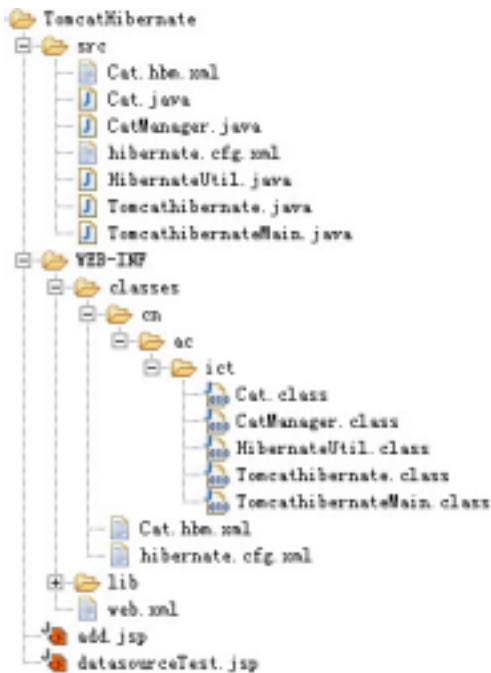


图25.8 Hibernate应用的程序结构

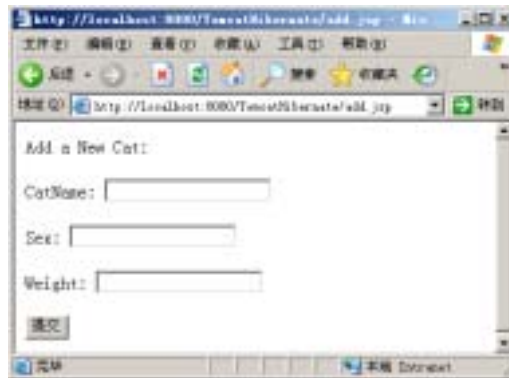


图25.9 添加一个新的Cat页面

在页面中输入如下数据：

- CatName : janes
- Sex : F
- Weight : 25.6

按提交按钮后，页面上方显示一个Cat对象的地址就表示成功了。打开MySQL数据库客户端，并是使用hibernate数据库，在其中查询cat表中的数据，此时如图25.10。

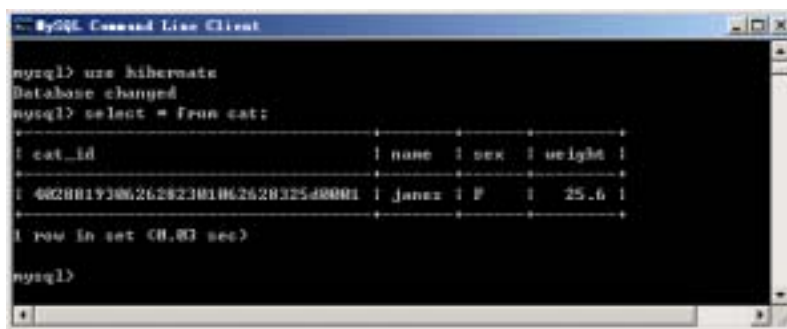


图25.10 Tomcat与Hibernate运行结果

## 25.6 相关问题

在获取Hibernate Session时出现错误如何解决？

这种情况一般是由于配置文件除了错误，可以看看两个配置文件的位置是否正确，然后再检查配置文件中是否有错别字或者错误设置的地方。

## 25.7 小结

Hibernate是一个面向Java环境的对象/关系数据库映射工具，使用它可以很容易的实现数据持久化的任务。

在本章中介绍了一些关于Hibernate的基础知识，使用这些知识完全可以理解本章中的两个小例子，其中第一个例子是Hibernate与普通应用程序结合使用的例子，相对简单，而第二个例子介绍的是Hibernate与Tomcat结合的例子，是本章的重点，读者可以着重通过这个例子理解Hibernate。