

今日任务

使用Hibernate完成对CRM系统中客户管理的DAO中的CRUD的操作

案例一：

使用Hibernate完成CRM客户管理的CRUD的操作

1.1 需求描述

CRM系统中客户信息管理模块功能包括：

- 新增客户信息
- 客户信息查询
- 修改客户信息
- 删除客户信息

1.2 CRM的概述

1.2.1 什么是 CRM

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。

1.2.2 CRM的功能模块

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。

1.2.3 JavaEE开发的三层结构

1.3 Hibernate概述

1.3.1 什么是 Hibernate

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（Object Relational Mapping,对象关系映射）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。

★ 收藏 | 2112 | 170

Hibernate（开放源代码的对象关系映射框架）

编辑

本词条由“科普中国”科学百科词条编写与应用工作项目 审核。

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JaveEE架构中取代CMP，完成[数据持久化](#)的重任。

中文名	对象关系映射框架	属 性	开放源代码的对象关系映射框架
外文名	Hibernate	作 用	数据库与界面之间的桥梁
典型应用	EJB的J2EE架构中取代CMP	关键技术	数据持久化

Hibernate就是一个持久层的ORM的框架。

ORM :Object Relational Mapping.对象关系映射。

1.3.2 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。

```
graph LR; subgraph ORM_Principle [ORM原理]; direction LR; subgraph Business_Model [业务逻辑模型]; direction TB; DMO[域模型对象]; end; subgraph Persistence_Layer [持久化层]; direction TB; ORM([O/R映射]); end; subgraph Data_Storage_Layer [数据存储层]; direction TB; RDB[关系型数据库]; end; DMO <--> ORM; ORM <--> RDB; end
```

与其它操作数据库的技术相比，Hibernate具有以下优点：

- 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。

1.3.3 Hibernate的入门：

1.3.3.1下载Hibernate5

<https://sourceforge.net/projects/hibernate/files/hibernate-orm/5.0.7.Final/>

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
hibernate-release-5.0.7.Final.tgz	2016-01-13	58.4 MB	0
Totals: 2 Items		152.5 MB	14










Hibernate5.0.7版本下载后，解压完的目录结构如图所示。

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	2016/1/13 12:33	文本文档	46 KB
hibernate_logo	2013/4/17 11:37	GIF 图片文件	2 KB
lgpl	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。

在lib/required子目录中，包含的JAR包

名称	修改日期	类型	大小
 antlr-2.7.7	2014/4/28 20:30	Executable Jar File	435 KB
 dom4j-1.6.1	2014/4/28 20:28	Executable Jar File	307 KB
 geronimo-jta_1.1_spec-1.1.1	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	2016/1/13 12:35	Executable Jar File	5,453 KB
 hibernate-jpa-2.1-api-1.0.0.Final	2014/4/28 20:30	Executable Jar File	111 KB
 jandex-2.0.0.Final	2015/11/30 10:22	Executable Jar File	184 KB
 javassist-3.18.1-GA	2014/4/28 20:28	Executable Jar File	698 KB
 jboss-logging-3.3.0.Final	2015/5/28 12:35	Executable Jar File	66 KB

1.3.3.2 创建数据库和表










```
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_phone` varchar(64) DEFAULT NULL COMMENT '固定电话',
  `cust_mobile` varchar(16) DEFAULT NULL COMMENT '移动电话',
  PRIMARY KEY (`cust_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

1.3.3.3 引入 Hibernate 开发的 jar 包




数据库驱动包

 mysql-connector-java-5.1.7-bin	2008/10/21 4:02	Executable Jar File	694 KB
--	-----------------	---------------------	--------

Hibernate/lib/required/*.jar

名称	修改日期	类型	大小
 antlr-2.7.7	2014/4/28 20:30	Executable Jar File	435 KB
 dom4j-1.6.1	2014/4/28 20:28	Executable Jar File	307 KB
 geronimo-jta_1.1_spec-1.1.1	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	2016/1/13 12:35	Executable Jar File	5,453 KB
 hibernate-jpa-2.1-api-1.0.0.Final	2014/4/28 20:30	Executable Jar File	111 KB
 jandex-2.0.0.Final	2015/11/30 10:22	Executable Jar File	184 KB
 javassist-3.18.1-GA	2014/4/28 20:28	Executable Jar File	698 KB
 jboss-logging-3.3.0.Final	2015/5/28 12:35	Executable Jar File	66 KB

日志记录的包

 log4j-1.2.16	2017/3/16 14:15	Executable Jar File	471 KB
 slf4j-api-1.6.1	2017/3/16 14:15	Executable Jar File	25 KB
 slf4j-log4j12-1.7.2	2017/3/16 14:15	Executable Jar File	9 KB

1.3.3.4 创建实体(持久化类)

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。

```
package com.admiral.domain;

/**
 *
 */
public class Customer {

    private Long cust_id;
    private String cust_name;
    private String cust_source;
    private String cust_industry;
    private String cust_level;
    private String cust_phone;
    private String cust_mobile;

    public Long getCust_id() {
        return cust_id;
    }

    public void setCust_id(Long cust_id) {
        this.cust_id = cust_id;
    }

    public String getCust_name() {
        return cust_name;
    }

    public void setCust_name(String cust_name) {
        this.cust_name = cust_name;
    }

    public String getCust_source() {
        return cust_source;
    }

    public void setCust_source(String cust_source) {
        this.cust_source = cust_source;
    }

    public String getCust_industry() {
        return cust_industry;
    }
}
```

```

    public void setCust_industry(String cust_industry) {
        this.cust_industry = cust_industry;
    }

    public String getCust_level() {
        return cust_level;
    }

    public void setCust_level(String cust_level) {
        this.cust_level = cust_level;
    }

    public String getCust_phone() {
        return cust_phone;
    }

    public void setCust_phone(String cust_phone) {
        this.cust_phone = cust_phone;
    }

    public String getCust_mobile() {
        return cust_mobile;
    }

    public void setCust_mobile(String cust_mobile) {
        this.cust_mobile = cust_mobile;
    }
}

```

1.3.3.5 创建映射文件

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"?>

<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <!-- 建立类和表的映射关系 -->
    <class name="com.admiral.domain.Customer" table="cst_customer">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="cust_id" column="cust_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
        </id>

        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="cust_name" column="cust_name" />
    </class>
</hibernate-mapping>

```

```

        <property name="cust_source" column="cust_source" />
        <property name="cust_industry" column="cust_industry" />
        <property name="cust_level" column="cust_level" />
        <property name="cust_phone" column="cust_phone" />
        <property name="cust_mobile" column="cust_mobile" />
    </class>
</hibernate-mapping>

```

1.3.3.6 创建 Hibernate 的核心配置文件

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"?>
<!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:///test</property>
            <property name="hibernate.connection.username">root</property>
            <property name="hibernate.connection.password">111111</property>

            <!-- Hibernate 的方言:根据配置的方言生成对应的 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>

            <!-- 加载映射文件 -->
            <mapping resource="com/admiral/domain/Customer.hbm.xml"/>
        </session-factory>
    </hibernate-configuration>

```

1.3.3.7 编写测试代码

```

package com.admiral.test;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.junit.Test;

import com.admiral.domain.Customer;

public class HibernateDemo1 {

```



```

@Test
/**
 * 使用 hibernate 保存数据
 */
public void saveData() {
    //1 .加载配置文件
    Configuration cfg = new Configuration().configure();
    //2. 创建一个 SessionFactory
    SessionFactory sessionFactory = cfg.buildSessionFactory();
    //3. 创建 Session 对象 Session 对象类似于 Connection
    Session session = sessionFactory.openSession();
    //4. 开启事务
    Transaction transaction = session.beginTransaction();
    //5. 执行先关操作
    Customer customer = new Customer();
    customer.setCust_name("小红红");

    session.save(customer);
    //6. 事务提交
    transaction.commit();
    //7. 释放资源
    session.close();
}
}

```

The screenshot shows an IDE with the following components:

- Editor:** Displays the `saveData()` method from the previous code block. Line 17 is highlighted.
- Outline:** Shows the project structure with `com.admiral` and `HibernateDemo1`. The `saveData()` method is listed under `HibernateDemo1`.
- Console:** Shows the following output:


```

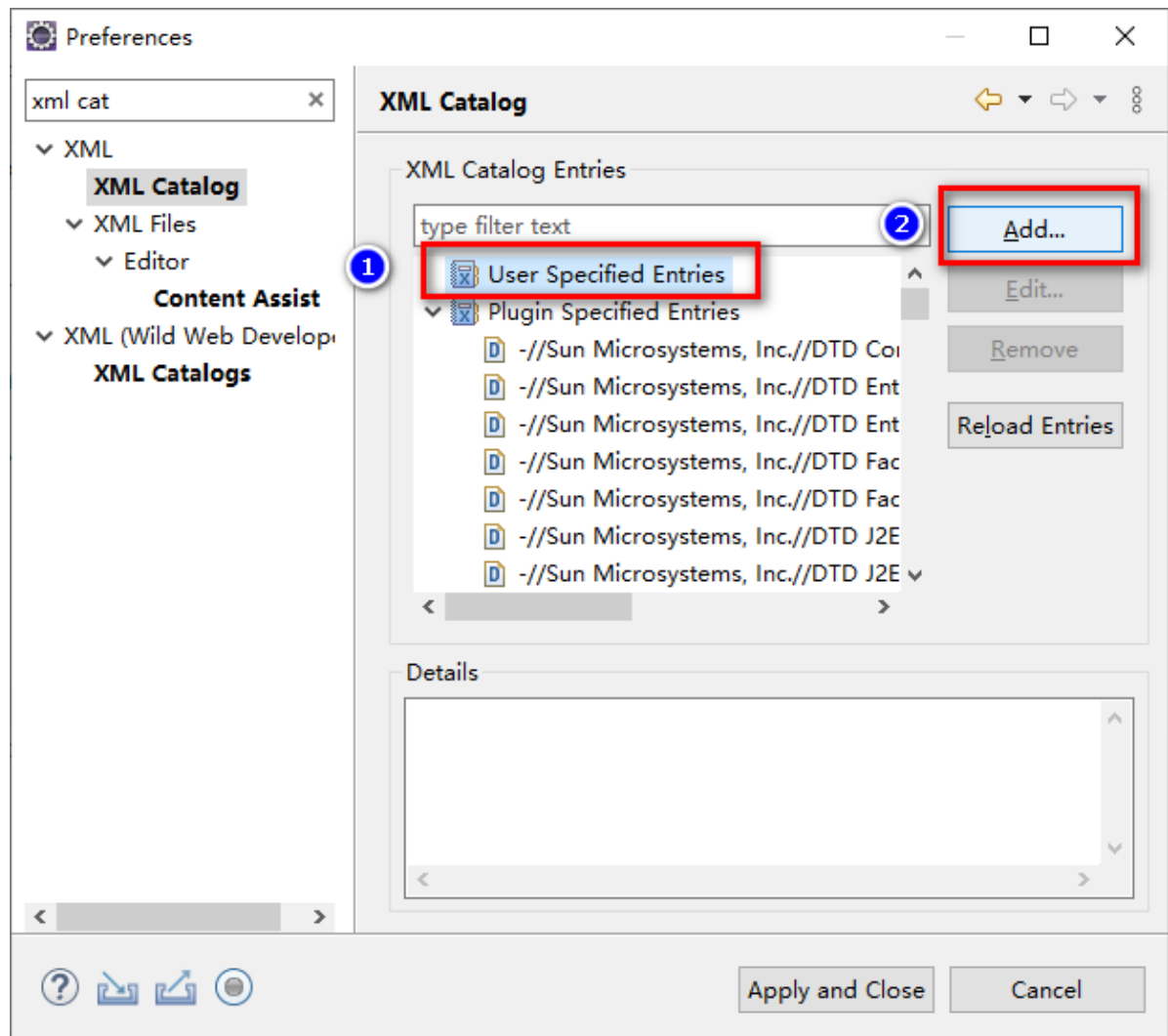
<terminated> HibernateDemo1.saveData [JUnit] D:\Program Files (x86)\Java\jre1.8.0_65\bin\javaw.exe (2020-9-14 10:20:05 - 10:20:11)
log4j:WARN No appenders could be found for logger (org.jboss.logging).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Hibernate: |
insert
into
  cst_customer
(cust_name, cust_source, cust_industry, cust_level, cust_phone, cust_mobile)
values
  (?, ?, ?, ?, ?, ?)
      
```

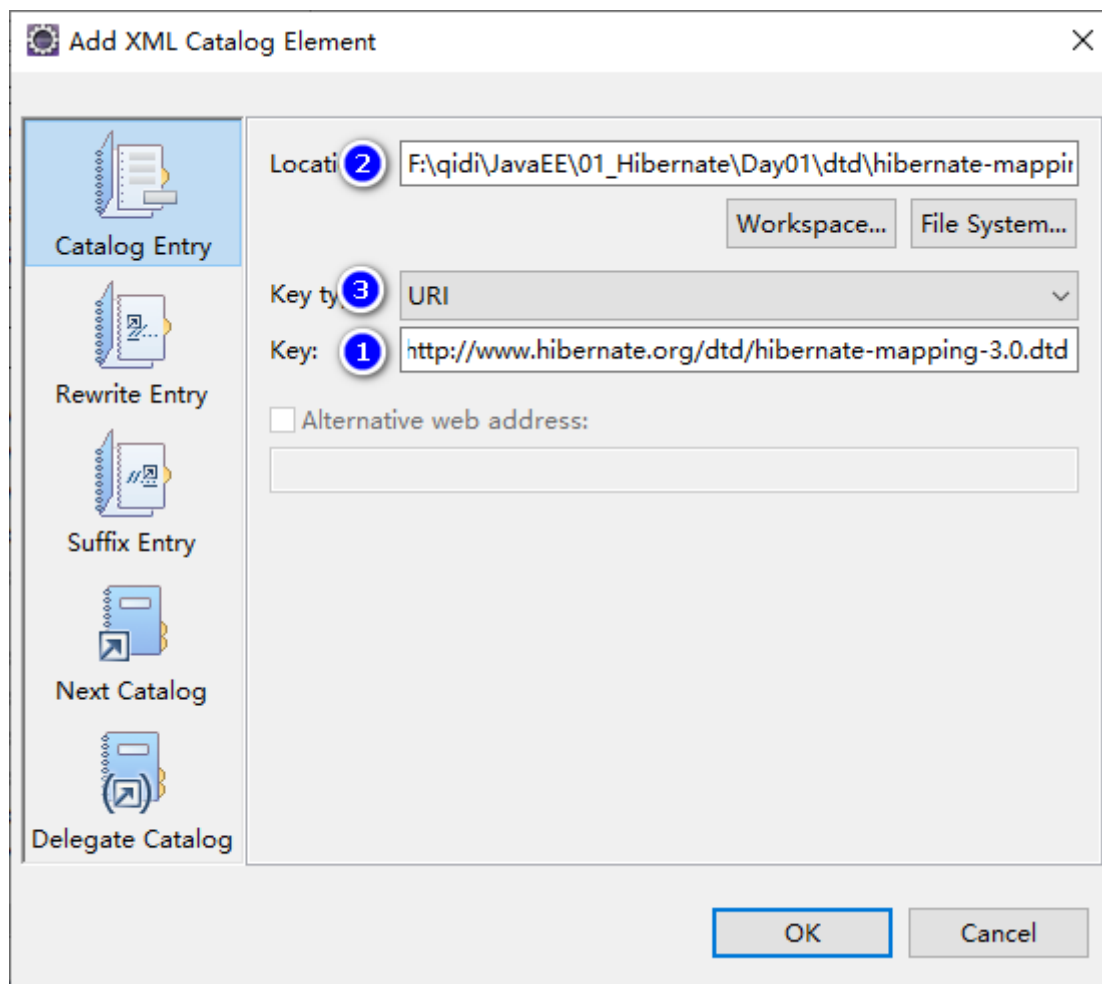
首先创建Configuration类的实例，并通过它来读取并解析配置文件hibernate.cfg.xml。然后创建SessionFactory读取解析映射文件信息，并将Configuration对象中的所有配置信息拷贝到SessionFactory内存中。接下来，打开Session,让SessionFactory提供连接，并开启一个事务，之后创建对象，向对象中添加数据，通过session.saveQ方法完成向数据库中保存数据的操作。最后提交事务，并关闭资源。

1.3.4 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。

1.3.4.1 XML 提示的配置





1.3.4.2 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"?>

<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <!-- 建立类和表的映射关系 -->
    <class name="com.admiral.domain.Customer" table="cst_customer">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="cust_id" column="cust_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
        </id>

        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="cust_name" column="cust_name" />
        <property name="cust_source" column="cust_source" />
        <property name="cust_industry" column="cust_industry" />
        <property name="cust_level" column="cust_level" />
        <property name="cust_phone" column="cust_phone" />
        <property name="cust_mobile" column="cust_mobile" />
    </class>
</hibernate-mapping>
```

```
</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

【class标签】

```

7<hibernate-mapping>
8  <!--
9      建立类和表的映射关系
10     name属性:   类中的全路径
11     table属性:  表名(如果类名和表名是一致的,那么表名可以省略)
12     catalog属性: 数据库名,可以省略
13  -->
14  <class name="com.admiral.domain.Customer" table="cst_customer">
```

- 【class标签的配置】

- 标签用来建立类与表的映射关系
- 属性:
 - name : 类的全路径
 - table : 表名（类名与表名一致，table可以省略）
 - catalog : 数据库名

【id标签】

```

14<class name="com.admiral.domain.Customer" table="cst_customer">
15  <!--建立类中的属性与表中的主键的映射关系 -->
16  <id name="cust_id" column="cust_id">
17    <!--主键的生成策略 -->
18    <generator class="native"></generator>
19  </id>
20
```

- 【id标签的配置】

- 标签用来建立类中的属性与表中的主键的对应关系
- 属性:
 - name : 类中的属性名
 - column : 表中的字段名（类中的属性名和表中的字段名如果一致，column可以省略）
 - length : 长度
 - type : 类型

【property标签】

```

21  <!-- 建立类中的普通属性与表中的字段的映射 -->
22  <property name="cust_name" column="cust_name" />
23  <property name="cust_source" column="cust_source" />
24  <property name="cust_industry" column="cust_industry" />
25  <property name="cust_level" column="cust_level" />
26  <property name="cust_phone" column="cust_phone" />
27  <property name="cust_mobile" column="cust_mobile" />
28
```

- 【property标签的配置】

- 标签用来建立类中的普通属性与表的字段的对应关系
- 属性:
 - name : 类中的属性名
 - column : 表中的字段名
 - length : 长度

- type : 类型
- not-null : 设置非空
- unique : 设置唯一

1.3.4.3 核心配置文件的配置

```
<session-factory>
    <!-- =====必要的配置===== -->
    <!-- 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">111111</property>
```

- 必须配置
 - 连接数据库的基本参数
 - 驱动类
 - url路径
 - 用户名
 - 密码
 - 方言

```
<!-- =====可选的配置===== -->
<!-- Hibernate 的方言:根据配置的方言生成对应的 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>
```

- 可选配置
 - 显示SQL : hibernate.show_sql
 - 格式化SQL : hibernate.format_sql
 - 自动建表 : hibernate.hbm2ddl.auto
 - none : 不使用hibernate的自动建表
 - create : 如果数据库中已经有表, 删除原有表, 重新创建, 如果没有表, 新建表。(测试)
 - create-drop : 如果数据库中已经有表, 删除原有表, 执行操作, 删除这个表。如果没有表, 新建一个, 使用完了删除该表。(测试)
 - update : 如果数据库中有表, 使用原有表, 如果没有表, 创建新表(更新表结构)
 - validate : 如果没有表, 不会创建表。只会使用数据库中原有的表。(校验映射和表结构)。
- 映射文件的引入
 - 引入映射文件的位置

```
<mapping resource="com/admiral/domain/Customer.hbm.xml"/>
```

1.3.5 Hibernate 的相关 API

1.3.5.1 Configuration:配置对象

Configuration主要用于Hibernate框架加载映射文件

Configuration

Configuration类的作用是对Hibernate进行配置，以及对它进行启动。在Hibernate的启动过程中，Configuration类的实例首先定位映射文档的位置，读取这些配置，然后创建一个SessionFactory对象。虽然Configuration类在整个Hibernate项目中只扮演着一个很小的角色，但它是启动hibernate时所遇到的第一个对象。

【加载核心配置文件】

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("com/admiral/domain/Customer.hbm.xml");
```

1.3.5.2 SessionFactory: Session 工厂对象

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();
```

SessionFactory具有以下特点：

- 它是线程安全的，它的同一个实例能够供多个线程共享。
- 它是重量级的，不能随意的创建和销毁它的实例。

由于SessionFactory的这些特点，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，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。

```
package com.admiral.utils;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtils {

    public static final Configuration configuration;
    public static final SessionFactory sessionFactory;

    static {
        configuration = new Configuration().configure();
        sessionFactory = configuration.buildSessionFactory();
    }

    /**
     * 提供获得 Session 的方法
     * @return
     */
    public Session openSession() {
        return sessionFactory.openSession();
    }
}
```

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内容如下

名称	修改日期	类型	大小
 c3p0-0.9.2.1	2014/4/28 20:30	Executable Jar File	414 KB
 hibernate-c3p0-5.0.7.Final	2016/1/13 12:42	Executable Jar File	12 KB
 mchange-commons-java-0.2.3.4	2014/4/28 20:30	Executable Jar File	568 KB

```
<!-- 配置C3P0连接池 -->
<property
name="connection.provider_class">org.hibernate.connection.C3P0ConnectionProvider
</property>
<!--在连接池中可用的数据库连接的最少数目 -->
<property name="c3p0.min_size">5</property>
<!--在连接池中所有数据库连接的最大数目 -->
<property name="c3p0.max_size">20</property>
<!--设定数据库连接的过期时间,以秒为单位, 如果连接池中的某个数据库连接处于空闲状态的时
间超过了timeout时间,就会从连接池中清除 -->
<property name="c3p0.timeout">120</property>
<!--每3000秒检查所有连接池中的空闲连接 以秒为单位 -->
<property name="c3p0.idle_test_period">3000</property>
```