目录

配套资料，免费下载
链接：https://pan.baidu.com/s/1TYbFKuqf8r3JFSX9bCiJBQ
提取码：u3g0
复制这段内容后打开百度网盘手机App，操作更方便哦

第一章 Hibernate5快速入门

1.1、Hibernate5概述

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进行了非常轻量级的对象封装，它将POJO与数据库表建立映射关系，Hibernate可以自动生成SQL语句并执行，使得Java程序员可以随心所欲的使用面向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。

1.2、Hibernate5下载

官方网址：hibernate orm

目录介绍：

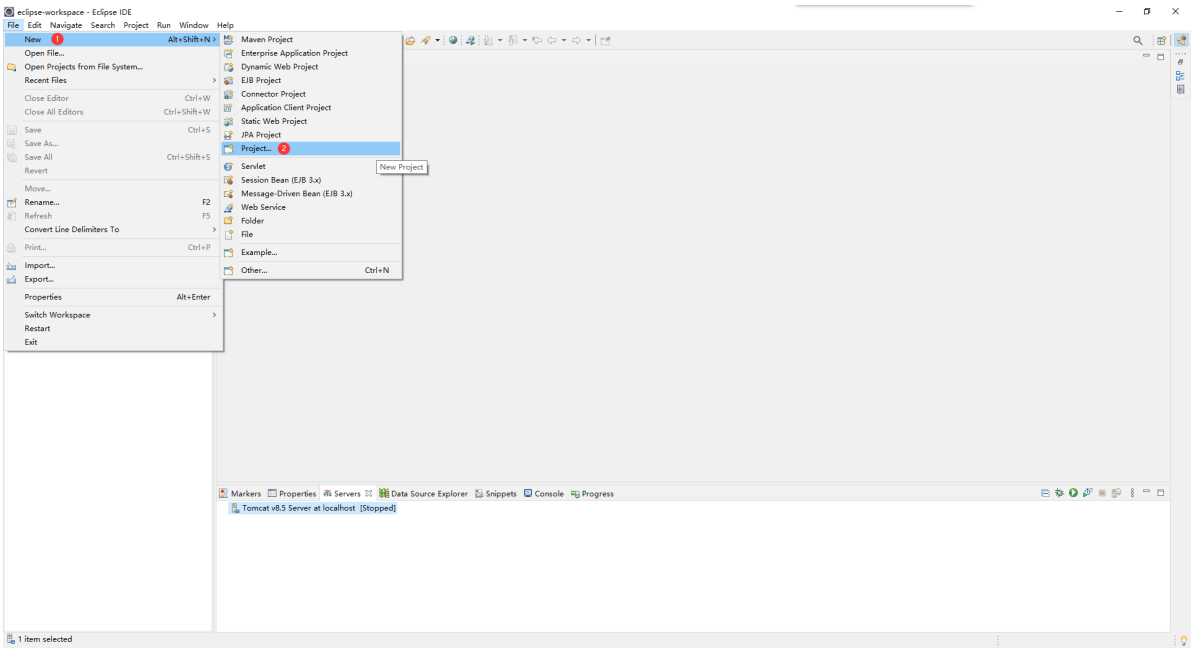
hibernate-release-5.0.7.Final文件夹介绍：

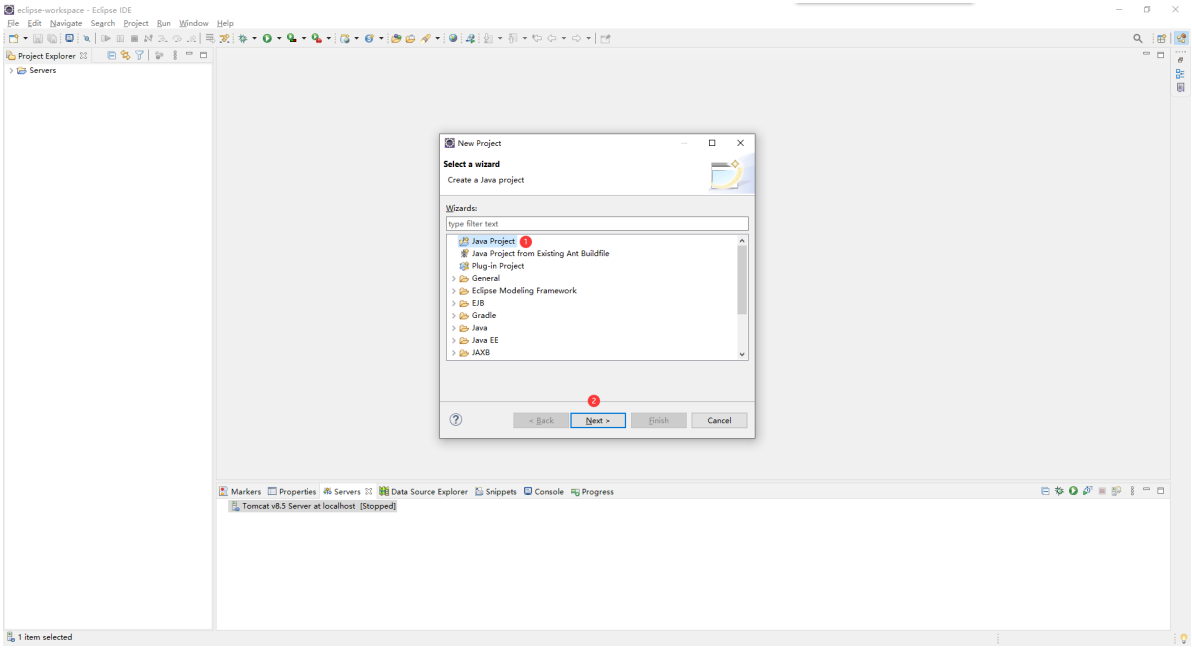
名称		修改日期	类型	大小
documentation	开发文档	2016/1/13 12:45	文件夹	
lib	相关依赖	2016/1/13 12:45	文件夹	
project	示例工程	2016/1/13 12:45	文件夹	
changelog.txt		2016/1/13 12:33	文本文档	46 KB
hibernate_logo.gif		2013/4/17 11:37	图片文件(.gif)	2 KB
lgpl.txt		2013/4/17 11:37	文本文档	26 KB

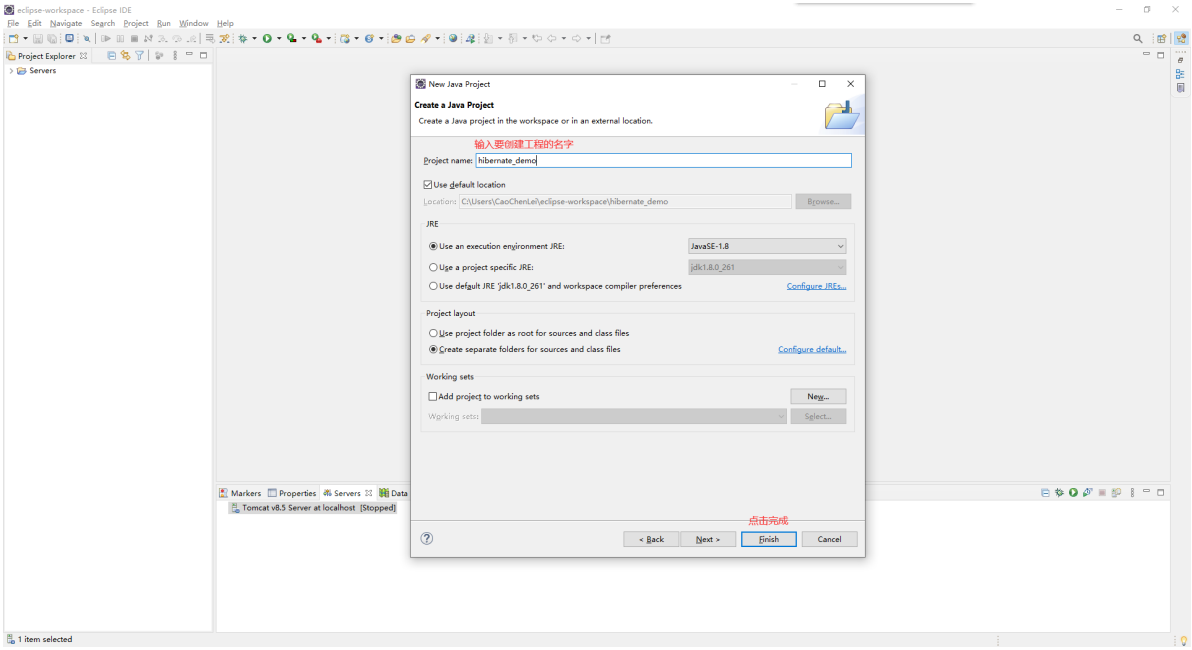
hibernate-release-5.0.7.Final\lib文件夹介绍：

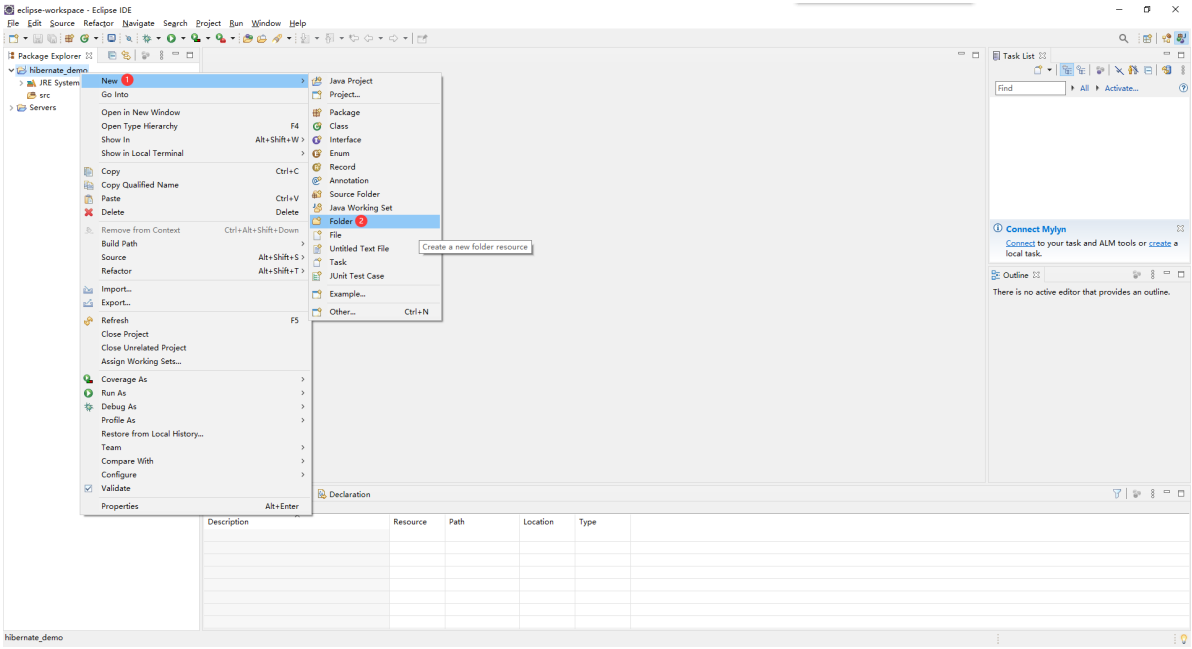
名称		修改日期	类型	大小
envers		2016/1/13 12:45	文件夹	
java8		2016/1/13 12:45	文件夹	
jpa		2016/1/13 12:45	文件夹	
jpa-metamodel-generator		2016/1/13 12:45	文件夹	
optional	可选包	2016/1/13 12:45	文件夹	
osgi		2016/1/13 12:45	文件夹	
required	必须包	2016/1/13 12:45	文件夹	

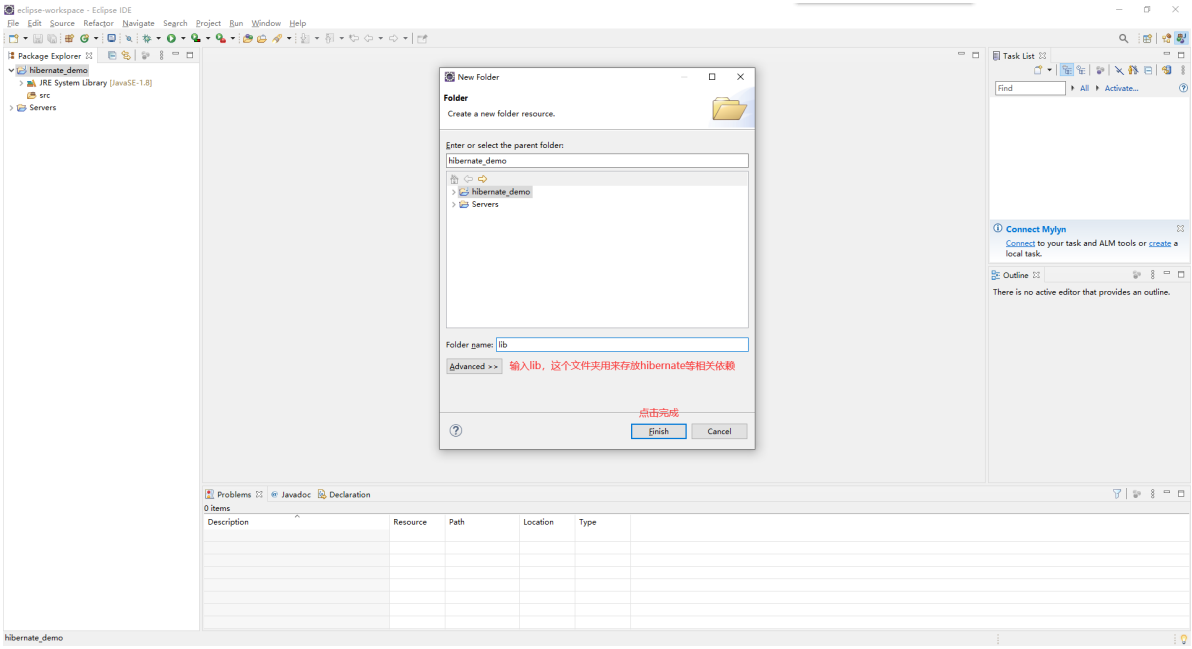
1.3、Hibernate5工程搭建




New Project
Select a wizard
Create a Java project
Wizards:
type filter text
Java Project
Java Project from Existing Ant Buildfile
Plug-in Project
General
Eclipse Modeling Framework
EJB
Gradle
Java
Java EE
JAXB
< Back
Next >
Finish
Cancel


New Java Project
Create a Java Project
Create a Java project in the workspace or in an external location.
输入要创建工程的名字
Project name: hibernate_demo
Use default location
JRE
Use an execution environment JRE:
JavaSE-1.8
Use a project specific JRE:
Use default JRE 'jdk1.8.0_261' and workspace compiler preferences
Configure JREs...
Project layout
Use project folder as root for sources and class files
Create separate folders for sources and class files
Configure default...
Working sets
Add project to working sets
New...
Select...
点击完成
< Back
Next >
Finish
Cancel


hibernate_demo
New
Go Into
Open in New Window
Open Type Hierarchy
Show In
Show in Local Terminal
Copy
Copy Qualified Name
Paste
Delete
Remove from Context
Build Path
Source
Refactor
Import...
Export...
Refresh
Close Project
Close Unrelated Project
Assign Working Sets...
Coverage As
Run As
Debug As
Profile As
Restore from Local History...
Team
Compare With
Configure
Validate
Properties
Java Project
Project...
Package
Class
Interface
Enum
Record
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
Task
JUnit Test Case
Example...
Other...
Create a new folder resource
Connect Mylyn
There is no active editor that provides an outline.
Declaration
Description
Resource
Path
Location
Type



把下面四个文件夹的内容全部拷贝到lib中

- hibernate-release-5.0.7.Final\lib\required

名称	修改日期	类型	大小
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

- hibernate-release-5.0.7.Final\lib\optional\c3p0

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

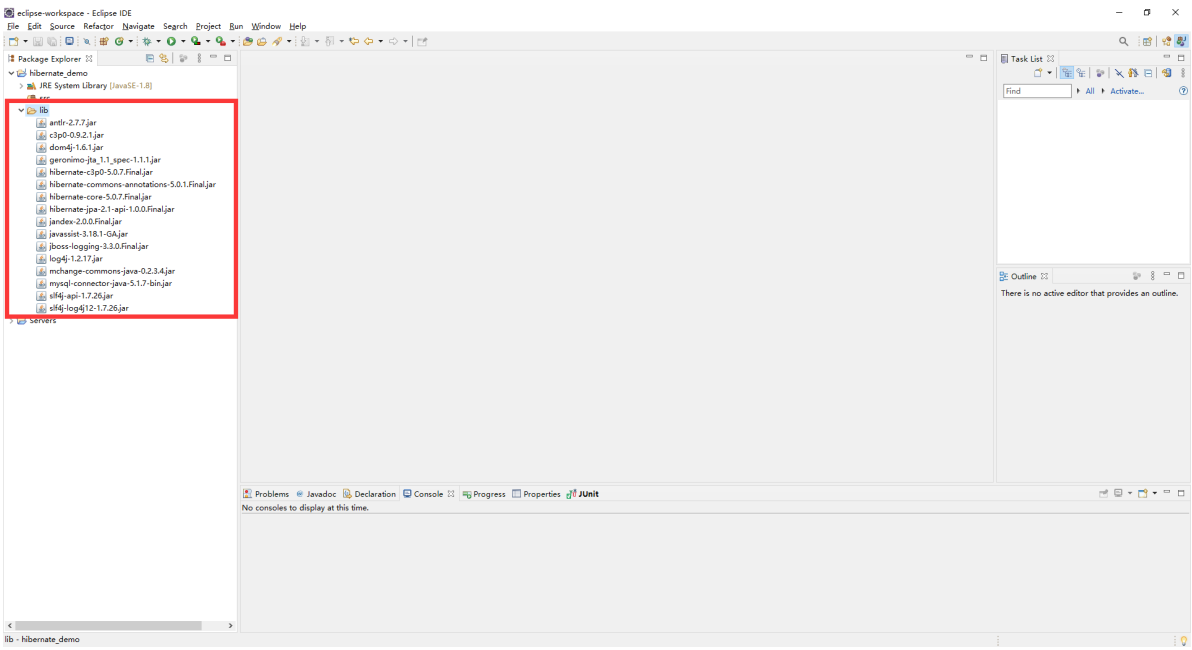
- 日志记录包

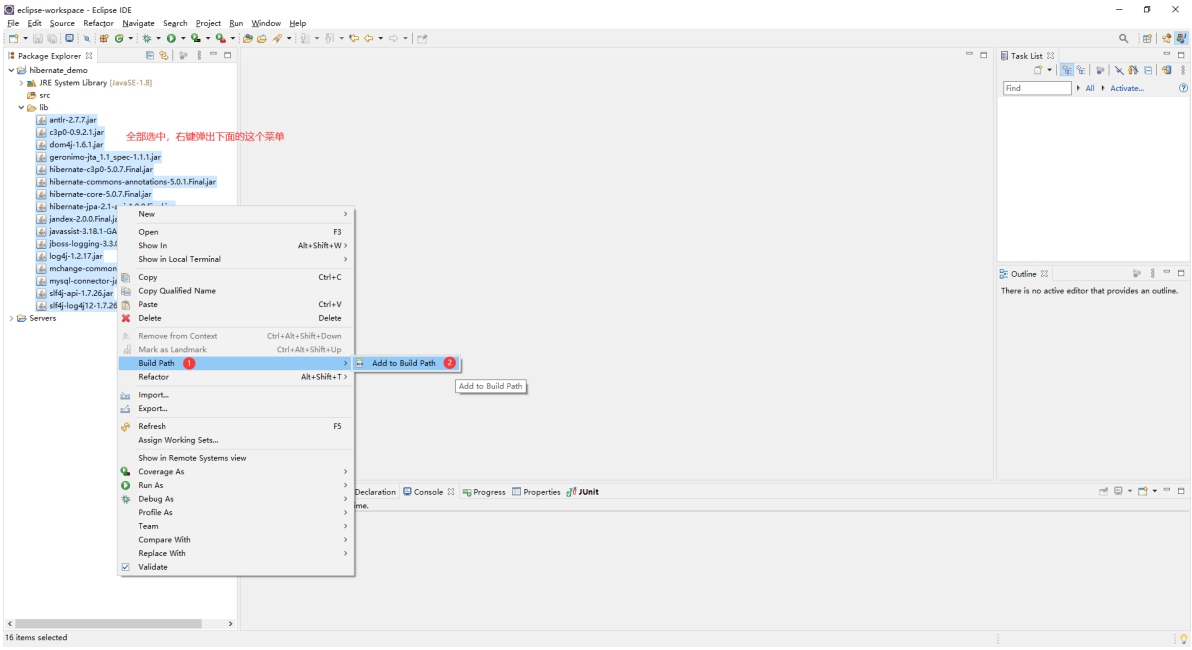
名称	修改日期	类型	大小
log4j-1.2.17.jar	2012/5/6 13:01	Executable Jar File	479 KB
slf4j-api-1.7.26.jar	2019/2/19 0:13	Executable Jar File	41 KB
slf4j-log4j12-1.7.26.jar	2019/2/19 0:13	Executable Jar File	12 KB

- 数据库驱动

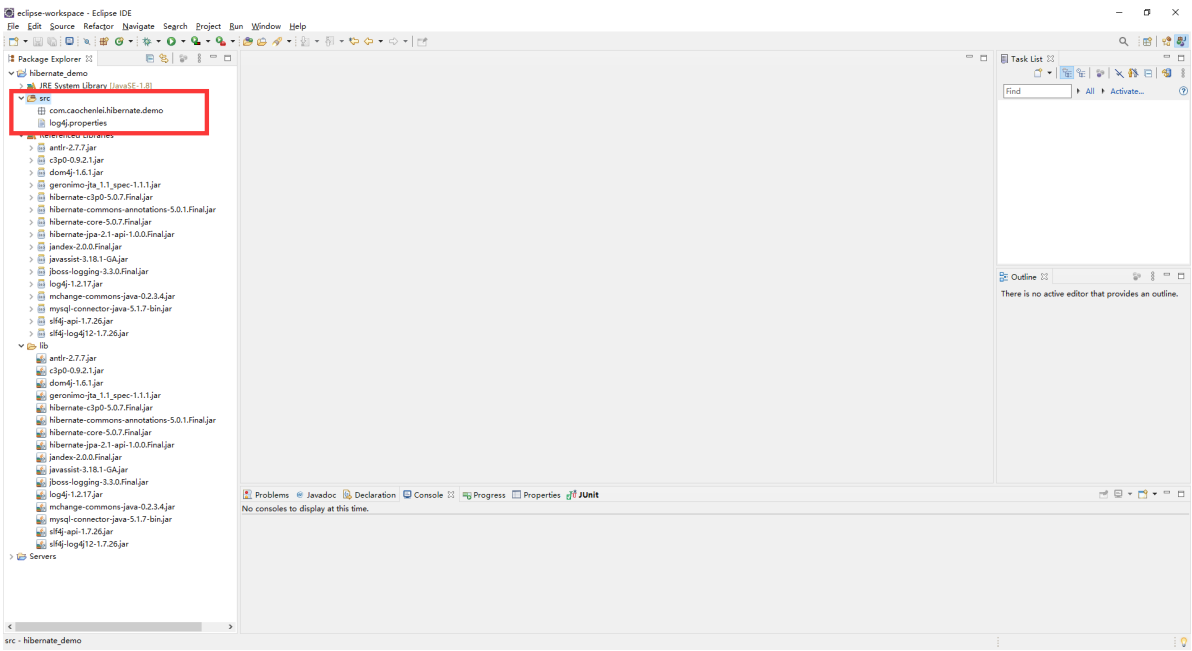
名称	修改日期	类型	大小
mysql-connector-java-5.1.7-bin.jar	2020/8/5 10:27	Executable Jar File	694 KB

把下面四个文件夹的内容全部拷贝到lib中后，全部选中，右键Build Path一下

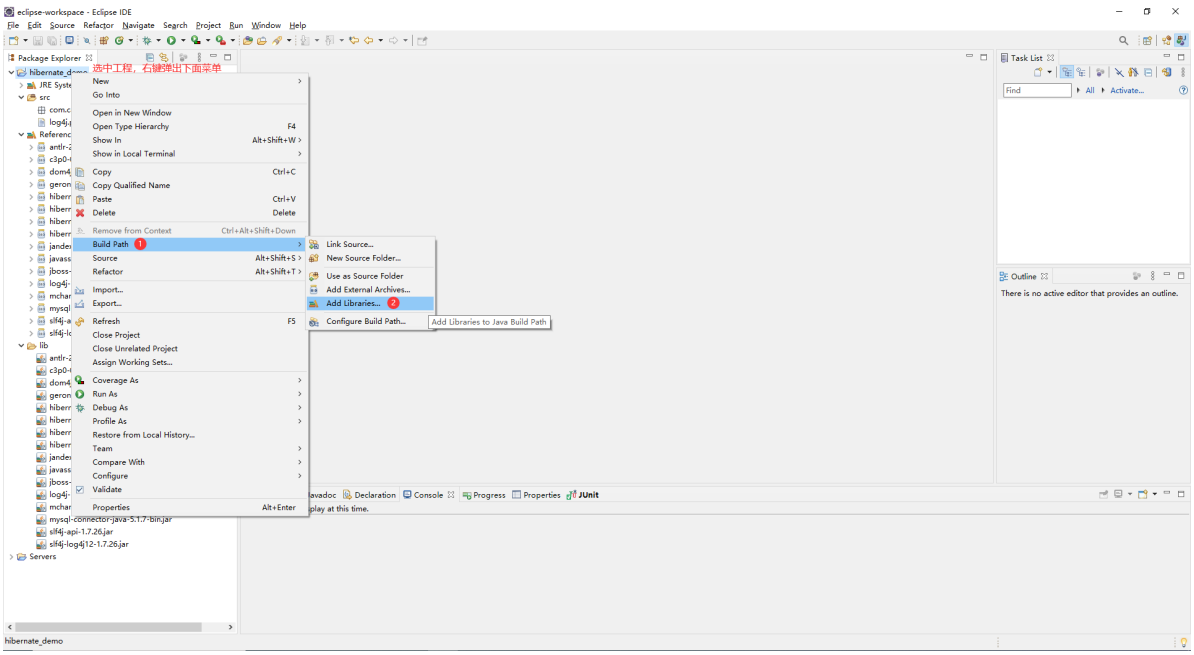


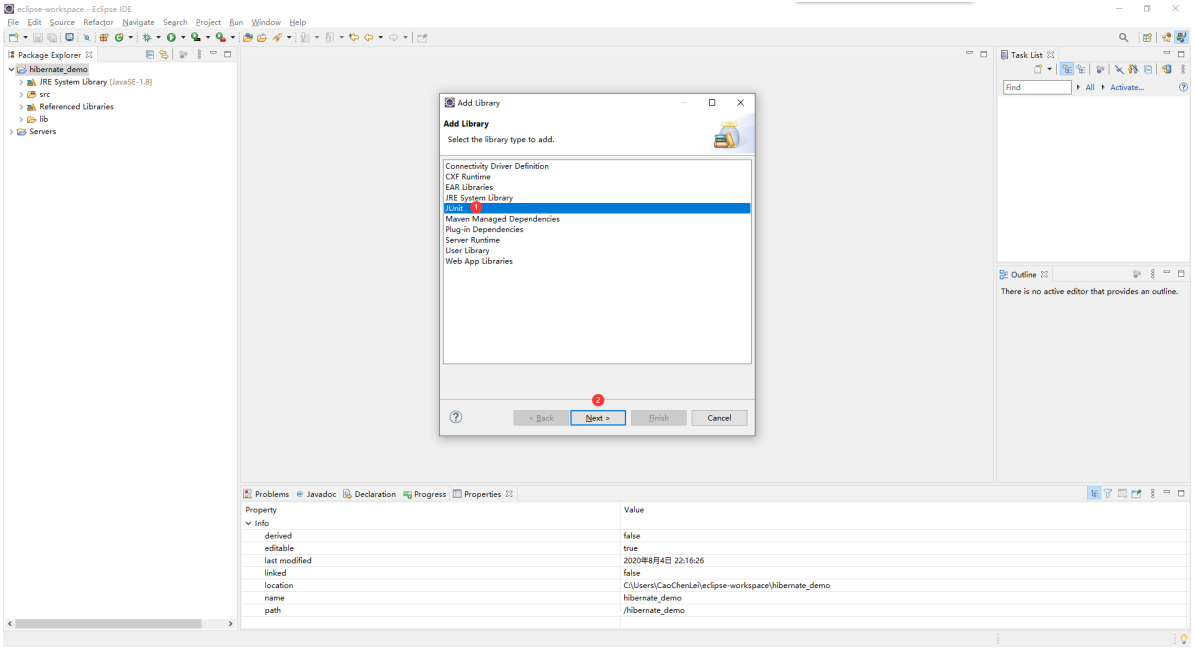


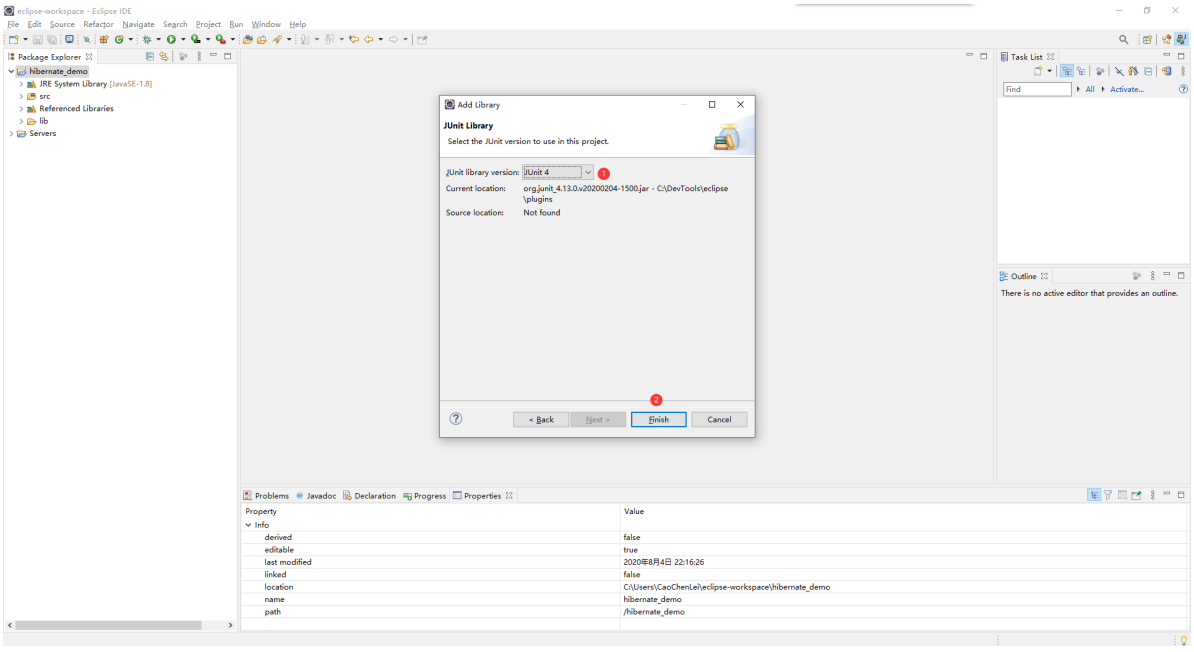
将日志记录配置文件复制到src中，并在src中创建com.caochenlei.hibernate.demo包，用于存放测试代码



由于我们需要进行单元测试，于是乎，我们需要给现在这个工程添加一个Junit4的依赖，具体操作如下图：





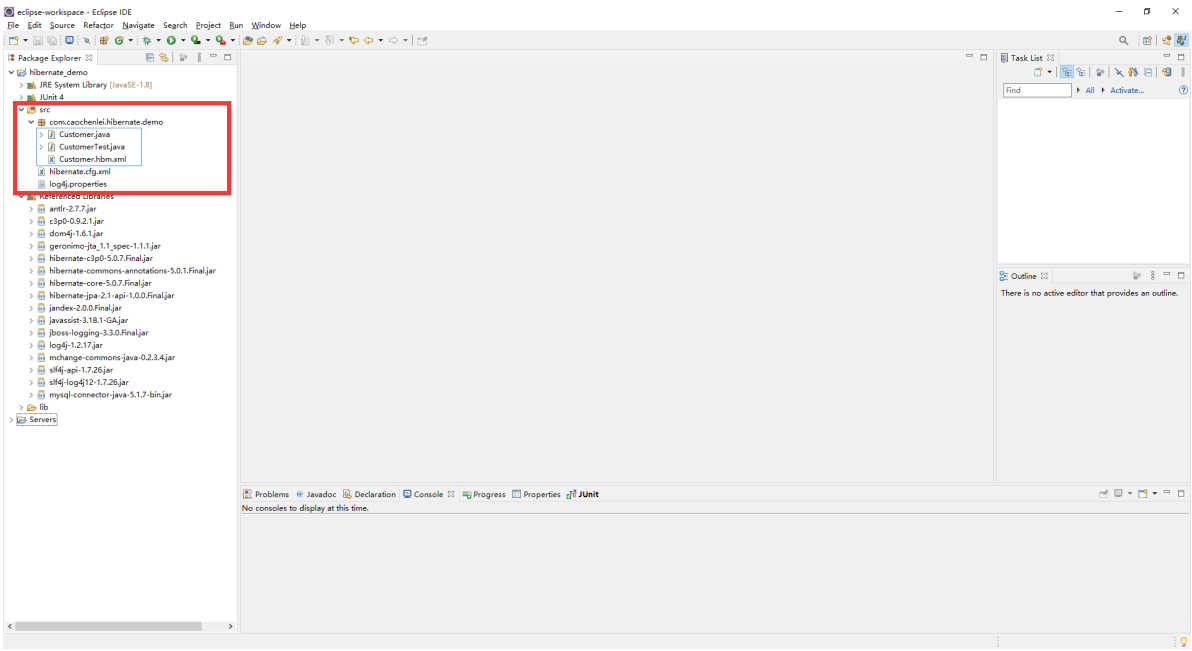


Java工程搭建好了以后，我们需要有一个数据库（mytest）和一张表（cst_customer）用于测试：

```
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',
  `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;
```

到此，准备工作就完成了，接下来就是我们的编程了

1.4、Hibernate5测试代码



Customer.java（全路径：/hibernate_demo/src/com/caochenlei/hibernate/demo/Customer.java）

```
package com.caochenlei.hibernate.demo;

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;
    }

    @Override
    public String toString() {
        return "Customer [cust_id=" + cust_id + ", cust_name=" + cust_name + ",
cust_source=" + cust_source
                + ", cust_industry=" + cust_industry + ", cust_level=" +
cust_level + ", cust_phone=" + cust_phone
                + ", cust_mobile=" + cust_mobile + "]";
    }
}
```

Customer.hbm.xml（全路径：/hibernate_demo/src/com/caochenlei/hibernate/demo/Customer.hbm.xml）

```
<?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>
    <!-- 建立类与表的映射关系 -->
    <class name="com.caochenlei.hibernate.demo.Customer" table="cst_customer">
        <!-- 设置主键字段映射 -->
        <id name="cust_id" column="cust_id">
            <generator class="native" />
        </id>

        <!-- 设置普通字段映射 -->
        <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>
```

hibernate.cfg.xml（全路径：/hibernate_demo/src/hibernate.cfg.xml）

```
<?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>
        <!-- 数据库的配置 -->
        <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
name="hibernate.connection.url">jdbc:mysql://127.0.0.1:3306/mytest</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">root</property>
        <property name="hibernate.connection.isolation">4</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property
name="hibernate.current_session_context_class">thread</property>

        <!-- 数据库连接池 -->
        <property
name="connection.provider_class">org.hibernate.connection.C3P0ConnectionProvider
</property>
        <property name="c3p0.min_size">5</property>
        <property name="c3p0.max_size">20</property>
        <property name="c3p0.timeout">120</property>
        <property name="c3p0.idle_test_period">3000</property>

        <!-- 其它的可选项 -->
```

```
        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.format_sql">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>

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

CustomerTest.java（全路径：/hibernate_demo/src/com/caochenlei/hibernate/demo/CustomerTest.java）

```
package com.caochenlei.hibernate.demo;

import java.util.Arrays;
import java.util.List;

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

public class CustomerTest {

    @Test
    public void testInsert() {
        Configuration configuration = new Configuration().configure();
        SessionFactory sessionFactory = configuration.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();

        Customer customer = new Customer();
        customer.setCust_name("李四");
        session.save(customer);

        tx.commit();
        session.close();
    }

    @Test
    public void testDelete() {
        Configuration configuration = new Configuration().configure();
        SessionFactory sessionFactory = configuration.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();

        Customer customer = new Customer();
        customer.setCust_id(1L);
        session.delete(customer);

        tx.commit();
        session.close();
    }
```

```
    @Test
    public void testUpdate() {
        Configuration configuration = new Configuration().configure();
        SessionFactory sessionFactory = configuration.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();

        Customer customer = new Customer();
        customer.setCust_id(2L);
        customer.setCust_name("张三");
        session.update(customer);

        tx.commit();
        session.close();
    }

    @Test
    public void testSaveOrUpdate() {
        Configuration configuration = new Configuration().configure();
        SessionFactory sessionFactory = configuration.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();

        Customer customer = new Customer();
        // 如果存在id就意味着更新，如果不存在id就意味着插入
        // customer.setCust_id(6L);
        customer.setCust_name("张三");
        session.saveOrUpdate(customer);

        tx.commit();
        session.close();
    }

    @Test
    public void testGet() {
        Configuration configuration = new Configuration().configure();
        SessionFactory sessionFactory = configuration.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();

        Customer customer = session.get(Customer.class, 4L);
        System.out.println(customer);

        tx.commit();
        session.close();
    }

    @Test
    public void testLoad() {
        Configuration configuration = new Configuration().configure();
        SessionFactory sessionFactory = configuration.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();

        Customer customer = session.load(Customer.class, 4L);
        System.out.println(customer);

        tx.commit();
```

```
        session.close();
    }

    @SuppressWarnings("unchecked")
    @Test
    public void testQuery() {
        Configuration configuration = new Configuration().configure();
        SessionFactory sessionFactory = configuration.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();

        Query query = session.createQuery("from Customer");
        List<Customer> list = query.list();
        for (Customer customer : list) {
            System.out.println(customer);
        }

        tx.commit();
        session.close();
    }

    @SuppressWarnings("unchecked")
    @Test
    public void testSQLQuery() {
        Configuration configuration = new Configuration().configure();
        SessionFactory sessionFactory = configuration.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();

        SQLQuery sqlQuery = session.createSQLQuery("select * from
cst_customer");
        List<Object[]> list = sqlQuery.list();
        for (Object[] customer : list) {
            System.out.println(Arrays.toString(customer));
        }

        tx.commit();
        session.close();
    }

}
```

1.5、Hibernate5工具类

通过上边的代码我们不难发现，很多代码都是重复的，这无疑增加了我们开发时代码量，所以我们需要将能抽取的部分抽取出来，由于Configuration和SessionFactory是线程安全的，且一个工程只需要有一个即可，所以我们把这一部分抽取出来，而Session是线程不安全的，我们不能抽取，只能使用局部变量的方式访问才不会出现线程安全的问题

HibernateUtil.java（全路径：/hibernate_demo/src/com/caochenlei/hibernate/utils/HibernateUtil.java）

```
package com.caochenlei.hibernate.utils;

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

```
public class HibernateUtil {

    private static Configuration configuration = null;
    private static SessionFactory sessionFactory = null;

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

    /**
     * 会话独立，需要关闭
     * @return
     */
    public static Session openSession() {
        return sessionFactory.openSession();
    }

    /**
     * 会话绑定，无需关闭
     * @return
     */
    public static Session getCurrentSession() {
        return sessionFactory.getCurrentSession();
    }

}
```

第二章 Hibernate5核心配置

2.1、两种配置方式

2.1.1、属性文件的方式（了解）

hibernate.properties（全路径：/hibernate_demo/src/hibernate.properties）

```
#数据库的配置
hibernate.connection.driver_class=com.mysql.jdbc.Driver
hibernate.connection.url=jdbc:mysql://127.0.0.1:3306/mytest
hibernate.connection.username=root
hibernate.connection.password=root
hibernate.connection.isolation=4
hibernate.dialect=org.hibernate.dialect.MySQLDialect
hibernate.current_session_context_class=thread

#数据库连接池
connection.provider_class=org.hibernate.connection.C3P0ConnectionProvider
c3p0.min_size=5
c3p0.max_size=20
c3p0.timeout=120
c3p0.idle_test_period=3000

#其它的可选项
hibernate.show_sql=true
hibernate.format_sql=true
hibernate.hbm2ddl.auto=update
```

CustomerPropertiesTest.java（全路径：/hibernate_demo/src/com/caochenlei/hibernate/demo/CustomerPropertiesTest.java）

```
package com.caochenlei.hibernate.demo;

import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;

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

public class CustomerPropertiesTest {

    @Test
    public void testMethod() {
        try {
            Configuration configuration = new Configuration();
            Properties properties = new Properties();
            InputStream inStream =
getClass().getClassLoader().getResourceAsStream("hibernate.properties");
            properties.load(inStream);
            configuration.addProperties(properties);

configuration.addResource("com/caochenlei/hibernate/demo/Customer.hbm.xml");

            SessionFactory sessionFactory = configuration.buildSessionFactory();
            Session session = sessionFactory.openSession();
            Transaction tx = session.beginTransaction();

            Customer customer = new Customer();
            customer.setCust_name("王五");
            session.save(customer);

            tx.commit();
            session.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

2.1.2、xml文件的方式（掌握）

hibernate.cfg.xml（全路径：/hibernate_demo/src/hibernate.cfg.xml）

```
<?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>
        <!-- 数据库的配置 -->
```

```
        <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
name="hibernate.connection.url">jdbc:mysql://127.0.0.1:3306/mytest</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">root</property>
        <property name="hibernate.connection.isolation">4</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property
name="hibernate.current_session_context_class">thread</property>

        <!-- 数据库连接池 -->
        <property
name="connection.provider_class">org.hibernate.connection.C3P0ConnectionProvider
</property>
        <property name="c3p0.min_size">5</property>
        <property name="c3p0.max_size">20</property>
        <property name="c3p0.timeout">120</property>
        <property name="c3p0.idle_test_period">3000</property>

        <!-- 其它的可选项 -->
        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.format_sql">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>

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

CustomerXMLTest.java（全路径：/hibernate_demo/src/com/caochenlei/hibernate/demo/CustomerXMLTest.java）

```
package com.caochenlei.hibernate.demo;

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

public class CustomerXMLTest {

    @Test
    public void testMethod() {
        Configuration configuration = new Configuration().configure();

        SessionFactory sessionFactory = configuration.buildSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();

        Customer customer = new Customer();
        customer.setCust_name("小七");
        session.save(customer);

        tx.commit();
```

```
        session.close();
    }

}
```

2.2、核心配置讲解

2.2.1、核心的配置

1. **数据库的配置**
    - hibernate.connection.driver_class：数据库驱动名称
    - hibernate.connection.url：数据库连接地址
    - hibernate.connection.username：数据库账户
    - hibernate.connection.password：数据库密码
    - hibernate.dialect：数据库方言
        - MySQL方言：org.hibernate.dialect.MySQLDialect
        - Oracle方言：org.hibernate.dialect.Oracle10gDialect
        - SQL Server方言：org.hibernate.dialect.SQLServerDialect
    - hibernate.connection.isolation：数据库隔离级别
        - 0：TRANSACTION_NONE
        - 1：TRANSACTION_READ_UNCOMMITTED
        - 2：TRANSACTION_READ_COMMITTED（Oracle常用）
        - 4：TRANSACTION_REPEATABLE_READ（MySQL常用）
        - 8：TRANSACTION_SERIALIZABLE
    - hibernate.current_session_context_class：获取当前会话的上下文
        - thread：对象的生命周期与本地线程绑定（常用）
        - jta：对象的生命周期与JTA事务绑定
        - managed：委托程序来管理session的生命周期
2. **数据库连接池**
    - connection.provider_class：设定数据库连接池提供方，一般使用C3P0
    - c3p0.min_size：在连接池中可用的数据库连接的最少数目
    - c3p0.max_size：在连接池中可用的数据库连接的最大数目
    - c3p0.timeout：设定数据库连接的过期时间，以秒为单位
    - c3p0.idle_test_period：每隔多少秒检查连接池中的空闲连接，以秒为单位
3. **其它的可选项**
    - hibernate.show_sql：显示sql语句
    - hibernate.format_sql：格式化sql语句
    - hibernate.hbm2ddl.auto：是否自动建表
        - 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，重新创建，然后执行操作，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，如果数据库中没有表，创建新表。（可以自动更新表结构，常用）
        - validate：如果数据库中没有表，不会创建表，只会使用数据库中原有的表。（可以校验映射和表结构，常用）
4. **加载映射文件**

-

2.2.2、映射的配置

1. **class标签的配置：用来建立类与表的映射关系**
   - 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（可以省略）
2. **id标签的配置：用来建立类中的属性与表中的主键的对应关系**
   - 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（用于自动建表的时候指定数据的长度，如果不指定则使用默认值，可以省略）
   - type：类型（用于自动建表的时候指定数据的类型，如果不指定，Hibernate可以自动帮你转换，可以省略）
     - 第一种：Java中的类型，例如：java.lang.String
     - 第二种：Hibernate中的类型，例如：string
     - 第三种：sql中的类型，需要把column属性去掉，在property中写一个子标签

   主键的生成策略如下：
   - **increment**：使用的是hibernate中提供的自动增长机制（先查询id最大值，然后再加1），适用于short、int、long类型的主键，在单线程中使用
   - **identity**：使用的是数据库底层的自动增长机制（设置主键的AUTO_INCREMENT），适用于short、int、long类型的主键，在单线程/多线程中使用，适用于Mysql、Sql Server，但不适用于Oracle
   - **sequence**：使用的是序列方式，适用于short、int、long类型的主键，在单线程/多线程中使用，不适用于Mysql、Sql Server，但适用于Oracle
   - **native**：本地策略，可以智能的在**identity**和**sequence**之间进行切换
   - **uuid**：使用的是hibernate中的随机方式生成字符串主键，适用于字符串类型的主键
   - **assigned**：hibernate放弃外键的管理，通过开发者自己编写程序来控制
3. **property标签的配置：用来建立类中的普通属性与表的字段的对应关系**
   - 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（用于自动建表的时候指定数据的长度，如果不指定则使用默认值，可以省略）
   - type：类型（用于自动建表的时候指定数据的类型，如果不指定，Hibernate可以自动帮你转换，可以省略）
     - 第一种：Java中的类型，例如：java.lang.String
     - 第二种：Hibernate中的类型，例如：string
     - 第三种：sql中的类型，需要把column属性去掉，在property中写一个子标签
   - not-null：设置非空
   - unique：设置唯一

2.3、核心对象讲解

Hibernate的API一共有6个，分别为：Configuration、SessionFactory、Session、Transaction、Query、Criteria。通过这些接口，可以对持久化对象进行存取、事务控制。

2.3.1、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。

2.3.2、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。

2.3.3、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。

2.3.4、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。

2.3.5、Query

Query接口让你方便地对数据库及持久对象进行查询，它可以有两种表达方式：HQL语言或本地数据库的SQL语句。Query经常被用来绑定查询参数、限制查询记录数量，并最终执行查询操作。

2.3.6、Criteria

Criteria接口与Query接口非常类似，允许创建并执行面向对象的标准化查询。值得注意的是Criteria接口也是轻量级的，它不能在Session之外使用。

第三章 Hibernate5持久化类

3.1、持久化类的概述

持久化：将内存中的对象持久化到数据库中的过程，而Hibernate就是用来完成持久化的框架

持久化类：一个Java对象与数据库的表建立了映射关系，那么这个类在Hibernate中称为持久化类，简而言之：持久化类 = Java类 + 映射文件

3.2、持久化类的规则

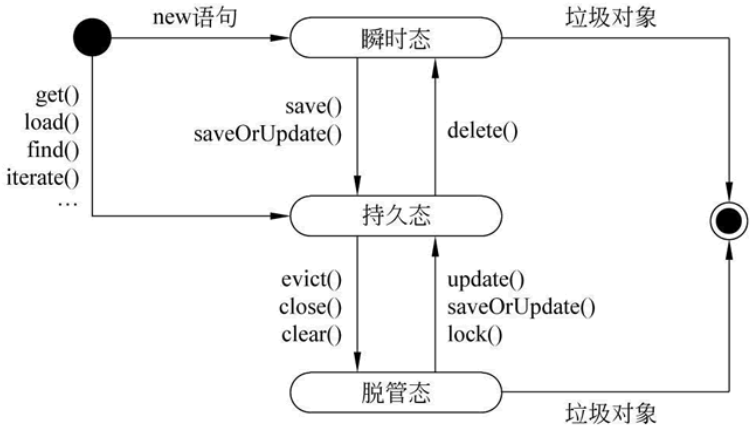
1. 对持久化类提供无参构造方法
2. 让持久化类的属性私有化，对私有属性提供public的get或者set方法
3. 让持久化类中的属性尽量使用包装类型而不是基本类型
4. 对持久化类提供一个唯一标识OID与数据库主键对应
5. 对持久化类不能使用final修饰

3.3、持久化类的状态

持久化类一共有三种不同的状态，它们分别如下：

- 瞬时态（transient）：这种对象没有唯一的标识OID，没有被Session管理
- 持久态（persistent）：这种对象有唯一的标识OID，并且被Session管理
- 脱管态（detached）：这种对象有唯一的标识OID，没有被Session管理

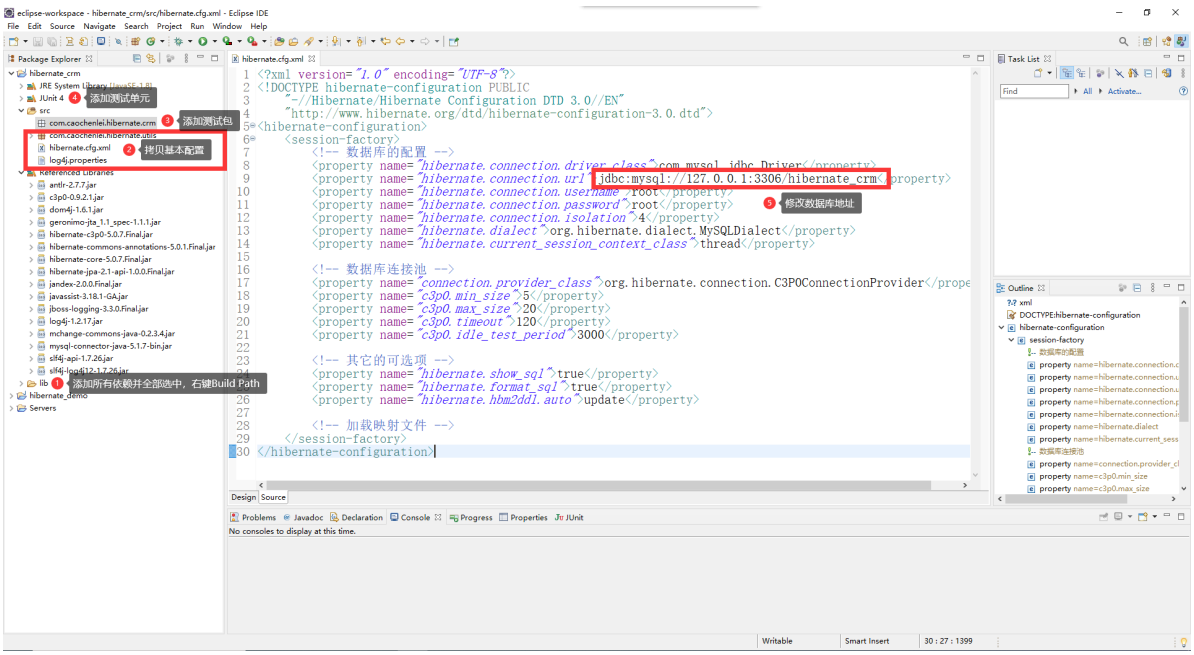
3.4、持久化类的转换



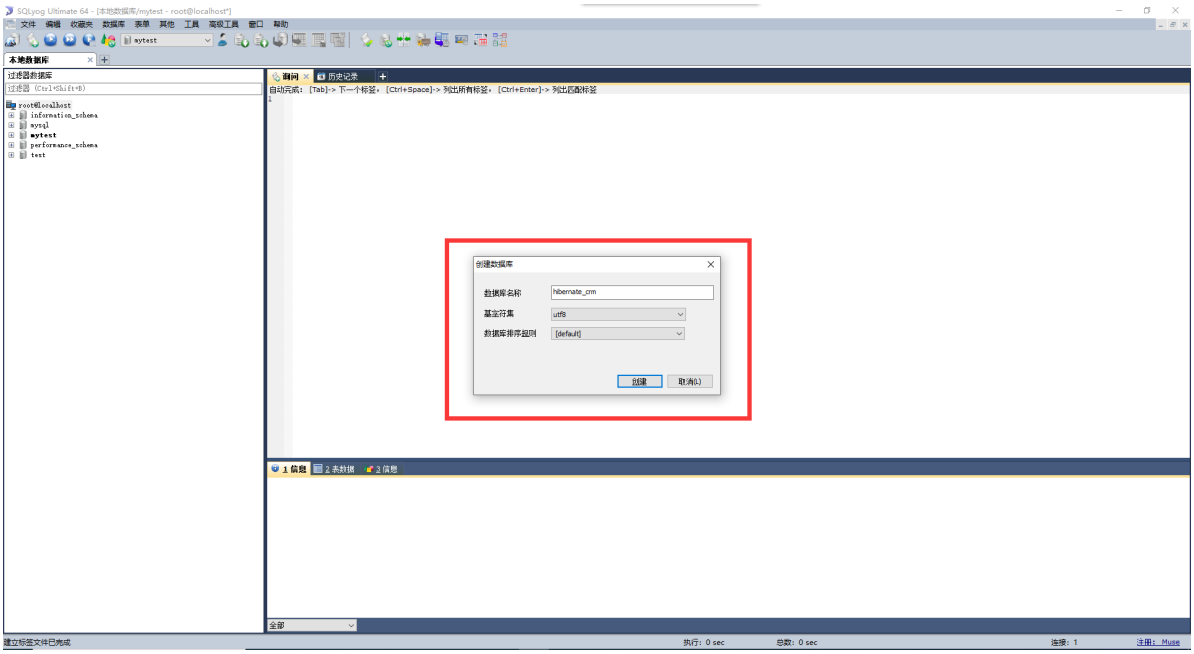
- **瞬时态对象**
  - 获得：直接new对象
  - 状态转换：
    - 瞬时态->持久态：save、saveOrUpdate
    - 瞬时态->托管态：直接设置OID
- **持久态对象**（特性：可以自动更新数据库）
  - 获得：get、load、find、iterate
  - 状态转换
    - 持久态->瞬时态：delete
    - 持久态->托管态：evict、close、clear
- **托管态对象**
  - 获得：直接new对象，然后设置OID
  - 状态转换：
    - 托管态->持久态：update、saveOrUpdate、lock
    - 托管态->瞬时态：直接设置OID为null

第四章 Hibernate5映射关系

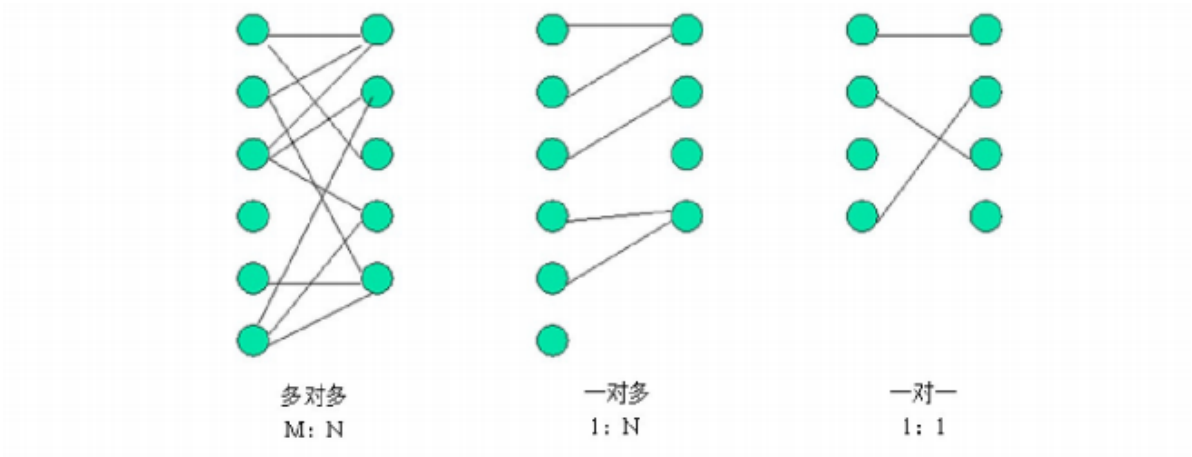
本章我们要对复杂的映射关系进行学习，按照前边的学习重新搭建一个项目：hibernate_crm



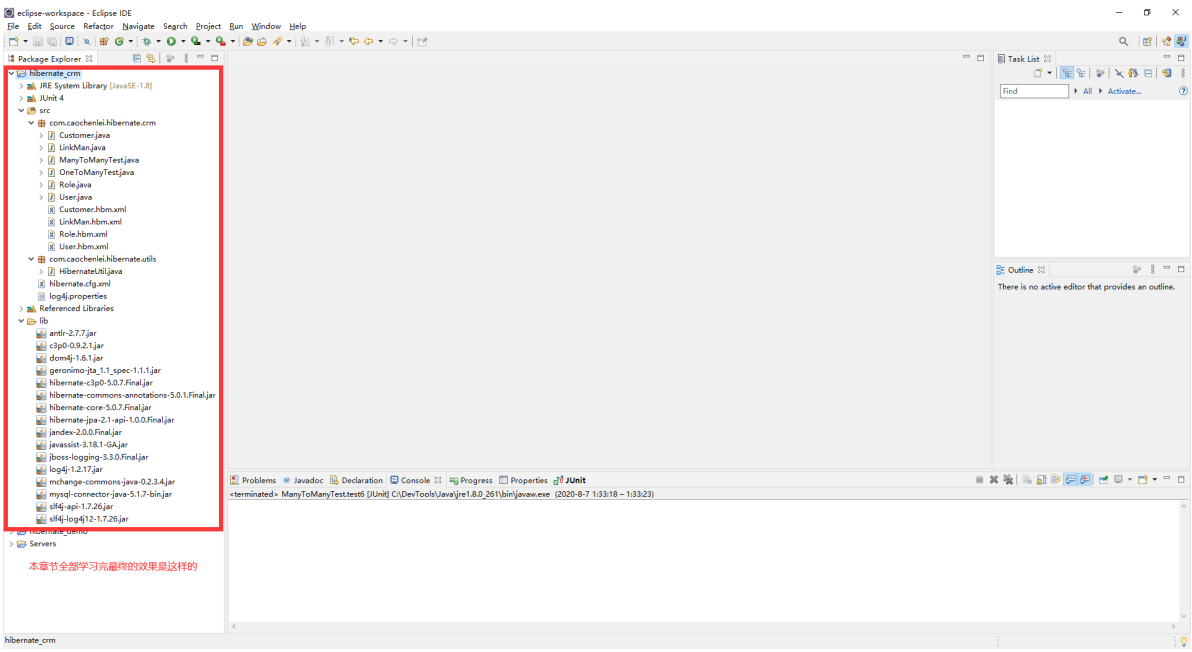
打开数据库，重新创建一个数据库，数据库表，下文会一一给出，在此我们先不创建



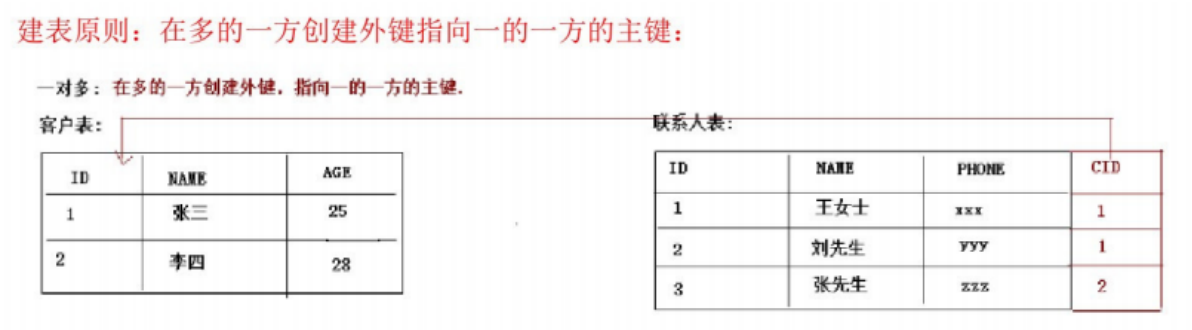
Hibernate框架实现了ORM的思想，将关系数据库中表的数据映射成对象，使开发人员把对数据库的操作转化为对对象的操作，Hibernate的关联关系映射主要包括多表的映射配置、数据的增加、删除等。数据库中多表之间存在着三种关系，也就是系统设计中的三种实体关系。如图所示：



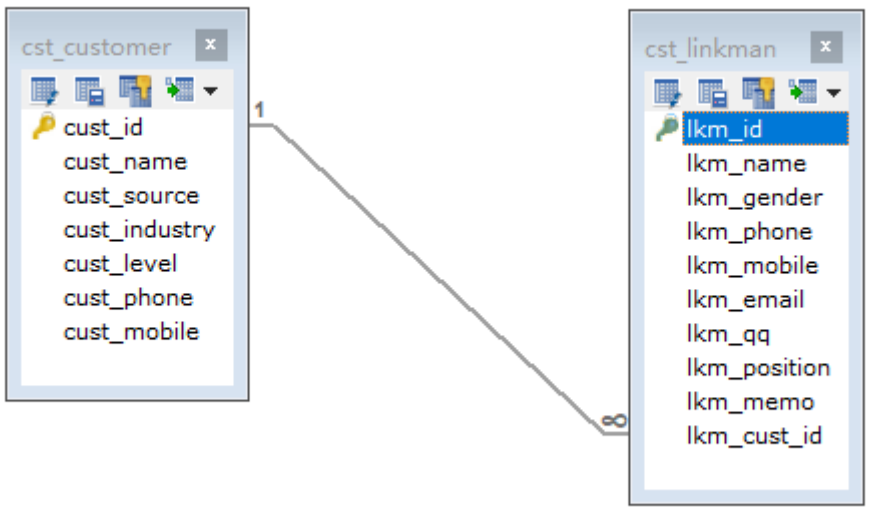
温馨提示：本章节全部学习完最终的效果是这样的



4.1、一对多关联



4.1.1、创建数据表



crm_cst_customer（客户表）

```sql
CREATE TABLE `cst_customer` (
  `cust_id` BIGINT(32) NOT NULL AUTO_INCREMENT COMMENT '客户编号(主键)',
  `cust_name` VARCHAR(32) DEFAULT NULL COMMENT '客户名称',
  `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;
```

crm_cst_linkman（联系人表）

```sql
CREATE TABLE `cst_linkman` (
  `lkm_id` BIGINT(32) NOT NULL AUTO_INCREMENT COMMENT '联系人编号(主键)',
  `lkm_name` VARCHAR(16) DEFAULT NULL COMMENT '联系人姓名',
  `lkm_gender` CHAR(1) DEFAULT NULL COMMENT '联系人性别',
  `lkm_phone` VARCHAR(16) DEFAULT NULL COMMENT '联系人办公电话',
  `lkm_mobile` VARCHAR(16) DEFAULT NULL COMMENT '联系人手机',
  `lkm_email` VARCHAR(64) DEFAULT NULL COMMENT '联系人邮箱',
  `lkm_qq` VARCHAR(16) DEFAULT NULL COMMENT '联系人qq',
  `lkm_position` VARCHAR(16) DEFAULT NULL COMMENT '联系人职位',
  `lkm_memo` VARCHAR(512) DEFAULT NULL COMMENT '联系人备注',
  `lkm_cust_id` BIGINT(32) NOT NULL COMMENT '客户id',
  PRIMARY KEY (`lkm_id`),
  KEY `FK_cst_linkman_lkm_cust_id` (`lkm_cust_id`),
  CONSTRAINT `FK_cst_linkman_lkm_cust_id` FOREIGN KEY (`lkm_cust_id`) REFERENCES
`cst_customer` (`cust_id`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=INNODB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

4.1.2、编写实体对象

Customer.java（全路径：/hibernate_crm/src/com/caochenlei/hibernate/crm/Customer.java）

```java
package com.caochenlei.hibernate.crm;

import java.util.HashSet;
import java.util.Set;

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;
    // 放置的是多的一方的集合
    private Set<LinkMan> linkMans = new HashSet<LinkMan>();

    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;
    }

    public Set<LinkMan> getLinkMans() {
        return linkMans;
    }

    public void setLinkMans(Set<LinkMan> linkMans) {
        this.linkMans = linkMans;
    }
}
```

LinkMan.java（全路径：/hibernate_crm/src/com/caochenlei/hibernate/crm/LinkMan.java）

```
package com.caochenlei.hibernate.crm;

public class LinkMan {
    private Long lkm_id;
    private String lkm_name;
    private String lkm_gender;
    private String lkm_phone;
    private String lkm_mobile;
    private String lkm_email;
    private String lkm_qq;
    private String lkm_position;
    private String lkm_memo;
    // 放置的是一的一方的对象
    private Customer customer;

    public Long getLkm_id() {
        return lkm_id;
    }

    public void setLkm_id(Long lkm_id) {
        this.lkm_id = lkm_id;
    }

    public String getLkm_name() {
        return lkm_name;
    }

    public void setLkm_name(String lkm_name) {
        this.lkm_name = lkm_name;
    }

    public String getLkm_gender() {
        return lkm_gender;
    }

    public void setLkm_gender(String lkm_gender) {
        this.lkm_gender = lkm_gender;
    }

    public String getLkm_phone() {
        return lkm_phone;
    }

    public void setLkm_phone(String lkm_phone) {
        this.lkm_phone = lkm_phone;
    }

    public String getLkm_mobile() {
        return lkm_mobile;
    }

    public void setLkm_mobile(String lkm_mobile) {
        this.lkm_mobile = lkm_mobile;
    }
```

```
    public String getLkm_email() {
        return lkm_email;
    }

    public void setLkm_email(String lkm_email) {
        this.lkm_email = lkm_email;
    }

    public String getLkm_qq() {
        return lkm_qq;
    }

    public void setLkm_qq(String lkm_qq) {
        this.lkm_qq = lkm_qq;
    }

    public String getLkm_position() {
        return lkm_position;
    }

    public void setLkm_position(String lkm_position) {
        this.lkm_position = lkm_position;
    }

    public String getLkm_memo() {
        return lkm_memo;
    }

    public void setLkm_memo(String lkm_memo) {
        this.lkm_memo = lkm_memo;
    }

    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
    }
}
```

4.1.3、编写映射文件

Customer.hbm.xml（全路径：/hibernate_crm/src/com/caochenlei/hibernate/crm/Customer.hbm.xml）

```
<?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>
    <class name="com.caochenlei.hibernate.crm.Customer" table="cst_customer">
        <!-- 设置主键字段映射 -->
        <id name="cust_id" column="cust_id">
            <generator class="native"/>
        </id>
```

```
        <!-- 设置普通字段映射 -->
        <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"/>
        <!-- 配置一对多的映射：放置的是多的一方的集合 -->
        <!--
            set标签：
                * name       ：多的一方的对象集合的属性名称
                * cascade    ：级联（保存更新、删除）
                * inverse    ：放弃外键维护权（14亿人民记住1个主席名称很容易，但是你让1个主席记住14亿人名称很难，所以一方放弃）
        -->
        <set name="linkMans" cascade="save-update,delete" inverse="true">
            <!--
                 key标签：
                    * column：多的一方的外键的名称
             -->
            <key column="lkm_cust_id"/>
            <!--
                one-to-many标签：
                    * class：多的一方的类的全路径
             -->
            <one-to-many class="com.caochenlei.hibernate.crm.LinkMan"/>
        </set>
    </class>
</hibernate-mapping>
```

LinkMan.hbm.xml（全路径：/hibernate_crm/src/com/caochenlei/hibernate/crm/LinkMan.hbm.xml）

```
<?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>
    <class name="com.caochenlei.hibernate.crm.LinkMan" table="cst_linkman">
        <!-- 设置主键字段映射 -->
        <id name="lkm_id" column="lkm_id">
            <generator class="native"/>
        </id>
        <!-- 设置普通字段映射 -->
        <property name="lkm_name" column="lkm_name"/>
        <property name="lkm_gender" column="lkm_gender"/>
        <property name="lkm_phone" column="lkm_phone"/>
        <property name="lkm_mobile" column="lkm_mobile"/>
        <property name="lkm_email" column="lkm_email"/>
        <property name="lkm_qq" column="lkm_qq"/>
        <property name="lkm_position" column="lkm_position"/>
        <property name="lkm_memo" column="lkm_memo"/>
        <!-- 配置多对一的关系：放置的是一的一方的对象 -->
        <!--
            many-to-one标签：
                * name       ：一的一方的对象的属性名称
                * class      ：一的一方的类的全路径
```

```
                * column     : 在多的一方的表的外键的名称
                * cascade    : 级联（保存更新、删除）
         -->
        <many-to-one name="customer"
class="com.caochenlei.hibernate.crm.Customer" column="lkm_cust_id"
cascade="save-update,delete"/>
    </class>
</hibernate-mapping>
```

```
<!-- 加载映射文件 -->
<mapping resource="com/caochenlei/hibernate/crm/Customer.hbm.xml"/>
<mapping resource="com/caochenlei/hibernate/crm/LinkMan.hbm.xml"/>
```

4.1.4、编写测试文件

OneToManyTest.java（全路径：/hibernate_crm/src/com/caochenlei/hibernate/crm/OneToManyTest.java）

```
package com.caochenlei.hibernate.crm;

import org.hibernate.Session;
import org.hibernate.Transaction;
import org.junit.Test;

import com.caochenlei.hibernate.utils.HibernateUtil;

public class OneToManyTest {

    /**
     * 不使用级联，保存客户，保存联系人
     * 去掉Customer.hbm.xml映射文件中的cascade
     * 去掉LinkMan.hbm.xml映射文件中的cascade
     * 修改hibernate.cfg.xml中的hibernate.hbm2ddl.auto=create
     */
    @Test
    public void test1() {
        Session session = HibernateUtil.getCurrentSession();
        Transaction tx = session.beginTransaction();

        // 创建两个客户
        Customer customer1 = new Customer();
        customer1.setCust_name("张三");
        Customer customer2 = new Customer();
        customer2.setCust_name("李四");

        // 创建三个联系人
        LinkMan linkMan1 = new LinkMan();
        linkMan1.setLkm_name("凤姐");
        LinkMan linkMan2 = new LinkMan();
        linkMan2.setLkm_name("如花");
        LinkMan linkMan3 = new LinkMan();
        linkMan3.setLkm_name("旺财");

        customer1.getLinkMans().add(linkMan1);
        customer1.getLinkMans().add(linkMan2);
        customer2.getLinkMans().add(linkMan3);
        linkMan1.setCustomer(customer1);
        linkMan2.setCustomer(customer1);
        linkMan3.setCustomer(customer2);
```

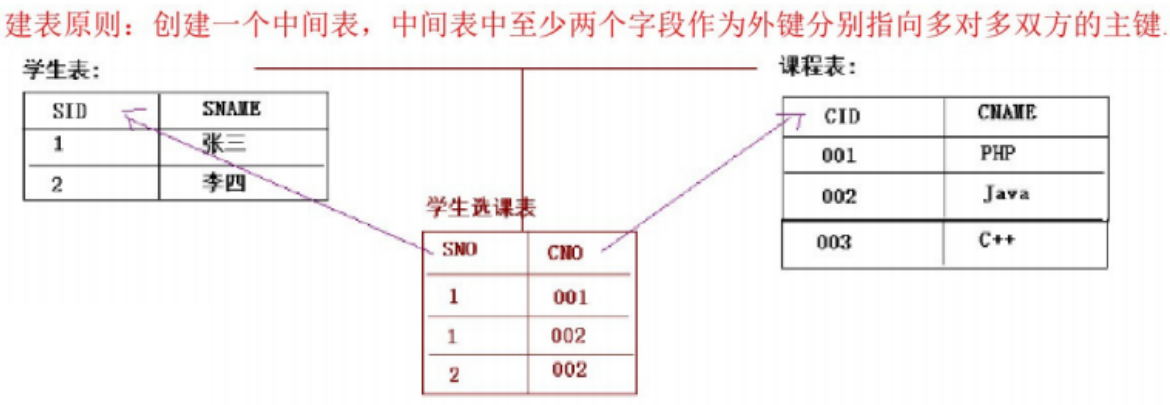
```
        session.save(customer1);
        session.save(customer2);
        session.save(linkMan1);
        session.save(linkMan2);
        session.save(linkMan3);

        tx.commit();
    }

    /**
     * 使用级联，插入客户，级联保存联系人
     * 加上Customer.hbm.xml映射文件中的cascade
     * 去掉LinkMan.hbm.xml映射文件中的cascade
     * 修改hibernate.cfg.xml中的hibernate.hbm2ddl.auto=create
     */
    @Test
    public void test2() {
        Session session = HibernateUtil.getCurrentSession();
        Transaction tx = session.beginTransaction();

        // 创建两个客户
        Customer customer1 = new Customer();
        customer1.setCust_name("张三");
        Customer customer2 = new Customer();
        customer2.setCust_name("李四");

        // 创建三个联系人
        LinkMan linkMan1 = new LinkMan();
        linkMan1.setLkm_name("凤姐");
        LinkMan linkMan2 = new LinkMan();
        linkMan2.setLkm_name("如花");
        LinkMan linkMan3 = new LinkMan();
        linkMan3.setLkm_name("旺财");

        customer1.getLinkMans().add(linkMan1);
        customer1.getLinkMans().add(linkMan2);
        customer2.getLinkMans().add(linkMan3);
        linkMan1.setCustomer(customer1);
        linkMan2.setCustomer(customer1);
        linkMan3.setCustomer(customer2);

        session.save(customer1);
        session.save(customer2);

        tx.commit();
    }

    /**
     * 使用级联，插入联系人，级联保存客户
     * 去掉Customer.hbm.xml映射文件中的cascade
     * 加上LinkMan.hbm.xml映射文件中的cascade
     * 修改hibernate.cfg.xml中的hibernate.hbm2ddl.auto=create
     */
    @Test
    public void test3() {
        Session session = HibernateUtil.getCurrentSession();
        Transaction tx = session.beginTransaction();
```

```
        // 创建两个客户
        Customer customer1 = new Customer();
        customer1.setCust_name("张三");
        Customer customer2 = new Customer();
        customer2.setCust_name("李四");

        // 创建三个联系人
        LinkMan linkMan1 = new LinkMan();
        linkMan1.setLkm_name("凤姐");
        LinkMan linkMan2 = new LinkMan();
        linkMan2.setLkm_name("如花");
        LinkMan linkMan3 = new LinkMan();
        linkMan3.setLkm_name("旺财");

        customer1.getLinkMans().add(linkMan1);
        customer1.getLinkMans().add(linkMan2);
        customer2.getLinkMans().add(linkMan3);
        linkMan1.setCustomer(customer1);
        linkMan2.setCustomer(customer1);
        linkMan3.setCustomer(customer2);

        session.save(linkMan1);
        session.save(linkMan2);
        session.save(linkMan3);

        tx.commit();
    }

    /**
     * 测试对象导航
     * 加上Customer.hbm.xml映射文件中的cascade
     * 加上LinkMan.hbm.xml映射文件中的cascade
     * 修改hibernate.cfg.xml中的hibernate.hbm2ddl.auto=create
     */
    @Test
    public void test4() {
        Session session = HibernateUtil.getCurrentSession();
        Transaction tx = session.beginTransaction();

        Customer customer = new Customer();
        customer.setCust_name("李兵");

        LinkMan linkMan1 = new LinkMan();
        linkMan1.setLkm_name("凤姐");
        LinkMan linkMan2 = new LinkMan();
        linkMan2.setLkm_name("如花");
        LinkMan linkMan3 = new LinkMan();
        linkMan3.setLkm_name("芙蓉");

        linkMan1.setCustomer(customer);
        customer.getLinkMans().add(linkMan2);
        customer.getLinkMans().add(linkMan3);

        //session.save(linkMan1); // 发送几条insert语句  4条
        //session.save(customer); // 发送几条insert语句  3条
        //session.save(linkMan2); // 发送几条insert语句  1条
```

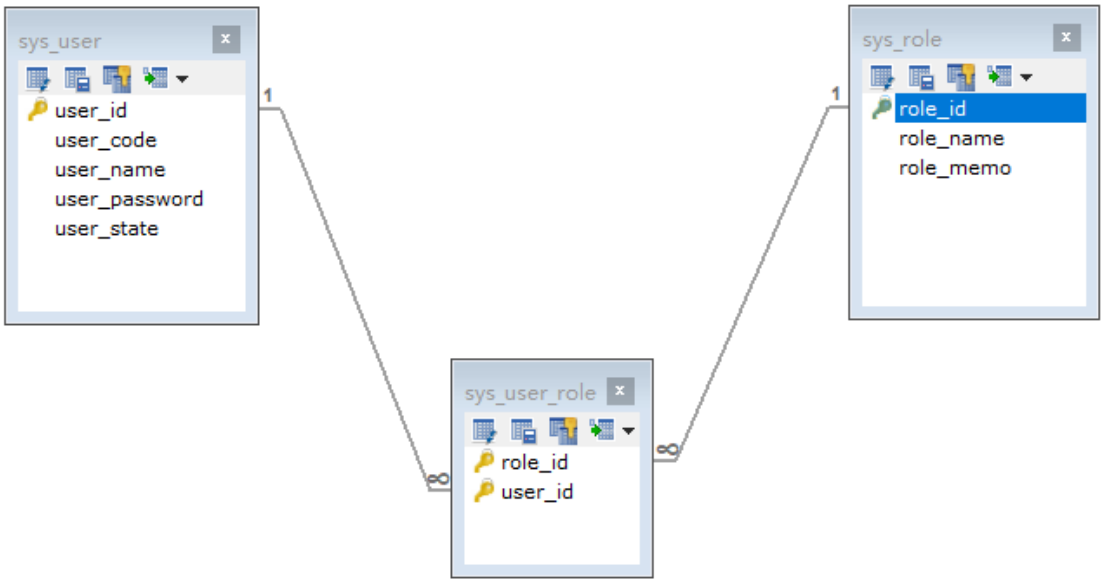
```
        tx.commit();
    }

    /**
     * 测试级联删除，删除客户，级联删除联系人
     * 加上Customer.hbm.xml映射文件中的cascade
     * 去掉LinkMan.hbm.xml映射文件中的cascade
     * 先运行test1进行建表，方便测试，然后执行test5
     * 修改hibernate.cfg.xml中的hibernate.hbm2ddl.auto=update
     */
    @Test
    public void test5() {
        Session session = HibernateUtil.getCurrentSession();
        Transaction tx = session.beginTransaction();

        Customer customer = session.get(Customer.class, 1L);
        session.delete(customer);

        tx.commit();
    }

    /**
     * 测试级联删除，删除联系人，级联删除客户
     * 去掉Customer.hbm.xml映射文件中的cascade
     * 加上LinkMan.hbm.xml映射文件中的cascade
     * 先运行test1进行建表，方便测试，然后执行test6
     * 修改hibernate.cfg.xml中的hibernate.hbm2ddl.auto=update
     */
    @Test
    public void test6() {
        Session session = HibernateUtil.getCurrentSession();
        Transaction tx = session.beginTransaction();

        LinkMan linkMan = session.get(LinkMan.class, 1L);
        session.delete(linkMan);

        tx.commit();
    }

    /**
     * 案例模拟演示，将2号联系人原来归1号客户，现在改为2号客户
     * 加上Customer.hbm.xml映射文件中的cascade
     * 加上LinkMan.hbm.xml映射文件中的cascade
     * 先运行test1进行建表，方便测试，然后执行test7
     * 修改hibernate.cfg.xml中的hibernate.hbm2ddl.auto=update
     */
    @Test
    public void test7() {
        Session session = HibernateUtil.getCurrentSession();
        Transaction tx = session.beginTransaction();

        // 查找2号联系人
        LinkMan linkMan = session.get(LinkMan.class, 2L);

        // 查找2号客户
        Customer customer = session.get(Customer.class, 2L);

        // 将2号联系人原来归1号客户，现在改为2号客户
```

```
        linkMan.setCustomer(customer);
        customer.getLinkMans().add(linkMan);

        session.save(linkMan);

        tx.commit();
    }

}
```

4.2、多对多关联



4.2.1、创建数据表



sys_user（用户表）

```
CREATE TABLE `sys_user` (
  `user_id` bigint(32) NOT NULL AUTO_INCREMENT COMMENT '用户id',
  `user_code` varchar(32) DEFAULT NULL COMMENT '用户账号',
  `user_name` varchar(64) DEFAULT NULL COMMENT '用户名称',
  `user_password` varchar(32) DEFAULT NULL COMMENT '用户密码',
  `user_state` char(1) DEFAULT NULL COMMENT '用户状态',
  PRIMARY KEY (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

sys_role（角色表）

```
CREATE TABLE `sys_role` (
  `role_id` bigint(32) NOT NULL AUTO_INCREMENT COMMENT '角色id',
  `role_name` varchar(32) DEFAULT NULL COMMENT '角色名称',
  `role_memo` varchar(128) DEFAULT NULL COMMENT '角色备注',
  PRIMARY KEY (`role_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

sys_user_role（中间表）

```
CREATE TABLE `sys_user_role` (
  `role_id` bigint(32) NOT NULL COMMENT '角色id',
  `user_id` bigint(32) NOT NULL COMMENT '用户id',
  PRIMARY KEY (`role_id`,`user_id`),
  KEY `FK_user_role_user_id` (`user_id`),
  CONSTRAINT `FK_user_role_role_id` FOREIGN KEY (`role_id`) REFERENCES
`sys_role` (`role_id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `FK_user_role_user_id` FOREIGN KEY (`user_id`) REFERENCES
`sys_user` (`user_id`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

4.2.2、编写实体对象

User.java（全路径：/hibernate_crm/src/com/caochenlei/hibernate/crm/User.java）

```
package com.caochenlei.hibernate.crm;

import java.util.HashSet;
import java.util.Set;

public class User {
    private Long user_id;
    private String user_code;
    private String user_name;
    private String user_password;
    private String user_state;
    // 放置的是角色的集合
    private Set<Role> roles = new HashSet<Role>();

    public Long getUser_id() {
        return user_id;
    }

    public void setUser_id(Long user_id) {
        this.user_id = user_id;
    }

    public String getUser_code() {
        return user_code;
    }

    public void setUser_code(String user_code) {
        this.user_code = user_code;
    }
```

```
    public String getUser_name() {
        return user_name;
    }

    public void setUser_name(String user_name) {
        this.user_name = user_name;
    }

    public String getUser_password() {
        return user_password;
    }

    public void setUser_password(String user_password) {
        this.user_password = user_password;
    }

    public String getUser_state() {
        return user_state;
    }

    public void setUser_state(String user_state) {
        this.user_state = user_state;
    }

    public Set<Role> getRoles() {
        return roles;
    }

    public void setRoles(Set<Role> roles) {
        this.roles = roles;
    }
}
```

Role.java（全路径：/hibernate_crm/src/com/caochenlei/hibernate/crm/Role.java）

```
package com.caochenlei.hibernate.crm;

import java.util.HashSet;
import java.util.Set;

public class Role {
    private Long role_id;
    private String role_name;
    private String role_memo;
    // 放置的是用户的集合
    private Set<User> users = new HashSet<User>();

    public Long getRole_id() {
        return role_id;
    }

    public void setRole_id(Long role_id) {
        this.role_id = role_id;
    }

    public String getRole_name() {
        return role_name;
```

```
    }

    public void setRole_name(String role_name) {
        this.role_name = role_name;
    }

    public String getRole_memo() {
        return role_memo;
    }

    public void setRole_memo(String role_memo) {
        this.role_memo = role_memo;
    }

    public Set<User> getUsers() {
        return users;
    }

    public void setUsers(Set<User> users) {
        this.users = users;
    }
}
```

4.2.3、编写映射文件

User.hbm.xml（全路径：/hibernate_crm/src/com/caochenlei/hibernate/crm/User.hbm.xml）

```
<?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>
    <class name="com.caochenlei.hibernate.crm.User" table="sys_user">
        <!-- 设置主键字段映射 -->
        <id name="user_id" column="user_id">
            <generator class="native"/>
        </id>
        <!-- 设置普通字段映射 -->
        <property name="user_code" column="user_code"/>
        <property name="user_name" column="user_name"/>
        <property name="user_password" column="user_password"/>
        <property name="user_state" column="user_state"/>
        <!-- 建立与角色的多对多的映射关系 -->
        <!--
            set标签：
                * name      ：对方的集合的属性名称
                * table     ：多对多的关系需要使用中间表，放的是中间表的名称
                * cascade   ：级联（保存更新、删除）
         -->
        <set name="roles" table="sys_user_role" cascade="save-update,delete">
            <!--
                key标签：
                    * column    ：当前的对象对应中间表的外键的名称
             -->
            <key column="user_id"/>
            <!--
                many-to-many标签：
```

```
                    * class      : 对方的类的全路径
                    * column     : 对方的对象在中间表中的外键的名称
             -->
            <many-to-many class="com.caochenlei.hibernate.crm.Role"
column="role_id"/>
        </set>
    </class>
</hibernate-mapping>
```

Role.hbm.xml（全路径：/hibernate_crm/src/com/caochenlei/hibernate/crm/Role.hbm.xml）

```
<?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>
    <class name="com.caochenlei.hibernate.crm.Role" table="sys_role">
        <!-- 设置主键字段映射 -->
        <id name="role_id" column="role_id">
            <generator class="native"/>
        </id>
        <!-- 设置普通字段映射 -->
        <property name="role_name" column="role_name"/>
        <property name="role_memo" column="role_memo"/>
        <!-- 建立与用户的多对多的映射关系 -->
        <!--
            set标签:
                * name       : 对方的集合的属性名称。
                * table      : 多对多的关系需要使用中间表，放的是中间表的名称
                * cascade    : 级联（保存更新、删除）
                * inverse    : 放弃外键维护权（被动方放弃）
         -->
        <set name="users" table="sys_user_role" cascade="save-update,delete"
inverse="true">
            <!--
                key标签:
                    * column     : 当前的对象对应中间表的外键的名称
             -->
            <key column="role_id"/>
            <!--
                many-to-many标签:
                    * class      : 对方的类的全路径
                    * column     : 对方的对象在中间表中的外键的名称
             -->
            <many-to-many class="com.caochenlei.hibernate.crm.User"
column="user_id"/>
        </set>
    </class>
</hibernate-mapping>
```

```
<mapping resource="com/caochenlei/hibernate/crm/User.hbm.xml"/>
<mapping resource="com/caochenlei/hibernate/crm/Role.hbm.xml"/>
```

4.2.4、编写测试文件

OneToManyTest.java（全路径：/hibernate_crm/src/com/caochenlei/hibernate/crm/OneToManyTest.java）

```
package com.caochenlei.hibernate.crm;

import org.hibernate.Session;
import org.hibernate.Transaction;
import org.junit.Test;

import com.caochenlei.hibernate.utils.HibernateUtil;

public class ManyToManyTest {

    /**
     * 不使用级联，保存用户，保存角色
     * 去掉User.hbm.xml映射文件中的cascade
     * 去掉Role.hbm.xml映射文件中的cascade
     * 修改hibernate.cfg.xml中的hibernate.hbm2ddl.auto=create
     */
    @Test
    public void test1() {
        Session session = HibernateUtil.getCurrentSession();
        Transaction tx = session.beginTransaction();

        // 创建两个用户
        User user1 = new User();
        user1.setUser_name("赵洪");
        User user2 = new User();
        user2.setUser_name("李兵");

        // 创建三个角色
        Role role1 = new Role();
        role1.setRole_name("研发部");
        Role role2 = new Role();
        role2.setRole_name("市场部");
        Role role3 = new Role();
        role3.setRole_name("公关部");

        role1.getUsers().add(user1);
        role2.getUsers().add(user1);
        role2.getUsers().add(user2);
        role3.getUsers().add(user2);
        user1.getRoles().add(role1);
        user1.getRoles().add(role2);
        user2.getRoles().add(role2);
        user2.getRoles().add(role3);

        session.save(user1);
        session.save(user2);
        session.save(role1);
        session.save(role2);
        session.save(role3);

        tx.commit();
    }
```

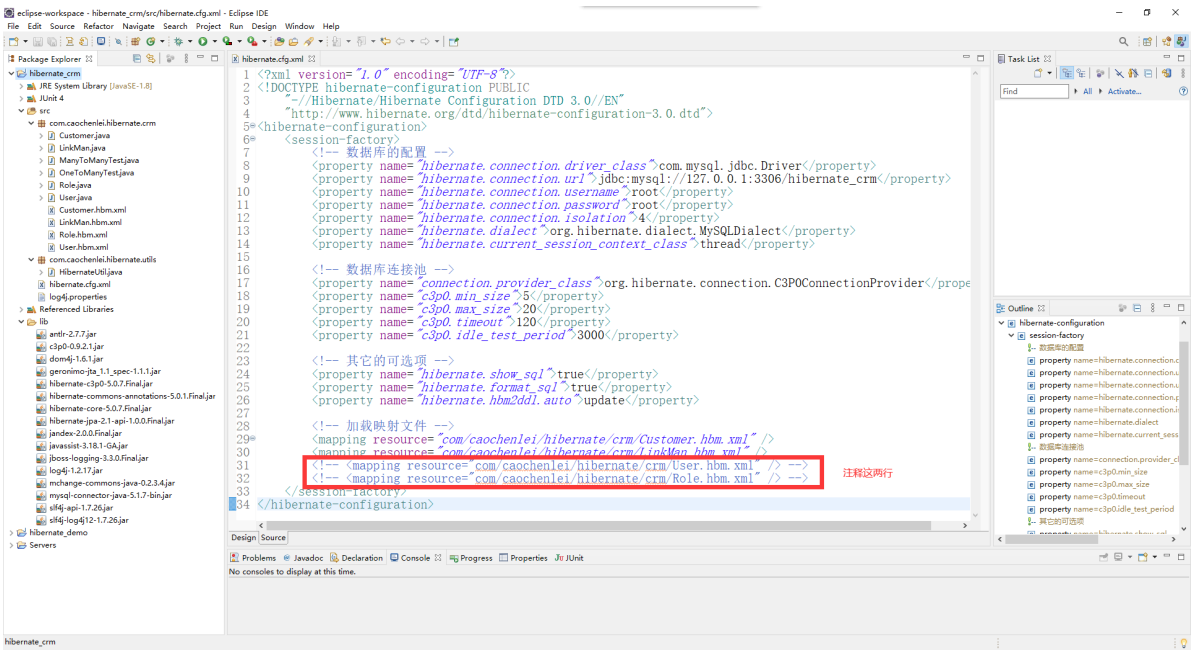
```java
/**
 * 使用级联，保存用户，级联保存角色
 * 加上User.hbm.xml映射文件中的cascade
 * 去掉Role.hbm.xml映射文件中的cascade
 * 修改hibernate.cfg.xml中的hibernate.hbm2ddl.auto=create
 */
@Test
public void test2() {
    Session session = HibernateUtil.getCurrentSession();
    Transaction tx = session.beginTransaction();

    // 创建两个用户
    User user1 = new User();
    user1.setUser_name("赵洪");
    User user2 = new User();
    user2.setUser_name("李兵");

    // 创建三个角色
    Role role1 = new Role();
    role1.setRole_name("研发部");
    Role role2 = new Role();
    role2.setRole_name("市场部");
    Role role3 = new Role();
    role3.setRole_name("公关部");

    role1.getUsers().add(user1);
    role2.getUsers().add(user1);
    role2.getUsers().add(user2);
    role3.getUsers().add(user2);
    user1.getRoles().add(role1);
    user1.getRoles().add(role2);
    user2.getRoles().add(role2);
    user2.getRoles().add(role3);

    session.save(user1);
    session.save(user2);

    tx.commit();
}

/**
 * 使用级联，保存角色，级联保存用户
 * 去掉User.hbm.xml映射文件中的cascade
 * 加上Role.hbm.xml映射文件中的cascade
 * 修改hibernate.cfg.xml中的hibernate.hbm2ddl.auto=create
 */
@Test
public void test3() {
    Session session = HibernateUtil.getCurrentSession();
    Transaction tx = session.beginTransaction();

    // 创建两个用户
    User user1 = new User();
    user1.setUser_name("赵洪");
    User user2 = new User();
    user2.setUser_name("李兵");
```

```
        // 创建三个角色
        Role role1 = new Role();
        role1.setRole_name("研发部");
        Role role2 = new Role();
        role2.setRole_name("市场部");
        Role role3 = new Role();
        role3.setRole_name("公关部");

        role1.getUsers().add(user1);
        role2.getUsers().add(user1);
        role2.getUsers().add(user2);
        role3.getUsers().add(user2);
        user1.getRoles().add(role1);
        user1.getRoles().add(role2);
        user2.getRoles().add(role2);
        user2.getRoles().add(role3);

        session.save(role1);
        session.save(role2);
        session.save(role3);

        tx.commit();
    }

    /**
     * 使用级联，删除用户，级联删除角色
     * 加上User.hbm.xml映射文件中的cascade
     * 去掉Role.hbm.xml映射文件中的cascade
     * 先运行test1进行建表，方便测试，然后执行test4
     * 修改hibernate.cfg.xml中的hibernate.hbm2ddl.auto=update
     */
    @Test
    public void test4() {
        Session session = HibernateUtil.getCurrentSession();
        Transaction tx = session.beginTransaction();

        User user = session.get(User.class, 1L);
        session.delete(user);

        tx.commit();
    }

    /**
     * 使用级联，删除角色，级联删除用户
     * 去掉User.hbm.xml映射文件中的cascade
     * 加上Role.hbm.xml映射文件中的cascade
     * 先运行test1进行建表，方便测试，然后执行test5
     * 修改hibernate.cfg.xml中的hibernate.hbm2ddl.auto=update
     */
    @Test
    public void test5() {
        Session session = HibernateUtil.getCurrentSession();
        Transaction tx = session.beginTransaction();

        Role role = session.get(Role.class, 2L);
        session.delete(role);

        tx.commit();
```

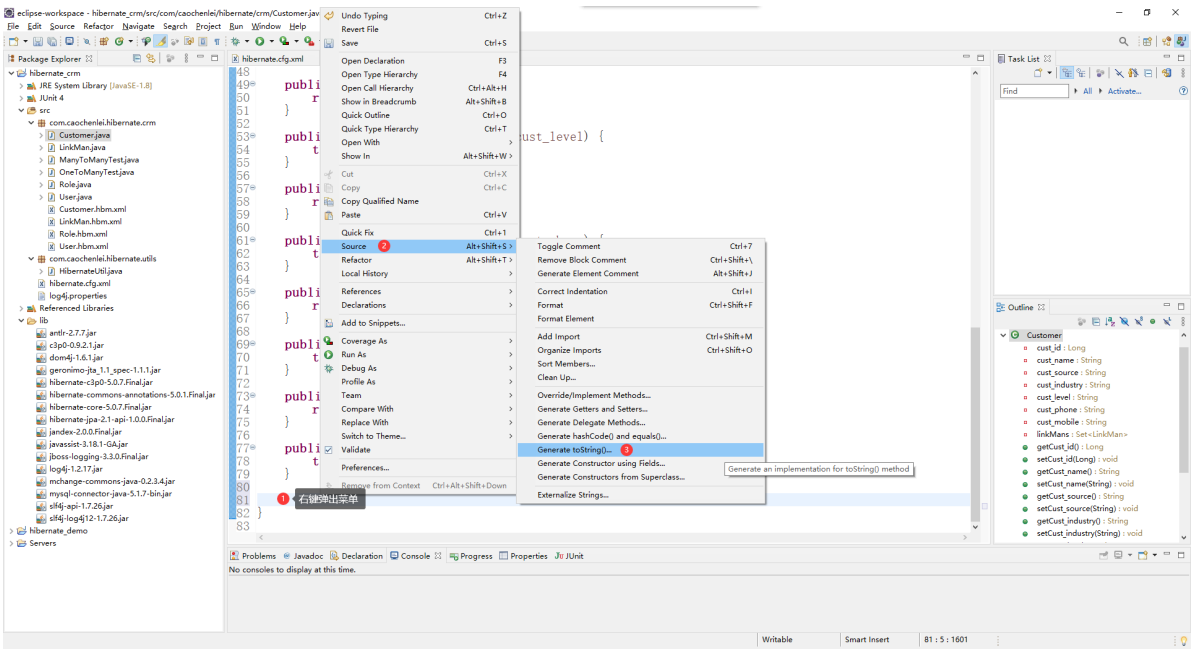
```
    }

    /**
     * 案例模拟演示，将2号用户原有的2号角色改为1号角色
     * 加上User.hbm.xml映射文件中的cascade
     * 加上Role.hbm.xml映射文件中的cascade
     * 先运行test1进行建表，方便测试，然后执行test6
     * 修改hibernate.cfg.xml中的hibernate.hbm2ddl.auto=update
     */
    @Test
    public void test6() {
        Session session = HibernateUtil.getCurrentSession();
        Transaction tx = session.beginTransaction();

        // 查询2号用户
        User user = session.get(User.class, 2L);

        // 查询1号角色
        Role role1 = session.get(Role.class, 1L);

        // 查询2号角色
        Role role2 = session.get(Role.class, 2L);

        // 将2号用户原有的2号角色改为1号角色
        user.getRoles().remove(role2);
        user.getRoles().add(role1);

        session.save(user);

        tx.commit();
    }

}
```

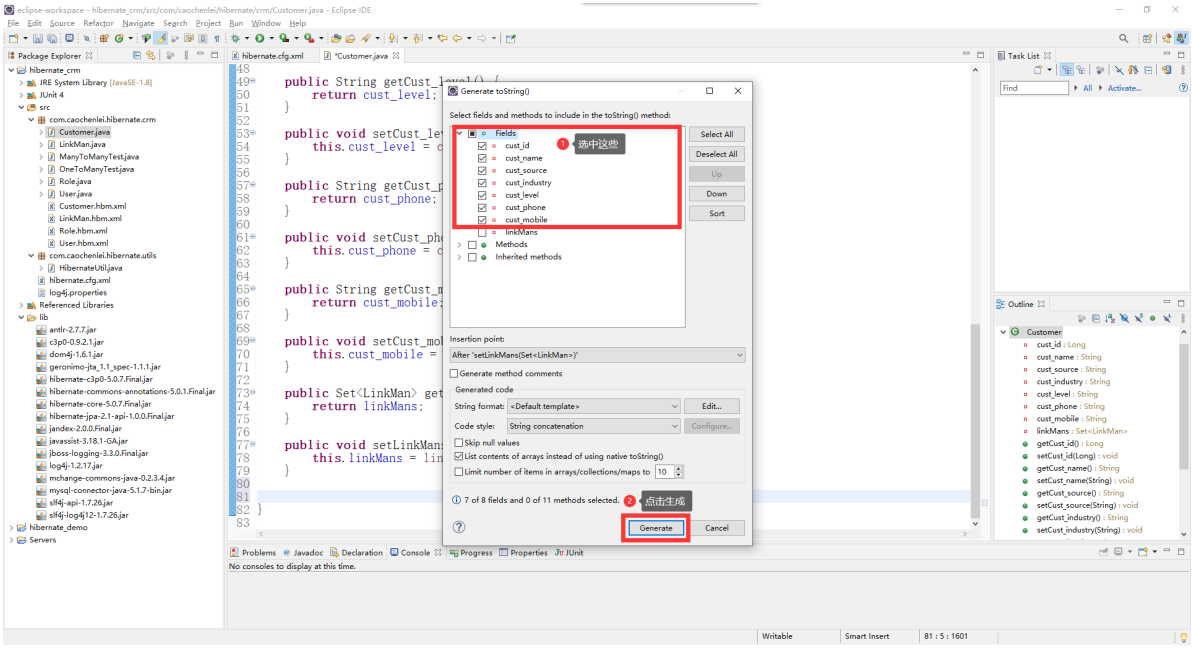
第五章 Hibernate5查询方式

修改文件：hibernate.cfg.xml



修改文件：Customer.java

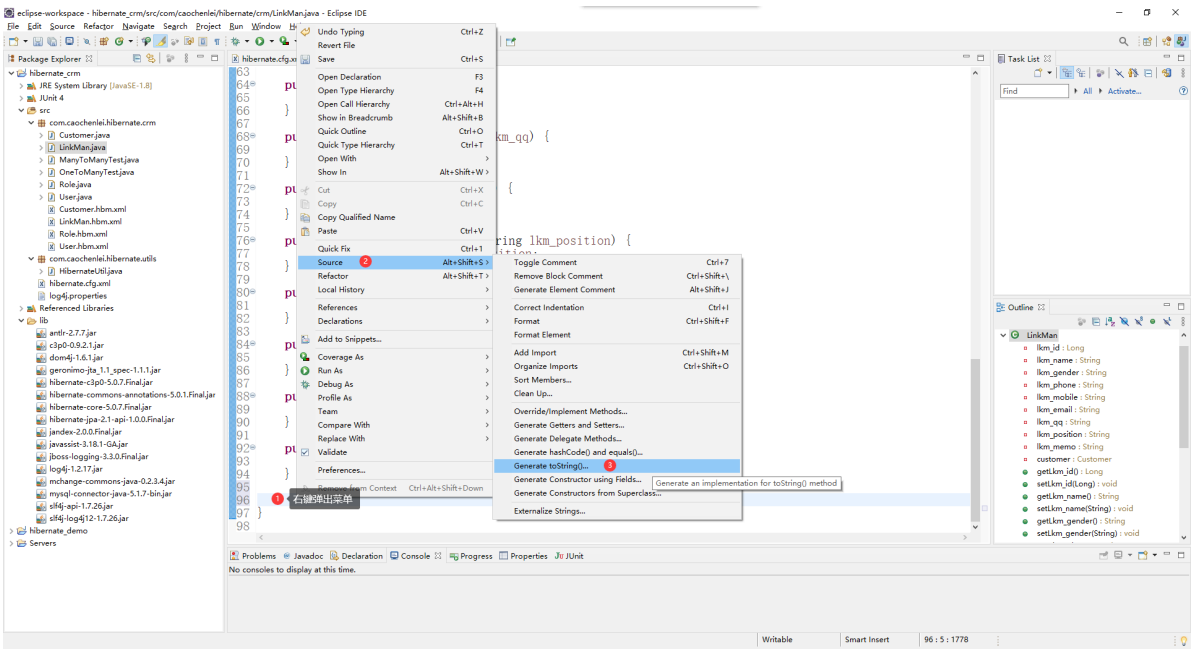


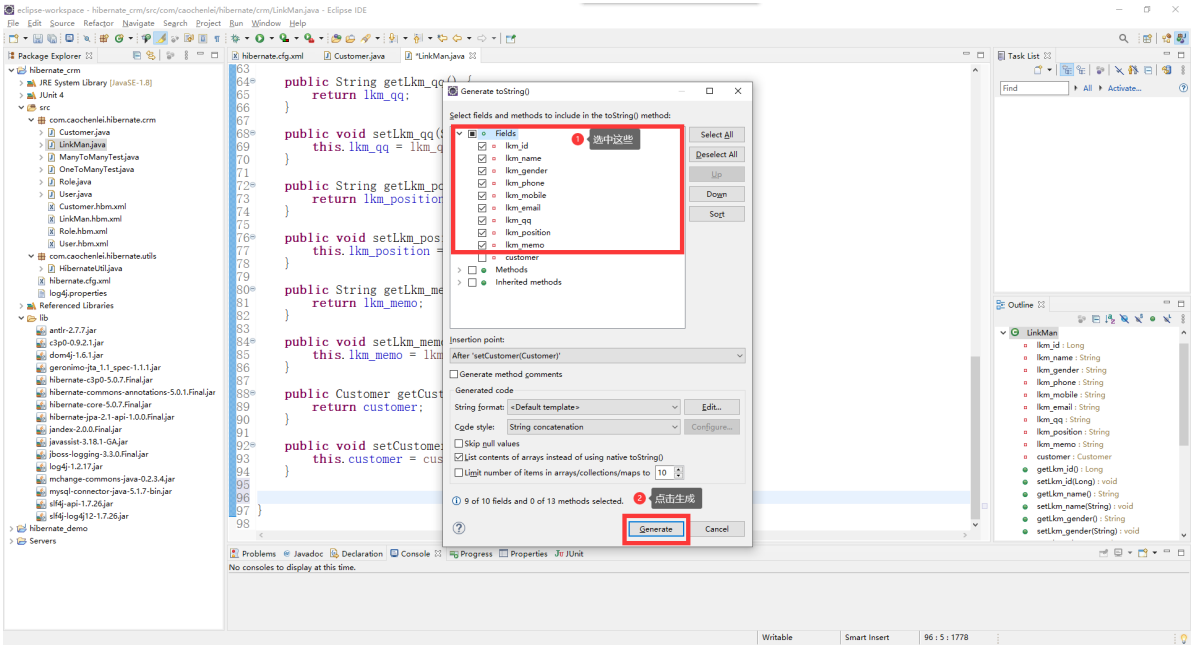


添加构造：Customer.java

```
public Customer() {
    super();
}

public Customer(String cust_name, String cust_source) {
    super();
    this.cust_name = cust_name;
    this.cust_source = cust_source;
}
```

修改文件：LinkMan.java





初始数据：InitQueryData.java（全路径：/hibernate_crm/src/com/caochenlei/hibernate/crm/InitQueryData.java）

```
package com.caochenlei.hibernate.crm;

import org.hibernate.Session;
import org.hibernate.Transaction;
import org.junit.Test;

import com.caochenlei.hibernate.utils.HibernateUtil;

public class InitQueryData {

    /**
     * 执行前：修改hibernate.cfg.xml的hibernate.hbm2ddl.auto=create
     * 执行此方法
     * 执行后：修改hibernate.cfg.xml的hibernate.hbm2ddl.auto=update
     */
    @Test
    public void init() {
```

```
        Session session = HibernateUtil.getCurrentSession();
        Transaction tx = session.beginTransaction();

        // 创建一个客户
        Customer customer1 = new Customer();
        customer1.setCust_name("张三");
        customer1.setCust_source("电视广告");
        // 创建十个联系人
        for (int i = 1; i <= 10; i++) {
            LinkMan linkMan = new LinkMan();
            linkMan.setLkm_name("张三的联系人" + i);
            linkMan.setCustomer(customer1);
            customer1.getLinkMans().add(linkMan);
            session.save(linkMan);
        }
        session.save(customer1);

        // 创建一个客户
        Customer customer2 = new Customer();
        customer2.setCust_name("李四");
        customer2.setCust_source("网络论坛");
        // 创建十个联系人
        for (int i = 1; i <= 10; i++) {
            LinkMan linkMan = new LinkMan();
            linkMan.setLkm_name("李四的联系人" + i);
            linkMan.setCustomer(customer2);
            customer2.getLinkMans().add(linkMan);
            session.save(linkMan);
        }
        session.save(customer2);

        // 创建一个客户
        Customer customer3 = new Customer();
        customer3.setCust_name("王五");
        customer3.setCust_source("街边告示");
        // 创建十个联系人
        for (int i = 1; i <= 10; i++) {
            LinkMan linkMan = new LinkMan();
            linkMan.setLkm_name("王五的联系人" + i);
            linkMan.setCustomer(customer3);
            customer3.getLinkMans().add(linkMan);
            session.save(linkMan);
        }
        session.save(customer3);

        // 创建一个客户
        Customer customer4 = new Customer();
        customer4.setCust_name("王五");
        customer4.setCust_source("电视广告");
        session.save(customer4);

        tx.commit();
    }

}
```

5.1、OID查询

概述：Hibernate根据对象的OID（主键）进行检索

5.1.1、使用get方法

```
Customer customer = session.get(Customer.class,1L);
```

5.1.2、使用load方法

```
Customer customer = session.load(Customer.class,1L);
```

5.2、对象导航查询

概述：Hibernate根据一个已经查询到的对象，获得其关联的对象的一种查询方式

5.2.1、查询客户关联查询联系人

```
Customer customer = session.get(Customer.class,2L);
Set<LinkMan> linkMans = customer.getLinkMans();
```

5.2.2、查询联系人关联查询客户

```
LinkMan linkMan = session.get(LinkMan.class,1L);
Customer customer  = linkMan.getCustomer();
```

5.3、HQL查询

概述：Hibernate Query Language，Hibernate的查询语言，是一种面向对象的方式的查询语言，语法类似SQL。通过session.createQuery()，用于接收一个HQL进行查询方式。一般来说，它查询的是对象而不是表，查询的是对象属性而不是表字段

5.3.1、简单查询

```
@Test
public void test1() {
    Session session = HibernateUtil.getCurrentSession();
    Transaction tx = session.beginTransaction();

    // 查询所有客户
    Query query = session.createQuery("from Customer");
    List<Customer> list = query.list();
    for (Customer customer : list) {
        System.out.println(customer);
    }
    System.err.println("----------------------------------------------------");

    tx.commit();
}
```

5.3.2、别名查询

```
@Test
public void test2() {
    Session session = HibernateUtil.getCurrentSession();
    Transaction tx = session.beginTransaction();

    // 查询所有客户
    Query query1 = session.createQuery("from Customer c");
    List<Customer> list1 = query1.list();
    for (Customer customer : list1) {
        System.out.println(customer);
    }
    System.err.println("----------------------------------------------------");

    // 查询所有客户
    Query query2 = session.createQuery("select c from Customer c");
    List<Customer> list2 = query2.list();
    for (Customer customer : list2) {
        System.out.println(customer);
    }
    System.err.println("----------------------------------------------------");

    tx.commit();
}
```

5.3.3、排序查询

```
@Test
public void test3() {
    Session session = HibernateUtil.getCurrentSession();
    Transaction tx = session.beginTransaction();

    // 采用链式调用，默认情况（升序）
    List<Customer> list1 = session.createQuery("from Customer order by
cust_id").list();
    for (Customer customer : list1) {
        System.out.println(customer);
    }
    System.err.println("----------------------------------------------------");

    // 采用链式调用，升序情况
    List<Customer> list2 = session.createQuery("from Customer order by cust_id
asc").list();
    for (Customer customer : list2) {
        System.out.println(customer);
    }
    System.err.println("----------------------------------------------------");

    // 采用链式调用，降序情况
    List<Customer> list3 = session.createQuery("from Customer order by cust_id
desc").list();
    for (Customer customer : list3) {
        System.out.println(customer);
    }
    System.err.println("----------------------------------------------------");
```

```
    tx.commit();
}
```

5.3.4、条件查询

```
@Test
public void test4() {
    Session session = HibernateUtil.getCurrentSession();
    Transaction tx = session.beginTransaction();

    // 条件查询：按参数位置绑定
    Query query1 = session.createQuery("from Customer where cust_source = ? and
cust_name like ?");
    query1.setParameter(0, "电视广告");
    query1.setParameter(1, "王%");
    List<Customer> list1 = query1.list();
    for (Customer customer : list1) {
        System.out.println(customer);
    }
    System.err.println("----------------------------------------------------");

    // 条件查询：按参数名称绑定
    Query query2 = session.createQuery("from Customer where cust_source = :aaa
and cust_name like :bbb");
    query2.setParameter("aaa", "电视广告");
    query2.setParameter("bbb", "王%");
    List<Customer> list2 = query2.list();
    for (Customer customer : list2) {
        System.out.println(customer);
    }
    System.err.println("----------------------------------------------------");

    tx.commit();
}
```

5.3.5、投影查询

```
@Test
public void test5() {
    Session session = HibernateUtil.getCurrentSession();
    Transaction tx = session.beginTransaction();

    // 查询所有客户名称：单个字段查询
    Query query1 = session.createQuery("select c.cust_name from Customer c");
    List<Object> list1 = query1.list();
    for (Object cust_name : list1) {
        System.out.println(cust_name);
    }
    System.err.println("----------------------------------------------------");

    // 查询所有客户名称、客户来源：多个字段查询，封装到数组中
    Query query2 = session.createQuery("select c.cust_name,c.cust_source from
Customer c");
    List<Object[]> list2 = query2.list();
    for (Object[] objects : list2) {
        System.out.println(Arrays.toString(objects));
    }
```

```
        System.err.println("----------------------------------------------------");

        // 查询所有客户名称、客户来源：多个字段查询，封装到对象中
        Query query3 = session.createQuery("select new
Customer(c.cust_name,c.cust_source) from Customer c");
        List<Customer> list3 = query3.list();
        for (Customer customer : list3) {
            System.out.println(customer);
        }
        System.err.println("----------------------------------------------------");

        tx.commit();
    }
```

5.3.6、分页查询

```
    @Test
    public void test6() {
        Session session = HibernateUtil.getCurrentSession();
        Transaction tx = session.beginTransaction();

        // 分页查询
        Query query = session.createQuery("from LinkMan");
        query.setFirstResult(0);
        query.setMaxResults(10);
        List<LinkMan> list = query.list();
        for (LinkMan linkMan : list) {
            System.out.println(linkMan);
        }
        System.err.println("----------------------------------------------------");

        tx.commit();
    }
```

5.3.7、分组查询

```
    @Test
    public void test7() {
        Session session = HibernateUtil.getCurrentSession();
        Transaction tx = session.beginTransaction();

        // 聚合函数：count(),max(),min(),avg(),sum()
        Object object = session.createQuery("select count(*) from
Customer").uniqueResult();
        System.out.println(object);
        System.err.println("----------------------------------------------------");

        // 分组统计：
        List<Object[]> list = session.createQuery("select cust_source,count(*) from
Customer group by cust_source").list();
        for (Object[] objects : list) {
            System.out.println(Arrays.toString(objects));
        }
        System.err.println("----------------------------------------------------");

        tx.commit();
    }
```

5.3.8、多表查询

```
@Test
public void test8() {
    Session session = HibernateUtil.getCurrentSession();
    Transaction tx = session.beginTransaction();

    // 内连接
    List<Object[]> list1 = session.createQuery("from Customer c inner join
c.linkMans").list();
    for (Object[] objects : list1) {
        System.out.println(Arrays.toString(objects));
    }
    System.err.println("----------------------------------------------------");

    // 迫切内连接（hibernate独有，将另一个对象的数据封装到该对象中）
    List<Customer> list2 = session.createQuery("select distinct c from Customer
c inner join fetch c.linkMans").list();
    for (Customer customer : list2) {
        System.out.println(customer);
    }
    System.err.println("----------------------------------------------------");

    // 左外连接
    List<Object[]> list3 = session.createQuery("from Customer c left outer join
c.linkMans").list();
    for (Object[] objects : list3) {
        System.out.println(Arrays.toString(objects));
    }
    System.err.println("----------------------------------------------------");

    // 迫切左外连接（hibernate独有，将另一个对象的数据封装到该对象中）
    List<Customer> list4 = session.createQuery("select distinct c from Customer
c left outer join fetch c.linkMans").list();
    for (Customer customer : list4) {
        System.out.println(customer);
    }
    System.err.println("----------------------------------------------------");

    // 右外连接
    List<Object[]> list5 = session.createQuery("from Customer c right outer join
c.linkMans").list();
    for (Object[] objects : list5) {
        System.out.println(Arrays.toString(objects));
    }
    System.err.println("----------------------------------------------------");

    tx.commit();
}
```

5.4、QBC查询

概述：Query By Criteria，条件查询。是一种更加面向对象化的查询的方式

5.4.1、简单查询

```
@Test
public void test1() {
    Session session = HibernateUtil.getCurrentSession();
    Transaction tx = session.beginTransaction();

    // 查询所有客户
    Criteria criteria = session.createCriteria(Customer.class);
    List<Customer> list = criteria.list();
    for (Customer customer : list) {
        System.out.println(customer);
    }
    System.err.println("------------------------------------------------------");

    tx.commit();
}
```

5.4.2、排序查询

```
@Test
public void test2() {
    Session session = HibernateUtil.getCurrentSession();
    Transaction tx = session.beginTransaction();

    // 升序查询
    Criteria criteria1 = session.createCriteria(Customer.class);
    criteria1.addOrder(Order.asc("cust_id"));
    List<Customer> list1 = criteria1.list();
    for (Customer customer : list1) {
        System.out.println(customer);
    }
    System.err.println("------------------------------------------------------");

    // 降序查询
    Criteria criteria2 = session.createCriteria(Customer.class);
    criteria2.addOrder(Order.desc("cust_id"));
    List<Customer> list2 = criteria2.list();
    for (Customer customer : list2) {
        System.out.println(customer);
    }
    System.err.println("------------------------------------------------------");

    tx.commit();
}
```

5.4.3、条件查询

```
@Test
public void test3() {
    Session session = HibernateUtil.getCurrentSession();
    Transaction tx = session.beginTransaction();
```

```
    // 条件查询
    Criteria criteria = session.createCriteria(Customer.class);
    /**
     * 设置条件:
     * =   eq
     * >   gt
     * >=  ge
     * <   lt
     * <=  le
     * <>  ne
     *     like
     *     in
     *     and
     *     or
     */
    criteria.add(Restrictions.eq("cust_source", "电视广告"));
    criteria.add(Restrictions.like("cust_name", "王%"));
    List<Customer> list = criteria.list();
    for (Customer customer : list) {
        System.out.println(customer);
    }
    System.err.println("-----------------------------------------------------");

    tx.commit();
}
```

5.4.4、分页查询

```
@Test
public void test4() {
    Session session = HibernateUtil.getCurrentSession();
    Transaction tx = session.beginTransaction();

    // 分页查询
    Criteria criteria = session.createCriteria(LinkMan.class);
    criteria.setFirstResult(0);
    criteria.setMaxResults(10);
    List<LinkMan> list = criteria.list();
    for (LinkMan linkMan : list) {
        System.out.println(linkMan);
    }
    System.err.println("-----------------------------------------------------");

    tx.commit();
}
```

5.4.5、分组查询

```
@Test
public void test5() {
    Session session = HibernateUtil.getCurrentSession();
    Transaction tx = session.beginTransaction();

    Criteria criteria = session.createCriteria(Customer.class);
    /**
     * add              :分组前的条件
```

```
     * addOrder          :排序
     * setProjection     :分组后的条件
     */
    criteria.setProjection(Projections.rowCount());
    Long num = (Long) criteria.uniqueResult();
    System.out.println(num);
    System.err.println("-----------------------------------------------------");

    tx.commit();
}
```

5.4.6、离线查询

```
@Test
public void test6() {
    // 相当控制层
    DetachedCriteria detachedCriteria =
DetachedCriteria.forClass(Customer.class);
    detachedCriteria.add(Restrictions.like("cust_name", "王%"));

    // 相当业务层
    Session session = HibernateUtil.getCurrentSession();
    Transaction transaction = session.beginTransaction();

    Criteria criteria = detachedCriteria.getExecutableCriteria(session);
    List<Customer> list = criteria.list();
    for (Customer customer : list) {
        System.out.println(customer);
    }
    System.err.println("-----------------------------------------------------");

    transaction.commit();
}
```

5.5、SQL查询

概述：通过使用sql语句进行查询

```
@Test
public void test1() {
    Session session = HibernateUtil.getCurrentSession();
    Transaction tx = session.beginTransaction();

    // 直接查询
    SQLQuery sqlQuery1 = session.createSQLQuery("select * from cst_customer");
    List<Object[]> list1 = sqlQuery1.list();
    for (Object[] objects : list1) {
        System.out.println(Arrays.toString(objects));
    }
    System.err.println("-----------------------------------------------------");

    // 对象查询
    SQLQuery sqlQuery2 = session.createSQLQuery("select * from cst_customer");
    sqlQuery2.addEntity(Customer.class);
    List<Customer> list2 = sqlQuery2.list();
    for (Customer customer : list2) {
```

```
            System.out.println(customer);
        }
        System.err.println("-----------------------------------------------------");

        tx.commit();
    }
```

第六章 Hibernate5优化机制

6.1、延迟加载

延迟加载：lazy（又称为懒加载），执行到该行代码的时候，不会发送语句去进行查询，在真正使用这个对象的属性的时候才会发送SQL语句进行查询

- 类级别的延迟加载
  - 指的是通过load方法查询某个对象的时候，是否采用延迟加载
  - 类级别延迟加载通过上的lazy进行配置，如果让lazy失效
    - 将lazy设置为false
    - 将持久化类使用final修饰
    - Hibernate. Initialize(对象)
- 关联级别的延迟加载
  - 指的是在查询到某个对象的时候，查询其关联的对象的时候，是否采用延迟加载
  - 通过客户获得联系人的时候，联系人对象是否采用了延迟加载，称为是关联级别的延迟
  - 抓取策略往往会和关联级别的延迟加载一起使用，优化语句

6.2、抓取策略

抓取策略：通过一个对象抓取到关联对象需要发送SQL语句，SQL语句如何发送，发送成什么样格式通过策略进行配置

6.2.1、set上的fetch和lazy

- fetch：抓取策略，控制SQL语句格式
  - select： 默认值，发送普通的select语句，查询关联对象
  - join： 发送一条迫切左外连接查询关联对象
  - subselect： 发送一条子查询查询其关联对象
- lazy：延迟加载，控制查询关联对象的时候是否采用延迟
  - true： 默认值，查询关联对象的时候，采用延迟加载
  - false： 查询关联对象的时候，不采用延迟加载
  - extra： 极其懒惰

在实际开发中，一般都采用默认值，如果有特殊的需求，可能需要配置join

6.2.2、many-to-one上的fetch和lazy

- fetch： 抓取策略，控制SQL语句格式
  - select： 默认值，发送普通的select语句，查询关联对象
  - join： 发送一条迫切左外连接查询关联对象
- lazy： 延迟加载，控制查询关联对象的时候是否采用延迟
  - proxy： 默认值，proxy具体的取值，取决于另一端的上的lazy的值
  - false： 查询关联对象，不采用延迟
  - no-proxy： （不会使用）

在实际开发中，一般都采用默认值，如果有特殊的需求，可能需要配置join

6.3、批量抓取

批量抓取：关联对象一起抓取，默认抓取一条，可以通过设置batch-size属性进行调整

- 一的一方的：batch-size="5"（例如：获取联系人的时候，批量去抓取客户）
- 一的一方的：batch-size="5"（例如：获取客户的时候 ，批量去抓取联系人 ）