

第 1 天：hibernate 的基础入门

第1章 hibernate 和 ORM 的概念部分

1.1 Hibrenate 概述

Hibernate 框架是当今主流的 Java 持久层框架之一，由于它具有简单易学、灵活性强、扩展性强等特点，能够大大地简化程序的代码量，提高工作效率，因此受到广大开发人员的喜爱。

Hibernate 是一个开放源代码的 ORM 框架，它对 JDBC 进行了轻量级的对象封装，使得 Java 开发人员可以使用面向对象的编程思想来操作数据库。

Hibernate（开放源代码的对象关系映射框架）

编辑

Hibernate是一个开放源代码的对象关系映射框架，它对JDBC进行了非常轻量级的对象封装，它将POJO与数据库表建立映射关系，是一个全自动的orm框架，hibernate可以自动生成SQL语句，自动执行，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，完成数据持久化的重任。

1.2 ORM 概述

Object Relation Mapping 对象关系映射。

对象-关系映射（OBJECT/RELATIONALMAPPING，简称 ORM），是随着面向对象的软件开发方法发展而产生的。用来把对象模型表示的对象映射到基于 SQL 的关系模型数据库结构中去。这样，我们在具体的操作实体对象的时候，就不需要再去和复杂的 SQL 语句打交道，只需简单的操作实体对象的属性和方法^[2]。ORM 技术是在对象和关系之间提供了一条桥梁，前台的对象型数据和数据库中的关系型的数据通过这个桥梁来相互转化^[1]。

简单的说就是把我们的程序中的实体类和数据库表建立起来对应关系。

+ | ★ 收藏 | 756 | 12

ORM

编辑

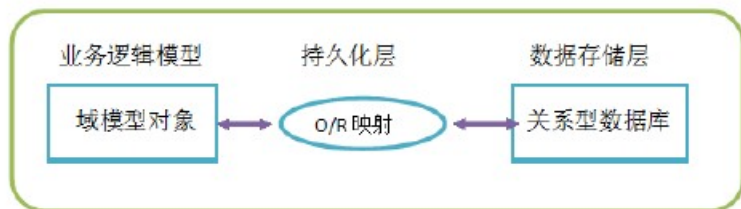
对象关系映射（英语：Object Relation Mapping，简称ORM，或O/RM，或O/R mapping），是一种程序技术，用于实现面向对象编程语言里不同类型系统的数据之间的转换^[1]。从效果上说，它其实是创建了一个可在编程语言里使用的--“虚拟对象数据库”。



1.2.1.1 为什么要学习 Hibernate

使用传统的 JDBC 开发应用系统时，如果是小型应用系统，并不觉得有什么麻烦，但是对于大型应用系统的开发，使用 JDBC 就会显得力不从心。例如对几十、几百张包含几十个字段的表进行插入操作时，编写的 SQL 语句不但很长，而且繁琐，容易出错；在读取数据时，需要写多条 getXXX 语句从结果集中取出各个字段的信息，不但枯燥重复，并且工作量非常大。为了提高数据访问层的编程效率，Gavin King 开发出了一个当今最流行的 ORM 框架，它就是 Hibernate 框架。

所谓的 ORM 就是利用描述对象和数据库表之间映射的元数据，自动把 Java 应用程序中的对象，持久化到关系型数据库的表中。通过操作 Java 对象，就可以完成对数据库表的操作。可以把 ORM 理解为关系型数据和对象的一个纽带，开发人员只需要关注纽带一端映射的对象即可。ORM 原理如图 1-1 所示。



ORM 原理

与其它操作数据库的技术相比，Hibernate 具有以下几点优势：

- **Hibernate** 对 JDBC 访问数据库的代码做了轻量级封装，大大简化了数据访问层繁琐的重复性代码，并且减少了内存消耗，加快了运行效率。
- **Hibernate** 是一个基于 JDBC 的主流持久化框架，是一个优秀的 ORM 实现，它很大程度的简化了 DAO（Data Access Object，数据访问对象）层编码工作。
- **Hibernate** 的性能非常好，映射的灵活性很出色。它支持很多关系型数据库，从一对一到多对多的各种复杂关系。
- 可扩展性强，由于源代码的开源以及 API 的开放，当本身功能不够用时，可以自行编码进行扩展。

明确：

操作实体类就相当于操作数据库表

第2章 CRM 介绍

CRM（Customer Relationship Management）客户关系管理，是利用相应的信息技术以及互联网技术来协调企业与顾客间在销售、营销和服务上的交互，向客户提供创新式的个性化的客户交互和服务的过程。

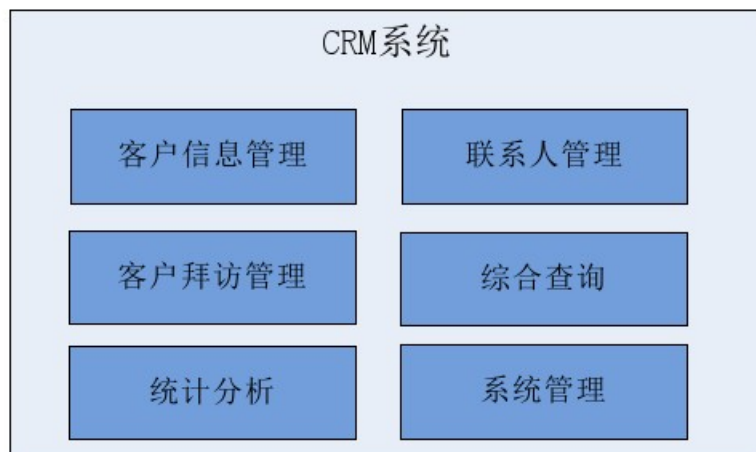
其最终目标是将面向客户的各项信息和活动集成起来，组建一个以客户为中心的企业，实现对面向客户的活动的全面管理。

2.1 功能模块划分：

CRM 系统实现了对企业销售、营销、服务等各阶段的客户信息、客户活动进行统一管理。

CRM 系统功能涵盖企业销售、营销、用户服务等各业务流程，业务流程中与客户相关活动都会在 CRM 系统统一管理。

下边列出一些基本的功能模块，包括：客户信息管理、联系人管理、商机管理、统计分析等。



客户信息管理

对客户信息统一维护，客户是指存量客户或拟营销的客户，通过员工录入形成公司的“客户库”是公司最重要的数据资源。

联系人管理

对客户的联系人信息统一管理，联系人是指客户企业的联系人，即企业的业务人员和客户的哪些人在打交道。

客户拜访管理

业务员（用户）要开发客户需要去拜访客户，客户拜访信息记录了业务员与客户沟通交流方面的不足、采取的策略不当、有待改进的地方或值得分享的沟通技巧等方面的信息。

综合查询

客户相关信息查询，包括：客户信息查询、联系人信息查询、商机信息查询等。

统计分析

按分类统计客户信息，包括：客户信息来源统计、按行业统计客户、客户发展数量统计等。

系统管理

系统管理属于 crm 系统基础功能模块，包括：数据字典、账户管理、角色管理、权限管理、操作日志管理等。

第3章 Hibernate 快速入门

3.1 需求介绍

本章节我们是实现的功能是保存一个客户到数据库的客户表中。

3.2 开发包和版本介绍

下载网址：<http://sourceforge.net/projects/hibernate/files/hibernate-orm/5.0.7.Final/>

页面显示如下图：

Home / hibernate-orm / 5.0.7.Final			
Name	Modified	Size	Downloads / Week
↑ Parent folder			
hibernate-release-5.0.7.Final.zip	2016-01-13	94.1 MB	14 <input type="text"/>
hibernate-release-5.0.7.Final.tgz	2016-01-13	58.4 MB	1 <input type="text"/>

开发包目录，如下图所示：

计算机 > 新加卷 (D:) > hibernate-release-5.0.7.Final			
名称	修改日期	类型	大小
documentation	2016/1/13 12:45	文件夹	
lib	2016/1/13 12:45	文件夹	
project	2016/1/13 12:45	文件夹	
changelog.txt	2016/1/13 12:33	文本文档	46 KB
hibernate_logo.gif	2013/4/17 11:37	GIF 文件	2 KB
lgpl.txt	2013/4/17 11:37	文本文档	26 KB

从图可以看出，hibernate5.0.7 的解压 s 目录中包含一系列的子目录，这些子目录分别用于存放不同功能的文件，接下来针对这些子目录进行简单介绍，具体如下：

documentation 文件夹：存放 Hibernate 的相关文档，包括参考文档的 API 文档。


lib 文件夹：存放 Hibernate 编译和运行所依赖的 JAR 包。其中 **required** 子目录下包含了运行 Hibernate5 项目必须的 JAR 包。

project 文件夹：存放 Hibernate 各种相关的源代码。










3.3 搭建 hibernate 开发环境（重点内容）

3.3.1 第一步：拷贝必备的 jar 包到开发目录




数据库驱动包，如下图：

	mysql-connector-java-5.1.7-bin.jar	2014/7/8 18:41	Executable Jar File	694 KB
---	------------------------------------	----------------	---------------------	--------

Hibernate/lib/required/*.jar 如下图

	antlr-2.7.7.jar	2014/4/28 20:30	Executable Jar File	435 KB
	dom4j-1.6.1.jar	2014/4/28 20:28	Executable Jar File	307 KB
	geronimo-jta_1.1_spec-1.1.1.jar	2015/5/5 11:26	Executable Jar File	16 KB
	hibernate-commons-annotations-5.0....	2015/11/30 10:22	Executable Jar File	74 KB
	hibernate-core-5.0.7.Final.jar	2016/1/13 12:35	Executable Jar File	5,453 KB
	hibernate-jpa-2.1-api-1.0.0.Final.jar	2014/4/28 20:30	Executable Jar File	111 KB
	jandex-2.0.0.Final.jar	2015/11/30 10:22	Executable Jar File	184 KB
	javassist-3.18.1-GA.jar	2014/4/28 20:28	Executable Jar File	698 KB
	jboss-logging-3.3.0.Final.jar	2015/5/28 12:35	Executable Jar File	66 KB

日志记录的包，如下图

	log4j-1.2.16.jar	2015/8/6 14:04	Executable Jar File	471 KB
	slf4j-api-1.6.1.jar	2015/8/6 14:05	Executable Jar File	25 KB
	slf4j-log4j12-1.7.2.jar	2015/8/6 14:05	Executable Jar File	9 KB

3.3.2 第二步：创建数据库和实体类

持久化类是应用程序中的业务实体类，这里的持久化是指类的对象能够被持久化保存到数据库中。Hibernate 使用普通 Java 对象（Plain Old Java Object），即 POJO 的编程模式来进行持久化。POJO 类中包含的是与数据库表相对应的各个属性，这些属性通过 getter 和 setter 方法来访问，对外部隐藏了内部的实现细节。下面就来编写 Customer 持久化类。

在项目 src 目录下，创建 cn.itcast.domain 包，并在包中创建实体类 Customer（对应数据库表 cst_customer），Customer 类包含与 cst_customer 数据表字段对应的属性，以及相应的 getXxx() 和 setXxx() 方法。

```

/* 创建客户表 */
CREATE TABLE `cst_customer` (
  `cust_id` bigint(32) NOT NULL AUTO_INCREMENT COMMENT '客户编号(主键)',
  `cust_name` varchar(32) NOT NULL COMMENT '客户名称(公司名称)',
  `cust_source` varchar(32) DEFAULT NULL COMMENT '客户信息来源',
  `cust_industry` varchar(32) DEFAULT NULL COMMENT '客户所属行业',

```



```
`cust_level` varchar(32) DEFAULT NULL COMMENT '客户级别',
`cust_address` varchar(128) DEFAULT NULL COMMENT '客户联系地址',
`cust_phone` varchar(64) DEFAULT NULL COMMENT '客户联系电话',
PRIMARY KEY (`cust_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

```
/**
 * 客户的实体类
 */
public class Customer implements Serializable {

    private Long custId;
    private String custName;
    private String custSource;
    private String custIndustry;
    private String custLevel;
    private String custAddress;
    private String custPhone;

    public Long getCustId() {
        return custId;
    }

    public void setCustId(Long custId) {
        this.custId = custId;
    }

    public String getCustName() {
        return custName;
    }

    public void setCustName(String custName) {
        this.custName = custName;
    }

    public String getCustSource() {
        return custSource;
    }

    public void setCustSource(String custSource) {
        this.custSource = custSource;
    }

    public String getCustIndustry() {
        return custIndustry;
    }

    public void setCustIndustry(String custIndustry) {
        this.custIndustry = custIndustry;
    }

    public String getCustLevel() {
```




```

        return custLevel;
    }

    public void setCustLevel(String custLevel) {
        this.custLevel = custLevel;
    }

    public String getCustAddress() {
        return custAddress;
    }

    public void setCustAddress(String custAddress) {
        this.custAddress = custAddress;
    }

    public String getCustPhone() {
        return custPhone;
    }

    public void setCustPhone(String custPhone) {
        this.custPhone = custPhone;
    }

    @Override
    public String toString() {
        return "Customer [custId=" + custId + ", custName=" + custName + ",
custSource=" + custSource
                + ", custIndustry=" + custIndustry + ", custLevel=" +
custLevel + ", custAddress=" + custAddress
                + ", custPhone=" + custPhone + "]\n";
    }
}

```

3.3.3 第三步：编写映射配置文件（xml）

实体类 **Customer** 目前还不具备持久化操作的能力，而 **Hibernate** 需要知道实体类 **Customer** 映射到数据库 **Hibernate** 中的哪个表，以及类中的哪个属性对应数据库表中的哪个字段，这些都需要在映射文件中配置。

在实体类 **Customer** 所在的包中，创建一个名称为 **Customer.hbm.xml** 的映射文件，在该文件中定义了实体类 **Customer** 的属性是如何映射到 **cst_customer** 表的列上的。

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- 导入约束:dtd约束
位置：在 Hibernate 的核心 jar 包中名称为 hibernate-mapping-3.0.dtd
明确该文件中的内容：
    实体类和表的对应关系
    实体类中属性和表的字段的对应关系
-->
<!DOCTYPE hibernate-mapping PUBLIC

```



```

    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
    <hibernate-mapping package="com.itheima.domain"><!-- package 属性用于设定包的
名称，接下来该配置文件中凡是用到此包中的对象时都可以省略包名 -->
    <!-- class 标签
        作用：建立实体类和表的对应关系
        属性：
            name: 指定实体类的名称
            table: 指定数据库表的名称
    -->
    <class name="Customer" table="cst_customer">
        <!-- id 标签
            作用：用于映射主键
            属性：
                name: 指定的是属性名称。也就是 get/set 方法后面的部分，并且首字母要
转小写。
                column: 指定的是数据库表的字段名称
        -->
        <id name="custId" column="cust_id">
            <!-- generator 标签：
                作用：配置主键的生成策略。
                属性：
                    class: 指定生成方式的取值。
                    取值之一：native。使用本地数据库的自动增长能力。
                    mysql 数据库的自动增长能力是让某一列自动+1。但是不是所有数据库都支持
这种方式。
            -->
            <generator class="native"></generator>
        </id>
        <!-- property 标签：
            作用：映射其他字段
            属性：
                name: 指定属性的名称。和 id 标签的 name 属性含义一致
                column: 指定数据库表的字段名称
        -->
        <property name="custName" column="cust_name"></property>
        <property name="custLevel" column="cust_level"></property>
        <property name="custSource" column="cust_source"></property>
        <property name="custIndustry" column="cust_industry"></property>
        <property name="custAddress" column="cust_address"></property>
        <property name="custPhone" column="cust_phone"></property>
    </class>
</hibernate-mapping>

```




3.3.4 第四步：编写主配置文件（hibernate.cfg.xml）

Hibernate 的映射文件反映了持久化类和数据库表的映射信息，而 Hibernate 的配置文件则主要用来配置数据库连接以及 Hibernate 运行时所需要的各个属性的值。在项目的 src 下创建一个名称为 hibernate.cfg.xml 的文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- 导入 dtd 约束:
位置: 在核心 jar 包中的名称为 hibernate-configuration-3.0.dtd 中
-->
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<!-- 配置 SessionFactory
SessionFactory 就是一个工厂，用于生产 Session 对象的。
Session 就是我们使用 hibernate 操作数据库的核心对象了。
明确:
    它和我们 Web 阶段的 HttpSession 没一点关系。
    此配置文件中的内容不需要背，很多配置都是可以在开发包中找到的。
    但是要求必须知道:
    创建 SessionFactory 由三部分组成，缺一不可。要知道是哪三部分
        1、连接数据库的基本信息
        2、hibernate 的基本配置
        3、映射文件的位置
    找配置文件的 key 都是在 hibernate 的开发包中 project 文件夹下的 etc 目录中的
hibernate.properties
-->
<session-factory>
    <!-- 1、连接数据库的基本信息 -->
    <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/day36_ee48_hiber
nate</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">1234</property>
    <!-- 2、hibernate 的基本配置 -->
    <!-- 数据库的方言 -->
    <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <!-- 是否显示 SQL 语句 -->
    <property name="hibernate.show_sql">true</property>
    <!-- 是否格式化 SQL 语句 -->
    <property name="hibernate.format_sql">true</property>
```



```

<!-- 是否让 hibernate 根据表结构的变化来生成 DDL 语句
DDL: 数据定义语言
hibernate 可以根据映射文件来为我们生成数据库的表结构。
但是他不能生成数据库。
hbm2ddl.auto 的取值
    * none: 不用 Hibernate 自动生成表。
    * create: 每次都会创建一个新的表。(测试)
    * create-drop: 每次都会创建一个新的表，执行程序结束后删除这个表。(测试)
    * update: 如果数据库中有表，使用原来的表，如果没有表，创建一个新表。可以更新表结构。
    * validate: 只会使用原有的表，对映射关系进行校验。

-->
<property name="hibernate.hbm2ddl.auto">update</property>
<!-- 3、映射文件的位置 -->
<mapping resource="com/itheima/domain/Customer.hbm.xml"/>
</session-factory>
</hibernate-configuration>

```

3.4 实现保存操作：

在项目中新建一个名称为 `cn.itcast.test` 的包，然后在包中建立一个名为 `HibernateDemo1.java` 的文件，该文件是用来测试的类文件。

```

/**
 * hibernate 的入门案例：
 * 需求：把一个客户保存到数据库中
 */
public class HibernateDemo1 {

    /**
     * 步骤分析：
     * 1、加载主配置文件
     * 2、根据主配置文件中的配置构建 SessionFactory
     * 3、使用工厂生产一个 Session 对象
     * 4、使用 Session 对象开启事务
     * 5、执行保存客户操作
     * 6、提交事务
     * 7、释放资源
     */
    @Test
    public void test1() {
        Customer c = new Customer();
        c.setCustName("黑马程序员 1");
    }
}

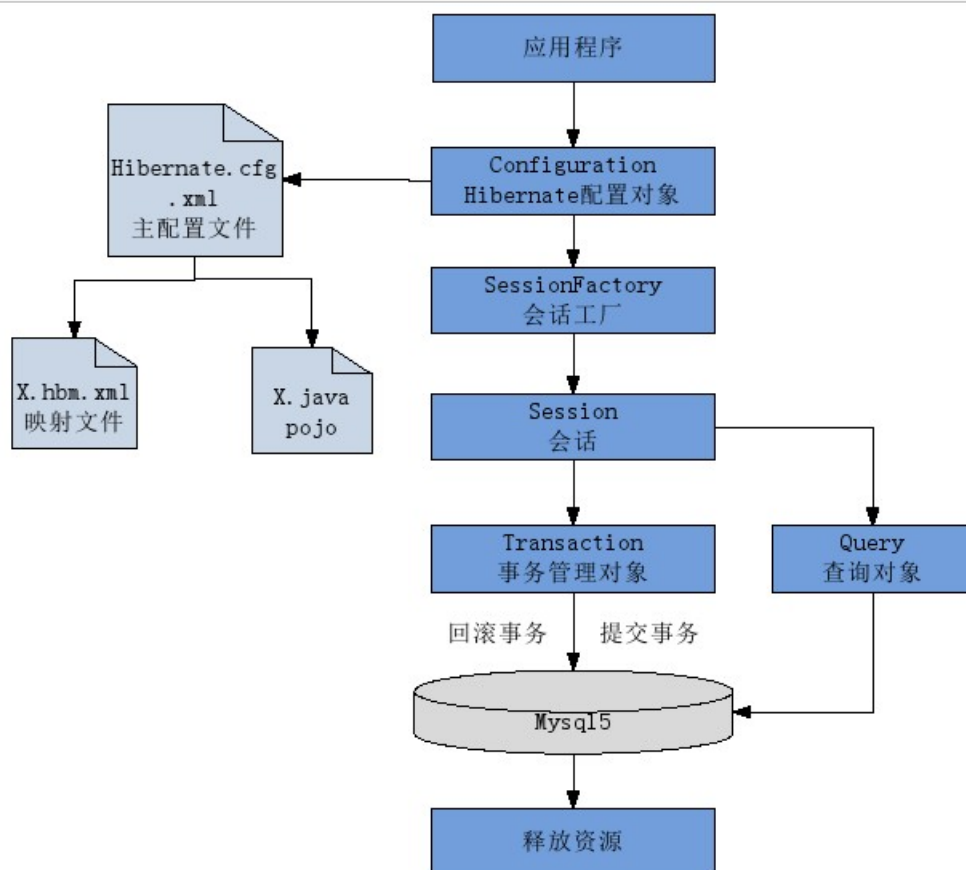
```



```
//1.加载主配置文件
Configuration cfg = new Configuration();
cfg.configure();
//2.构建 SessionFactory
SessionFactory factory = cfg.buildSessionFactory();
//3.使用 SessionFactory 生产一个 Session
Session session = factory.openSession();//打开一个新的 Session
//4.开启事务
Transaction tx = session.beginTransaction();
//5.保存客户
session.save(c);//根据映射配置文件，生成 SQL 语句，实现保存。
//6.提交事务
tx.commit();
//7.释放资源
session.close();
factory.close();
}
}
```

3.5 入门案例的执行过程

首先创建 **Configuration** 类的实例，并通过它来读取并解析配置文件 **hibernate.cfg.xml**。然后创建 **SessionFactory** 读取解析映射文件信息，并将 **Configuration** 对象中的所有配置信息拷贝到 **SessionFactory** 内存中。接下来，打开 **Session**，让 **SessionFactory** 提供连接，并开启一个事务，之后创建对象，向对象中添加数据，通过 **session.save()**方法完成向数据库中保存数据的操作。最后提交事务，并关闭资源。



3.6 Hibernate 的常见配置

在案例中，已经接触过 **Hibernate** 的映射文件和配置文件。接下来，将对这些文件进行详细的讲解。

3.6.1 映射文件的配置

该文件用于向 **Hibernate** 提供持久化类到关系型数据库的映射，每个映射文件的的结构基本都是相同的，其普遍的代码形式如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<!--映射文件的 dtd 信息 -->
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <!-- name 代表的是实体类名 table 代表的是表名 -->
    <class name="XXX" table="xxx">
        <!-- name=id 代表的是 XXX 类中属性 column=id 代表的是 xxx 表中的字段 -->
        <id name="id" column="id">
            <generator class="native"/><!-- 主键生成策略 -->
        </id>
```



```
<!-- 其它属性使用 property 标签来映射 -->
<property name="XXX" column="xxx" type="string" />
</class>
</hibernate-mapping>
```

映射文件通常是一个 XML 文件即可,但一般命名为类名.hbm.xml

3.6.2 核心配置

Hibernate 的配置文件,包含了连接持久层与映射文件所需的基本信息,其配置文件有两种格式,具体如下:

- 一种是 **properties** 属性文件格式的配置文件,它使用键值对的形式存放信息,默认文件名称为 **hibernate.properties**;
- 另一种是 **XML** 格式的配置文件,XML 配置文件的默认名称为 **hibernate.cfg.xml**。

上述两种格式的配置文件是等价的,具体使用哪个可以自由选择。**XML** 格式的配置文件更易于修改,配置能力更强,当改变底层应用配置时不需要改变和重新编译代码,只修改配置文件的相应属性即可,而 **properties** 格式的文件则不具有此优势,因此,在实际开发项目中,大多数情况会使用 **XML** 格式的配置文件。下面将对 **XML** 格式的配置文件进行详细介绍。

hibernate.cfg.xml 配置文件一般在开发时会放置在 **src** 的源文件夹下,发布后,该文件会在项目的 **WEB-INF/classes** 路径下。配置文件的常用配置信息如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- 必要的配置信息:连接数据库的基本参数 -->
        <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
name="hibernate.connection.url">jdbc:mysql:///hibernate_day01</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">1234</property>

        <!-- Hibernate 的属性 -->
        <!-- Hibernate 的方言:作用,根据配置的方言生成相应的 SQL 语句 -->
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

        <!-- Hibernate 显示 SQL 语句: -->
        <property name="hibernate.show_sql">true</property>
        <!-- Hibernate 格式化 SQL 语句: -->
        <property name="hibernate.format_sql">true</property>
```



```
<!-- Hibernate 的 hbm2ddl (数据定义语言:create drop alter ...) 属性 -->
<!--
    hbm2ddl.auto 的取值
    * none          :不用 Hibernate 自动生成表.
    * create        :每次都会创建一个新的表.(测试)
    * create-drop   :每次都会创建一个新的表, 执行程序结束后删除这个表.(测试)
    * update        :如果数据库中有表, 使用原来的表, 如果没有表, 创建一个新表, 可以更新表结构.
    * validate      :只会使用原有的表. 对映射关系进行校验.
-->
<property name="hibernate.hbm2ddl.auto">update</property>

<!-- Hibernate 加载映射 -->
<mapping resource="cn/itcast/domain/Customer.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

在上述代码中, 首先进行了 xml 声明, 然后是配置文件的 dtd 信息, 该信息同样可以在核心包 hibernate-core-5.0.7.Final.jar 下的 org.hibernate 包中的 hibernate-configuration-3.0.dtd 文件中找到, 读者只需要复制过来用即可, 不需要刻意记忆。

Hibernate 配置文件的根元素是 hibernate-configuration, 该元素包含子元素 session-factory, 在 session-factory 元素中又包含多个 property 元素, 这些 property 元素用来对 Hibernate 连接数据库的一些重要信息进行配置。例如, 上面的配置文件中, 使用了 property 元素配置了数据库的方言、驱动、URL、用户名、密码等信息。最后通过 mapping 元素的配置, 加载出映射文件的信息。

Hibernate 配置文件的一些常用属性名称及用途, 如表所示。

表1-1 Hibernate 常用配置属性

名称	用途
hibernate.dialect	操作数据库方言
hibernate.connection.driver_class	连接数据库驱动程序
hibernate.connection.url	连接数据库 URL
hibernate.connection.username	数据库用户名
hibernate.connection.password	数据库密码
hibernate.show_sql	在控制台上输出 SQL 语句
hibernate.format_sql	格式化控制台输出的 SQL 语句
hibernate.hbm2ddl.auto	当 SessionFactory 创建时是否根据映射文件自动验证表结构或自动创建、自动更新数据库表结构。该参数的取值为: validate、update、create 和 create-drop。
hibernate.connection.autocommit	事务是否自动提交



第4章 Hibernate 中 API 介绍

4.1 Configuration 对象

4.1.1 作用：

在使用 Hibernate 时，首先要创建 Configuration 实例，Configuration 实例主要用于启动、加载、管理 hibernate 的配置文件信息。在启动 Hibernate 的过程中，Configuration 实例首先确定 Hibernate 配置文件的位置，然后读取相关配置，最后创建一个唯一的 SessionFactory 实例。Configuration 对象只存在于系统的初始化阶段，它将 SessionFactory 创建完成后，就完成了自己的使命。

Hibernate 通常使用 `Configuration config = new Configuration().configure();` 的方式创建实例，此种方式默认会去 src 下读取 `hibernate.cfg.xml` 配置文件。如果不想使用默认的 `hibernate.cfg.xml` 配置文件，而是使用指定目录下（或自定义）的配置文件，则需要向 `configure()` 方法中传递一个文件路径的参数，其代码写法如下：

```
Configuration config = new Configuration().configure("xml 文件位置");
```

此种写法 hibernate 会去指定位置查找配置文件，例如，想要使用 src 下 config 包中的 `hibernate.cfg.xml` 文件，只需将文件位置加入 `configure()` 中即可，其代码如下所示：

```
Configuration config = new Configuration().configure("/config/hibernate.cfg.xml");
```

【加载映射文件】

Hibernate 除了可以使用 Configuration 对象加载核心配置文件以外，还可以利用该对象加载映射文件。因为如何使用 properties 文件作为 Hibernate 的核心配置文件，其他的属性可以使用 `key=value` 的格式来设置，但是映射没有办法加载。这时这个对象就有了用武之地。可以在手动编写代码的时候去加载映射文件。

```
Configuration configuration = new Configuration().configure("xml 文件位置");
configuration.addResource("cn/itcast/domain/Customer.hbm.xml");
```

4.1.2 常用方法：

默认构造函数：

它只能加载类的根路径下，名称为 `hibernate.properties` 的配置文件。不能加载 `xml` 配置文件。

它用于加载类的根路径下，名称为 `hibernate.cfg.xml` 的配置文件。

`buildSessionFactory()`：

根据配置文件，构建 SessionFactory

`addResource(String url);`

指定映射文件的位置

`addClass(Class clazz);`

指定实体类的字节码



4.2 SessionFactory

4.2.1 作用：

SessionFactory

SessionFactory接口负责初始化Hibernate。它充当数据存储源的代理，并负责创建Session对象。这里用到了工厂模式。需要注意的是SessionFactory并不是轻量级的，因为一般情况下，一个项目通常只需要一个SessionFactory就够，当需要操作多个数据库时，可以为每个数据库指定一个SessionFactory。

SessionFactory 接口负责 Hibernate 的初始化和建立 Session 对象。它在 Hibernate 中起到一个缓冲区作用，Hibernate 可以将自动生成的 SQL 语句、映射数据以及某些可重复利用的数据放在这个缓冲区中。同时它还保存了对数据库配置的所有映射关系，维护了当前的二级缓存。

SessionFactory 实例是通过 Configuration 对象获取的，其获取方法如下所示。

```
SessionFactory sessionFactory = config.buildSessionFactory();
```

4.2.2 常用方法：（今天只能介绍一个，不是就有一个方法）

```
openSession(): 每次都是生成一个新的 Session
```

4.2.3 细节：

该对象维护了很多信息：

- 连接数据库的信息
- hibernate 的基本配置
- 映射文件的位置，以及映射文件中的配置
- 一些预定义的 SQL 语句（这些语句都是通用的） 比如：全字段保存，根据 id 的全字段更新，根据 id 的全字段查询，根据 id 的删除等等。
- hibernate 的二级缓存（了解）

同时，它是一个线程安全的对象，所有由该工厂生产的 Session 都共享工厂中维护的数据。

4.2.4 使用原则：

由于 SessionFactory 维护了很多信息同时又是线程安全的，一般情况下，一个项目中只需要一个 SessionFactory，只有当应用中存在多个数据源时，才为每个数据源建立一个 SessionFactory 实例。因此，不应该反复的创建和销毁。

原则：一个应用应该只有一个 SessionFactory。在应用加载时创建，应用卸载时销毁。

4.2.5 在 hibernate 中使用数据源(连接池)

SessionFactory 内部还维护了一个连接池，如果我们需要使用第三方的连接池如 C3P0，那么



需要我们自己手动进行配置

配置 C3P0 步骤如下：

1、导入连接池的 jar 包

```
c3p0-0.9.2.1.jar  
hibernate-c3p0-5.0.7.Final.jar  
mchange-commons-java-0.2.3.4.jar
```

2、在 hibernate 主配置文件中配置

```
<!-- 配置数据源的提供商 -->  
<property name="hibernate.connection.provider_class">  
    org.hibernate.connection.C3P0ConnectionProvider  
</property>
```

4.3 Session

4.3.1 作用：

Session

Session 接口负责执行被持久化对象的 CRUD 操作 (CRUD 的任务是完成与数据库的交流，包含了很多常见的 SQL 语句)。但需要注意的是 **Session 对象是非线程安全的**。同时，Hibernate 的 **session** 不同于 JSP 应用中的 **HttpSession**。这里当使用 **session** 这个术语时，其实指的是 Hibernate 中的 **session**，而以后会将 **HttpSession** 对象称为用户 **session**。

Session 是应用程序与数据库之间交互操作的一个单线程对象，是 **Hibernate** 运作的中心，它的主要功能是为持久化对象提供创建、读取和删除的能力，所有持久化对象必须在 **session** 的管理下才可以进行持久化操作。

创建 **SessionFactory** 实例后，就可以通过它获取 **Session** 实例。获取 **Session** 实例有两种方式，一种是通过 **openSession()** 方法，另一种是通过 **getCurrentSession()** 方法。两种方法获取 **session** 的代码如下所示：

```
//采用 openSession 方法创建 session  
Session session = sessionFactory.openSession();  
//采用 getCurrentSession() 方法创建 session  
Session session = sessionFactory.getCurrentSession();
```

以上两种获取 **session** 实例方式的主要区别是，采用 **openSession** 方法获取 **Session** 实例时，**SessionFactory** 直接创建一个新的 **Session** 实例，并且在使用完成后需要调用 **close** 方法进行手动关闭。而 **getCurrentSession** 方法创建的 **Session** 实例会被绑定到当前线程中，它在提交或回滚操作时会自动关闭。

在没有配置把 **Session** 绑定当前线程之前，**getCurrentSession** 方法无法使用，所以今天我们只使用 **openSession** 方法。配置的方式，我们明天来讲解。



4.3.2 常用方法：

```
save(Object entity); : 保存一个实体到数据库
update(Object entity);: 更新一个实体
delete(Object entity);: 删除一个实体
get(Class clazz,Serializable id);: 根据 id 查询一个实体。参数的含义: Class 表示要
查询的实体类字节码。Serializable 就是查询的条件。
beginTransaction();: 开启事务，并返回事务对象
```

4.3.3 细节：

由于 **SessionFactory** 已经维护了很多数据，所以 **Session** 就维护较少的内容。
它是一个轻量级对象。并且：它不是线程安全的!!!!!!
它维护了 **hibernate** 的一级缓存（关于一级缓存的问题都是明天的内容）。
它的反复创建销毁不会消耗太多资源。

4.3.4 使用原则：

每个线程都只有一个 **Session** 对象。

4.4 Transaction

4.4.1 作用：

Transaction

Transaction 接口是一个可选的API，可以选择不使用这个接口，取而代之的是Hibernate 的设计者自己写的底层事务处理代码。**Transaction** 接口是对实际事务实现的一个抽象，这些实现包括JDBC的事务、JTA 中的UserTransaction、甚至可以是CORBA 事务。之所以这样设计是能让开发者能够使用一个统一事务的操作界面，使得自己的项目可以在不同的环境和容器之间方便地移植。

Transaction 接口主要用于管理事务，它是 **Hibernate** 的数据库事务接口，且对底层的事务接口进行了封装。**Transaction** 接口的事务对象是通过 **Session** 对象开启的，其开启方式如下所示。

```
Transaction transaction = session.beginTransaction();
```

4.4.2 常用方法：

```
commit(): 提交事务
rollback(): 回滚事务
```

Session 执行完数据库操作后，要使用 **Transaction** 接口的 **commit()**方法进行事务提交，才能



真正的将数据操作同步到数据库中。发生异常时，需要使用 `rollback()` 方法进行事务回滚，以避免数据发生错误。因此，在持久化操作后，必须调用 `Transaction` 接口的 `commit()` 方法和 `rollback()` 方法。如果没有开启事务，那么每个 `Session` 的操作，都相当于一个独立的操作。

第5章 抽取 `HibernateUtil` 工具类

在上一章节中介绍了 `SessionFactory` 的这些特点，一般情况下，在实际项目使用中，通常会抽取出一个 `HibernateUtils` 的工具类，用来提供 `Session` 对象。

`Hibernate` 的工具类：

```
/**
 * hibernate 的工具类
 * 用于生产一个 Session 对象
 */
public class HibernateUtil {

    private static SessionFactory factory;

    static {
        try {
            Configuration cfg = new Configuration();
            cfg.configure();
            factory = cfg.buildSessionFactory();
        } catch (Exception e) {
            //e.printStackTrace();
            throw new ExceptionInInitializerError("初始化 SessionFactory 失败");
        }
    }

    /**
     * 使用工厂生产一个 Session 对象，
     * 每次都是一个新的
     * 此时 Session 还不符合自己的使用原则，调整符合放到 hibernate 的第二天
     * @return
     */
    public static Session openSession() {
        return factory.openSession();
    }
}
```



第6章 案例：使用 **Hibernate** 实现增删改查

6.1 保存操作

```
/**
 * hibernate 的增删改查 (查一个)
 */
public class HibernateDemo4 {

    /**
     * 保存
     */
    @Test
    public void testAdd() {
        Customer c = new Customer();
        c.setCustName("黑马程序员 test");
        c.setCustLevel("VIP 客户");
        c.setCustSource("网络");
        c.setCustIndustry("IT 教育");
        c.setCustAddress("昌平区北七家镇");
        c.setCustPhone("010-84389340");

        //1.使用工具类获取一个 Session
        Session session = HibernateUtil.openSession();
        //2.开启事务
        Transaction tx = session.beginTransaction();
        //3.保存客户
        session.save(c);
        //4.提交事务
        tx.commit();
        //5.释放资源
        session.close();
    }
}
```

6.2 查询一个实体

```
/**
 * 根据 id 查询一个实体
```




```
*/  
  
@Test  
public void testFindOne() {  
    //1.使用工具类获取一个 Session  
    Session session = HibernateUtil.openSession();  
    //2.开启事务  
    Transaction tx = session.beginTransaction();  
    //3.根据 id 查询  
    Customer c = session.get(Customer.class, 2L);  
    System.out.println(c);  
    //4.提交事务  
    tx.commit();  
    //5.释放资源  
    session.close();  
}
```

6.3 修改操作

```
/**  
 * 修改一个实体  
 */  
  
@Test  
public void testUpdate() {  
    //1.使用工具类获取一个 Session  
    Session session = HibernateUtil.openSession();  
    //2.开启事务  
    Transaction tx = session.beginTransaction();  
    //3.根据 id 查询  
    Customer c = session.get(Customer.class, 1L);  
    c.setCustName("TBD 云集中心");  
    //修改实体  
    session.update(c);  
  
    //4.提交事务  
    tx.commit();  
    //5.释放资源  
    session.close();  
}
```

6.4 删除操作

```
/**
```



```

    * 删除一个实体
    */
@Test
public void testDelete() {
    //1.使用工具类获取一个 Session
    Session session = HibernateUtil.openSession();
    //2.开启事务
    Transaction tx = session.beginTransaction();
    //3.根据 id 查询
    Customer c = session.get(Customer.class, 1L);
    //删除实体
    session.delete(c); //delete from cst_customer where cust_id = ?

    //4.提交事务
    tx.commit();
    //5.释放资源
    session.close();
}
}

```

6.5 实体查询的另一个方法 load

6.5.1 实体查询的概念

所谓实体查询即 **OID** 查询，就是使用主键作为条件来查询一个实体。其中涉及的方法是 **Session** 对象 **get** 方法和 **load** 方法。

在本章节都是使用客户查询示例。

6.5.2 方法的说明

get 方法:

```

/**
 * 根据 id 查询一个实体
 * @param entityType 指的是要查询的实体类字节码对象
 * @param id 查询的条件，即主键的值。
 * @return 返回的是实体类对象
 */
<T> T get(Class<T> entityType, Serializable id);

```

get 方法的代码演示:

```

/**
 * 需求： 使用 get 方法查询 id 为 1 的客户
 */

```



```
@Test

public void test1() {

    Session s = HibernateUtil.getCurrentSession();
    Transaction tx = s.beginTransaction();
    Customer c = s.get(Customer.class, 1L);
    System.out.println(c);
    tx.commit();

}
```

load 方法:

```
/**
 * 根据 id 查询一个实体
 * @param theClass 指的是要查询的实体类字节码
 * @param id 查询的条件，即主键的值。
 * @return 返回的是实体类对象或者是实体类对象的代理对象
 */
<T> T load(Class<T> theClass, Serializable id);
```

load 方法的代码演示:

```
/**
 * 需求： 使用 load 方法查询 id 为 1 的客户
 */

@Test

public void test2() {

    Session s = HibernateUtil.getCurrentSession();
    Transaction tx = s.beginTransaction();
    Customer c = s.load(Customer.class, 1L);
    System.out.println(c.toString());
    tx.commit();

}
```

问题：既然两个方法都是根据 ID 查询一个实体，他们有什么区别呢？

6.5.3 get 和 load 的区别

区别:

1、查询的时机不一样

get 方法任何时候都是立即加载，即只要一调用 get 马上发起数据库查询

load 方法默认情况下是延迟加载，即真正用到对象的非 OID 字段数据才发起查询

load 方法可以通过配置的方式改为立即加载。

配置的方式:

由于 load 方法是 hibernate 的方法所以只有 XML 的方式:

```
<class name="Customer" table="cst_customer" lazy="false">
```

2、返回的结果不一样

get 方法永远返回查询的实体类对象。

load 方法当是延迟加载时，返回的是实体类的代理对象。

涉及的概念:



立即加载：

是不管用不用马上查询。

延迟加载：

等到用的时候才真正发起查询。