**Hibernate5 第一天**

**Hibernate框架的学习路线**

1. 注意：Hibernate框架知识点非常多，比较杂乱，大家要做好笔记记录的工作

2. 学习的路线  
  **\* 第一天：主要是学习框架的入门，自己搭建框架，完成增删改查的操作  
  \* 第二天：主要学习一级缓存、事务管理和简单的查询  
  \* 第三天：主要学习一对多和多对多的操作等  
  \* 第四天：主要学习基本查询和查询优化**

**【学习目标】**

通过本内容的学习，学生：

## 1、能够描述CRM系统的需求

1. 说出CRM系统的概述。
2. 说出CRM系统的功能模块。

## 2、能够描述JavaEE的三层架构

1. 说出三层架构的介绍。
2. 说出Hibernate框架是属于EE开发中哪一层。
3. 说出持久层是作用。

## 3、能够描述Hibernate框架

1. 说出框架的介绍。
2. 说出Hibernate的介绍。
3. 说出Hibernate的优缺点。
4. 说出Hibernate和原始JDBC的优劣

## 4、能够独立搭建Hibernate的开发环境

1. 说出Hibernate运行必需的jar包从哪里找
2. 独立创建所需要的数据库。

## 5、能够独立编写客户的实体类对象

1. 说出编写实体类的基本原则。
2. 参照crm\_cst\_customer表创建客户实体类。

## 6、能够独立编写客户信息表的映射配置文件

1. 说出hbm映射文件的作用。
2. 独立配置hbm映射文件的dtd约束。
3. 解释hbm映射文件中类与表的映射方法。
4. 解释hbm映射文件中主键的映射方法。
5. 解释hbm映射文件中普通字段的映射方法。
6. 独立编写客户信息表映射文件的内容。

## 7、能够独立编写hibernate.cfg.xml配置文件

1. 说出hibernate.cfg.xml的文件名称及位置要求。
2. 独立创建hibernate.cfg.xml文件。
3. 独立配置hibernate.cfg.xml的dtd约束。
4. 说出hibernate.cfg.xml的主要配置属性。
5. 独立配置hibernate.cfg.xml的文件内容。
6. 独立完成使用mapping标签和resource属性完成映射文件的引入。

## 8、能够独立编写测试类完成保存客户操作

1. 独立完成保存客户的操作

## 9、能够说出hibernate映射文件中的常用标签和属性的配置

1. 说出class标签的作用
2. 说出class标签中name和table属性的作用
3. 说出id标签的作用
4. 说出id标签中name和column属性的作用
5. 说出property标签的作用
6. 说出property标签中name和column属性的作用

## 10、能够描述Configuration对象的作用

1. 说出Configuration对象的作用
2. 说出Configuration是如何加载hibernate的配置文件的

## 11、能够描述SessionFactory的作用及编写工具类

1. 说出SessionFactory对象的作用
2. 独立抽取Hibernate的工具类

## 12、能够描述Session对象的作用及常用方法

1. 说出session对象的作用
2. 说出session对象的常用方法

## 13、能够使用工具类完成增删该查的操作

1. 独立使用抽取的工具类完成添加客户操作
2. 独立使用抽取的工具类完成查询客户操作
3. 独立使用抽取的工具类完成修改客户操作
4. 独立使用抽取的工具类完成删除客户操作

## 14、能够描述Transaction的作用及常用方法

1. 说出Transaction对象的作用
2. 说出Transaction对象的常用方法

# 框架和CRM项目的整体介绍

      
    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  
      
    2. CRM的模块  
        \* 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  
      
    3. 模块的具体功能  
        \* 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

### 案例一：完成客户的添加的操作 ###  
      
  
      
**\*\*需求分析\*\***  
      
    1. CRM系统中客户信息管理模块功能包括  
        \* 新增客户信息  
        \* 客户列表查询  
      
    2. 要实现客户的新增功能

# Hibernate概述（了解）

## Hibernate框架的概述

  \* Hibernate是一个开放源代码的对象关系映射（ORM）框架，它对JDBC进行了非常轻量级的对象封装，使得Java程序员可以随心所欲的使用对象编程思维来操纵数据库。   
  \* Hibernate可以应用在任何使用JDBC的场合，既可以在Java的客户端程序使用，也可以在Servlet/JSP的Web应用中使用。  
  \* Hibernate是轻量级JavaEE应用的持久层解决方案，是一个关系数据库ORM框架  
 记住：Hibernate是一个持久层的ORM框架！！！

## 什么是ORM（对象关系映射）

    1. ORM映射：Object Relational Mapping  
        \* O：面向对象领域的Object（JavaBean对象）  
        \* R：关系数据库领域的Relational（表的结构）  
        \* M：映射Mapping（XML的配置文件）

将数据表的关系，映射为类和对象之间关系:

|  |  |
| --- | --- |
| 数据库（关系型） | Java（面向对象的编程语言） |
| 表 | 类 |
| 表的字段 | 类的属性 |
| 表的一行数据 | 类的一个实例对象 |

 2. 简单一句话：Hibernate使程序员通过操作对象的方式来操作数据库表记录

## Hibernate优点

## \* Hibernate对JDBC访问数据库的代码做了封装，大大简化了数据访问层繁琐的重复性代码      \* 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层编码工作      \* Hibernate的性能非常好，因为它是一个轻量级框架。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

# hibernate的快速入门

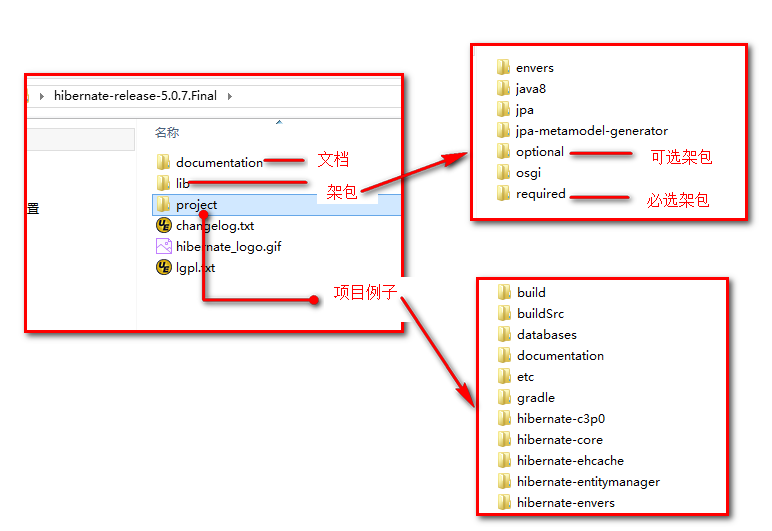
开发工具：eclipse mars2

数据库：mysql5

## hibernate的jar包下载和导入

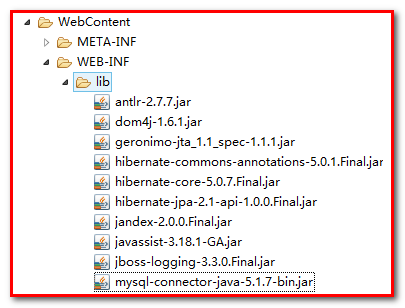
1. 下载相应的jar包等  
  \* http://sourceforge.net/projects/hibernate/files/hibernate-orm/5.0.7.Final/hibernate-release-5.0.7.Final.zip/download      
      
2. 解压后对目录结构有一定的了解

Hibernate开发包内容：



### Jar包的导入（开发环境的搭建）

新建web工程Hibernate5\_d01\_c03

* 引入Hibernate开发所需要的jar包  
    \* MySQL的驱动jar包  
    \* Hibernate开发需要的jar包（资料/hibernate-release-5.0.7.Final/lib/required/所有jar包）  
    

## 基于hibernate实现数据库表CRUD的操作（重点）

### 开发准备和配置（三个准备，7个步骤）

【第一个准备】 :创建表结构:

1. 建表语句如下

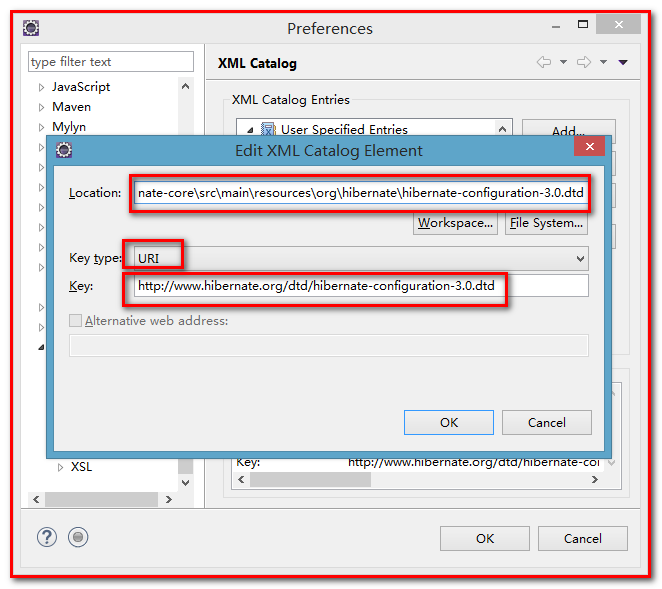
|  |
| --- |
| create database hibernate\_day01;  use hibernate\_day01;  CREATE TABLE `cst\_customer` (  `cust\_id` bigint(32) NOT NULL AUTO\_INCREMENT COMMENT '客户编号(主键)',  `cust\_name` varchar(32) NOT NULL COMMENT '客户名称(公司名称)',  `cust\_user\_id` bigint(32) DEFAULT NULL COMMENT '负责人id',  `cust\_create\_id` bigint(32) DEFAULT NULL COMMENT '创建人id',  `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\_linkman` varchar(64) DEFAULT NULL COMMENT '联系人',  `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; |

【第二个准备】: 编写Hibernate核心的配置文件

1. 在src目录下，创建名称为hibernate.cfg.xml的配置文件  
   2. 在XML中引入DTD约束

|  |
| --- |
| <!DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd"> |

没有联网的情况下需要做如下配置才有提示信息



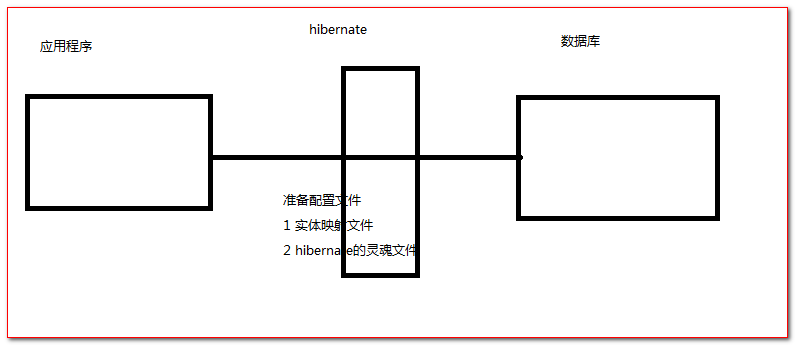
1. 打开：资料/hibernate-release-5.0.7.Final/project/etc/hibernate.properties，可以查看具体的配置信息  
           \* 必须配置的4大参数  
               #hibernate.connection.driver\_class com.mysql.jdbc.Driver  
               #hibernate.connection.url jdbc:mysql:///test  
               #hibernate.connection.username gavin  
               #hibernate.connection.password  
             
           \* 数据库的方言（必须配置的）  
               #hibernate.dialect org.hibernate.dialect.MySQLDialect  
             
           \* 可选的配置  
               #hibernate.show\_sql true  
               #hibernate.format\_sql true  
               #hibernate.hbm2ddl.auto update  
             
           \* 引入映射配置文件（一定要注意，要引入映射文件，框架需要加载映射文件）  
               \* <mapping resource="com/igeek/demo1/Customer.hbm.xml"/>  
         
   4. 具体的配置如下

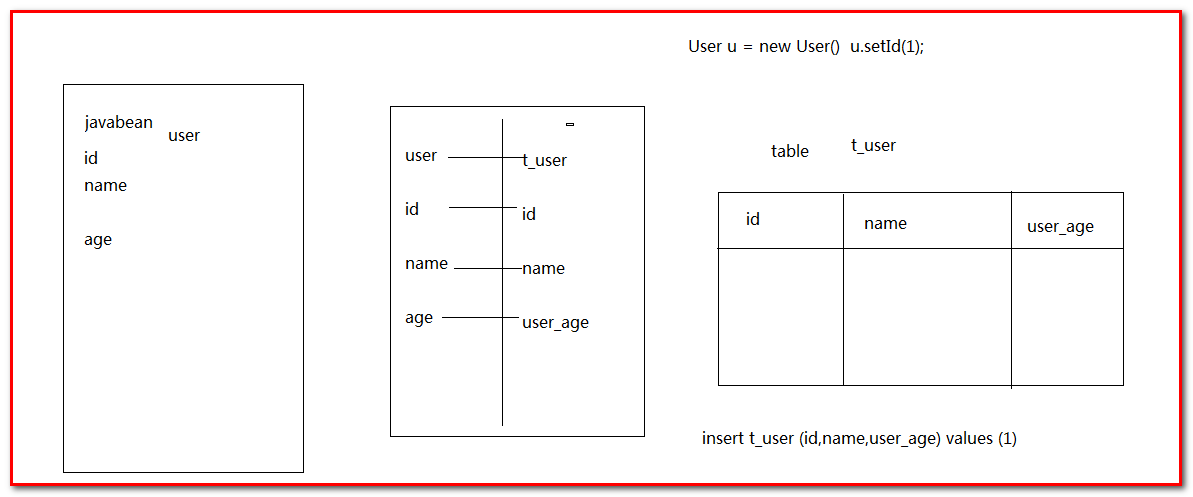
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">  <hibernate-configuration>  <session-factory>  <property name=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>  <property name=*"hibernate.connection.url"*>jdbc:mysql:///hibernate\_day01</property>  <property name=*"hibernate.connection.username"*>root</property>  <property name=*"hibernate.connection.password"*>123456</property>  <property name=*"hibernate.dialect org.hibernate.dialect.MySQLDialect"*></property>  </session-factory>  </hibernate-configuration> |

1. 在com.igeek.demo1中创建TestCustomer类，测试是否连接上数据库,代码如下：

|  |
| --- |
| **package** com.igeek.demo1;  **import** org.hibernate.cfg.Configuration;  **import** org.junit.Test;  **public** **class** TestCustomer {  @Test  **public** **void** run1(){  //hibernate读取连接数据的配置文件hibernate.cfg.xml  Configuration config = **new** Configuration();  config.configure();  //连接数据库  SessionFactory factory = config.buildSessionFactory();  }  } |

【第三个准备】:编写实体类和ORM映射文件





【编写Customer实体类】：

在com.igeekdemo1包中创建Customer.java类，类中代码如下：

1. Customer类的代码如下：

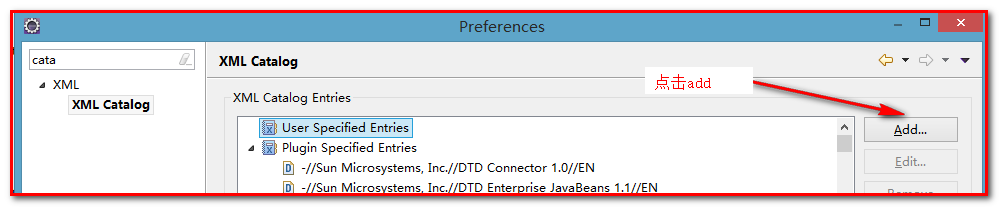
|  |
| --- |
| **public** **class** Customer {  **private** Long cust\_id;  **private** String cust\_name;  **private** Long cust\_user\_id;  **private** Long cust\_create\_id;  **private** String cust\_source;  **private** String cust\_industry;  **private** String cust\_level;  **private** String cust\_linkman;  **private** String cust\_phone;  **private** String cust\_mobile;  // 省略get和set方法  } |

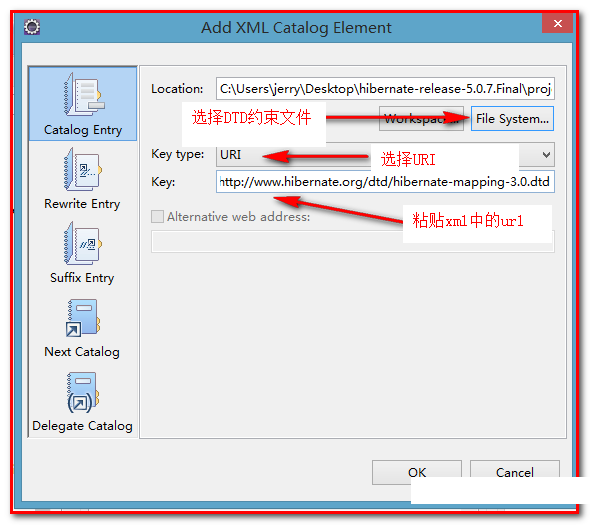
【创建类与表结构的映射】:

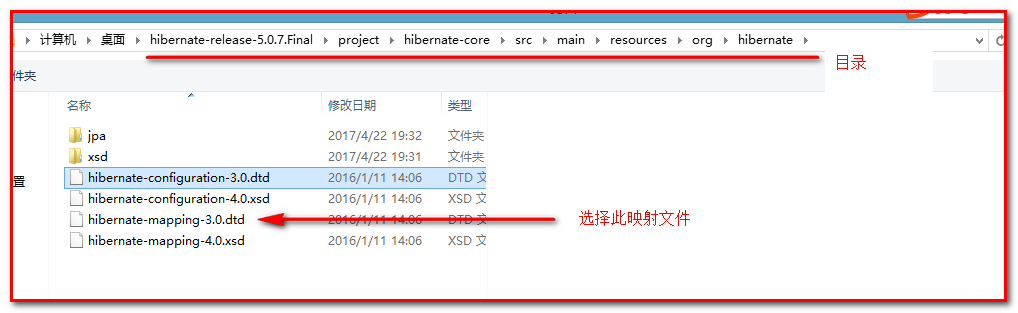
1. 在JavaBean所在的包下创建映射的配置文件  
    \* 默认的命名规则为：实体类名.hbm.xml  
    \* 在xml配置文件中引入约束（引入的是hibernate3.0的dtd约束，不要引入4的约束）

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

1. 如果不能上网，编写配置文件是没有提示的，需要自己来配置  
     \* 先复制http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd --> window --> preferences --> 搜索xml --> 选择xml catalog --> 点击add --> 选择URI --> 粘贴复制的地址 --> 选择location，选择本地的DTD的路径

【请注意】在配置完catalog之后，需要重新打开Customer.hbm.xml文件  




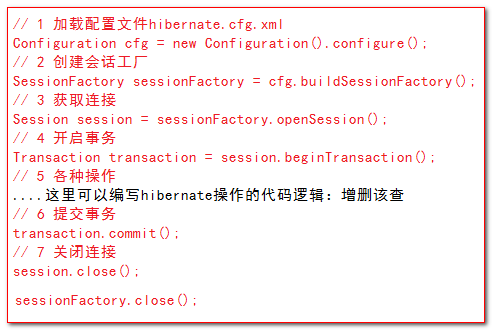
  
3. 编写映射的配置文件

|  |
| --- |
| <?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.igeek.demo1.Customer"* table=*"cst\_customer">*  <id name=*"cust\_id"* column=*"cust\_id"*>  <generator class=*"native"*></generator>  </id>  <property name=*"cust\_name"* column=*"cust\_name"*/>  <property name=*"cust\_user\_id"* column=*"cust\_user\_id"*/>  <property name=*"cust\_create\_id"* column=*"cust\_create\_id"*/>  <property name=*"cust\_source"* column=*"cust\_source"*/>  <property name=*"cust\_industry"* column=*"cust\_industry"*/>  <property name=*"cust\_level"* column=*"cust\_level"*/>  <property name=*"cust\_linkman"* column=*"cust\_linkman"*/>  <property name=*"cust\_phone"* column=*"cust\_phone"*/>  <property name=*"cust\_mobile"* column=*"cust\_mobile"*/>  </class>  </hibernate-mapping> |

4.在hibernate.cfg.xml中引入上述配置文件

|  |
| --- |
| <mapping resource=*"com/igeek/demo1/Customer.hbm.xml"*/> |

【7个步骤】 : java编程实现CRUD，基本示例如下：

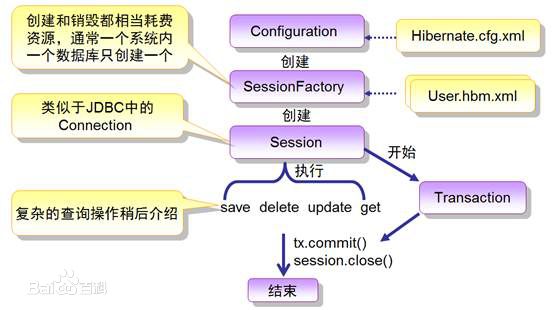


### 保存（插入）数据

在TestCustomer类中编写testSave方法用来保存数据,代码如下：

|  |
| --- |
| **public** **class** TestCustomer {  @Test  **public** **void** testSave(){  //先加载配置文件,默认是src目录下的hibernate.cfg.xml  Configuration config = **new** Configuration().configure();  //创建sessionFactory对象  SessionFactory factory = config.buildSessionFactory();  //创建session对象  Session session = factory.openSession();  //开启事务  Transaction tr = session.beginTransaction();  //编写保存代码  Customer c = **new** Customer();  c.setCust\_name("刘亦菲");  c.setCust\_mobile("1381383838888");  //保存客户  session.save(c);  //提交事务  tr.commit();  //释放资源  session.close();  factory.close();    }  } |

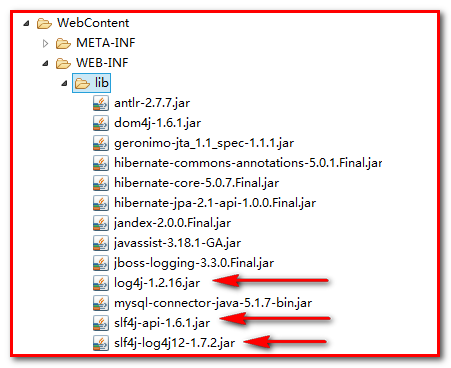
总结：



# 技术分析之：Hibernate常用的配置文件

## log4j的简单的使用  1. 做一个事情

\* 导入日志jar包（资料/jar包/log4j/所有jar包）  
  \* 把log4j.properties文件拷贝到src目录下就OK。

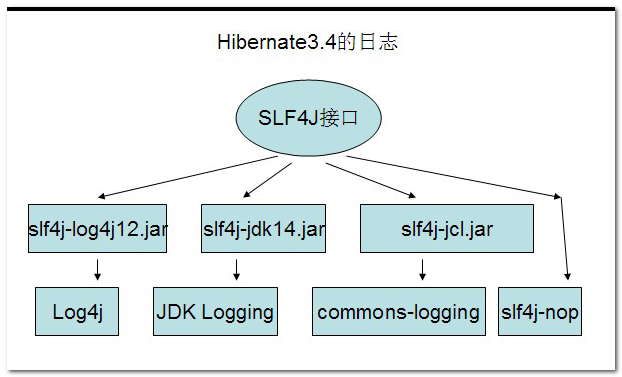


注意：Hibernate开发包中只是提供了一个日志接口的包slf4j，并没有提供日志的具体实现，因此，需要额外下载slf4j和log4j的开发包。



【什么是slf4j】





slf4j-api只是接口包（解耦合的思想，静态绑定思想），需要日志系统的实现log4j

* slf4j整合log4j的jar：slf4j-log4j12-1.7.2.jar slf4j-api-1.6.1.jar

log4j的开发jar：log4j-1.2.16.jar

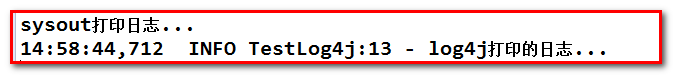
|  |
| --- |
| ### direct log messages to stdout ###  log4j.appender.stdout=org.apache.log4j.ConsoleAppender  log4j.appender.stdout.Target=System.err  log4j.appender.stdout.layout=org.apache.log4j.PatternLayout  log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c**{1}**:%L - %m%n  ### direct messages to file mylog.log ###  log4j.appender.file=org.apache.log4j.FileAppender  log4j.appender.file.File=d:\\mylog.log  log4j.appender.file.layout=org.apache.log4j.PatternLayout  log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c**{1}**:%L - %m%n  ### set log levels - for more verbose logging change 'info' to 'debug' ###  log4j.rootLogger=info, stdout |

代码演示：

在com.igeek.demo2中创建TestLog4j类

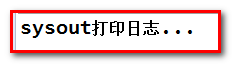
|  |
| --- |
| **package** com.igeek.demo2;  **import** org.apache.log4j.Logger;  **import** org.junit.Test;  **public** **class** TestLog4j {  //创建日志记录对象  **private** Logger logger = Logger.*getLogger*(TestLog4j.**class**);    @Test  **public** **void** run1(){  System.***out***.println("sysout打印日志...");  logger.info("log4j打印的日志...");  }  } |

运行单元测试，两种方式都可以输出日志：



修改log4j.properties，把log4j.rootLogger=info, stdout，改成log4j.rootLogger=off, stdout，关闭日志输出

运行单元测试，使用log4j关闭了日志输出：



修改log4j.properties，把log4j.rootLogger=info, stdout，改成log4j.rootLogger=info,file，

把日志输出到文件中log4j.appender.file.File=d:\\mylog.log

运行单元测试,在d盘中会自动创建mylog.log文件并保存日志信息(项目上线后使用这种方式)

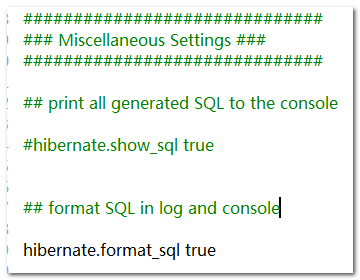
## 配置文件的提示的问题（必须要配置）

 1. 如果能上网，提示是默认就有的。  
 2. 如果不能上网，编写配置文件是没有提示的，需要自己来配置  
    \* 先复制http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd --> window --> preferences --> 搜索xml --> 选择xml catalog --> 点击add --> 选择URI --> 粘贴复制的地址 --> 选择location，选择本地的DTD的路径

## **Hibernate配置文件之映射配置文件**  1. 映射文件，即Customer.hbm.xml的配置文件    \* <class>标签        -- 用来将类与数据库表建立映射关系       \* name         -- 类的全路径       \* table         -- 表名.(类名与表名一致,那么table属性也可以省略)       \* catalog        -- 数据库的名称，基本上都会省略不写             \* <id>标签          -- 用来将类中的属性与表中的主键建立映射，id标签就是用来配置主键的。       \* name         -- 类中属性名       \* column        -- 表中的字段名.(如果类中的属性名与表中的字段名一致,那么column可以省略.)       \* length        -- 字段的程度，如果数据库已经创建好了，那么length可以不写。如果没有创建好，生成表结构时，length最好指定。             \* <property>         -- 用来将类中的普通属性与表中的字段建立映射.       \* name         -- 类中属性名       \* column        -- 表中的字段名.(如果类中的属性名与表中的字段名一致,那么column可以省略.)       \* length        -- 数据长度       \* type         -- 数据类型（一般都不需要编写，如果写需要按着规则来编写）        \* Hibernate的数据类型    type="string"         \* Java的数据类型       type="java.lang.String"         \* 数据库字段的数据类型    <column name="name" sql-type="varchar"/>

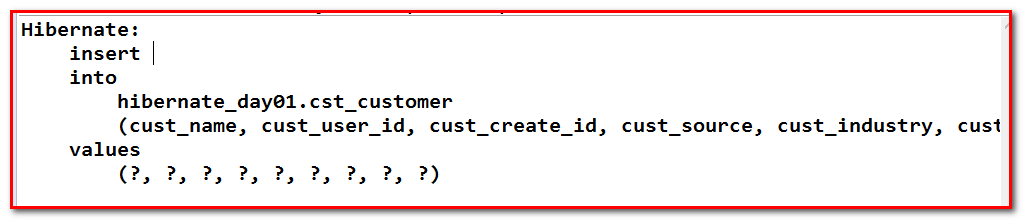
## Hibernate配置文件之核心配置文件 1. 核心配置文件的两种方式（了解）   \* 第一种方式是属性文件的形式，即properties的配置文件   \* hibernate.properties   \* hibernate.connection.driver\_class=com.mysql.jdbc.Driver   \* 缺点     \* 不能加载映射的配置文件，需要手动编写代码去加载            \* 第二种方式是XML文件的形式，开发基本都会选择这种方式     \* hibernate.cfg.xml    \* <property name="hibernate.connection.driver\_class" >com.mysql.jdbc.Driver</property>       \* 优点        \* 格式比较清晰        \* 编写有提示        \* 可以在该配置文件中加载映射的配置文件（最主要的）          2. 关于hibernate.cfg.xml的配置文件方式       \* 必须有的配置        \* 数据库连接信息:        hibernate.connection.driver\_class        -- 连接数据库驱动程序         hibernate.connection.url             -- 连接数据库URL         hibernate.connection.username          -- 数据库用户名         hibernate.connection.password          -- 数据库密码                    \* 方言:        hibernate.dialect                -- 操作数据库方言       \* 可选的配置         \* hibernate.show\_sql                -- 显示SQL         \* hibernate.format\_sql               -- 格式化SQL         \* hibernate.hbm2ddl.auto             -- 通过映射转成DDL语句           \* create      -- 每次都会创建一个新的表.---测试的时候           \* create-drop   -- 每次都会创建一个新的表,当执行结束之后,将创建的这个表删除.---测试的时候           \* update     -- 如果有表,使用原来的表.没有表,创建一个新的表.同时更新表结构.           \* validate   -- 如果有表,使用原来的表.同时校验映射文件与表中字段是否一致如果不一致就会报错.                 \* 加载映射          \* 如果XML方式：<mapping resource="cn/itcast/demo1/Customer.hbm.xml" />

### 控制台打印sql



建议在测试环境下，都将sql打开，便于调试。

|  |
| --- |
| <!-- 打印sql语句的配置 -->  <property name=*"hibernate.show\_sql"*>true</property>  <property name=*"hibernate.format\_sql"*>true</property> |



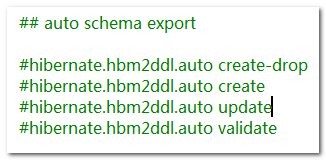
### Hibernate框架整合C3P0连接池

1. 先导入C3P0的jar包  
  \* 先导入C3P0的jar包（3个jar包），在hibernate-release-5.0.7.Final\lib\optional目录下  
      
2. 在hibernate.cfg.xml的配置文件中编写配置

|  |
| --- |
| <!-- C3P0的供应商 -->  <property name=*"hibernate.connection.provider\_class"*>org.hibernate.connection.C3P0ConnectionProvider</property>    <!-- 最小连接 -->    <property name=*"hibernate.c3p0.min\_size"*>5</property>    <!-- 最大连接数 -->    <property name=*"hibernate.c3p0.max\_size"*>10</property>    <!-- 每120秒检查空闲连接 -->    <property name=*"hibernate.c3p0.timeout"*>120</property> |

### 自动建表

该配置是让hibernate是否自动建立相应的表。也就是说，你数据库可以没有表，hibernate可以自动给你建立一个。



* create-drop:在程序运行时,（创建session工厂的时候）会自动建表,在程序停止时(关闭sessionFactory时),表删除

如何证明 create-drop的功能？

直接使用debug的方式，断点查看

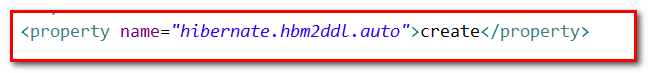


第一步，观察表是否创建成功（标志：原先的表被删除，数据丢失）

第二步，实现保存操作，观察表中是否有数据

第三步，程序运行结束，观察表是否被删除

* create:在程序运行时,如果表不存在,自动建表,如果存在,则不进行任何操作(mysql会先删除再创建)



* update:在程序运行时,如果表不存在,自动建表,如果表存在,则检查表和类的结构是否一致,如果不一致,则更新表结构.(推荐)

update:

在Customer类中加入如下代码：

|  |
| --- |
| //客户联系地址  **private** String cust\_address;    **public** String getCust\_address() {  **return** cust\_address;  }  **public** **void** setCust\_address(String cust\_address) {  **this**.cust\_address = cust\_address;  } |

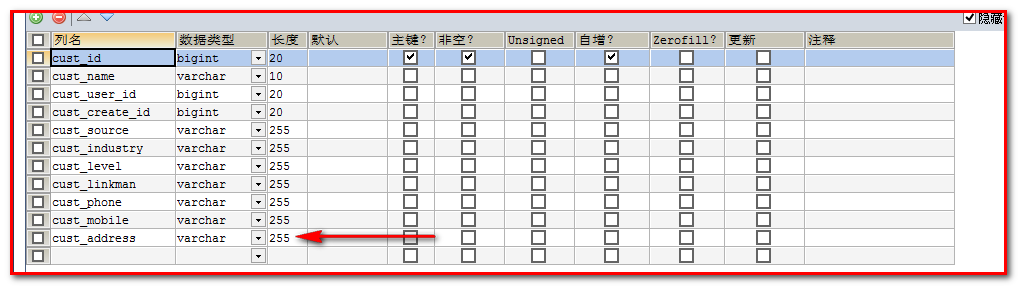
在Customer.hbm.xml中添加如下代码：

|  |
| --- |
| <property name=*"cust\_address"* column=*"cust\_address"*></property> |

修改hibernate.cfg.xml中建表配置：

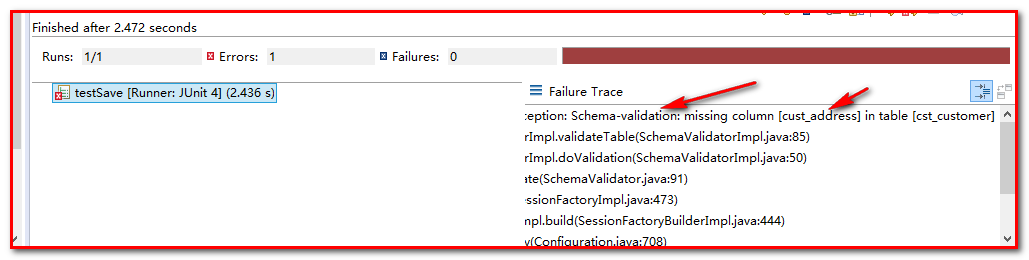
|  |
| --- |
| <property name=*"hibernate.hbm2ddl.auto"*>update</property> |

测试

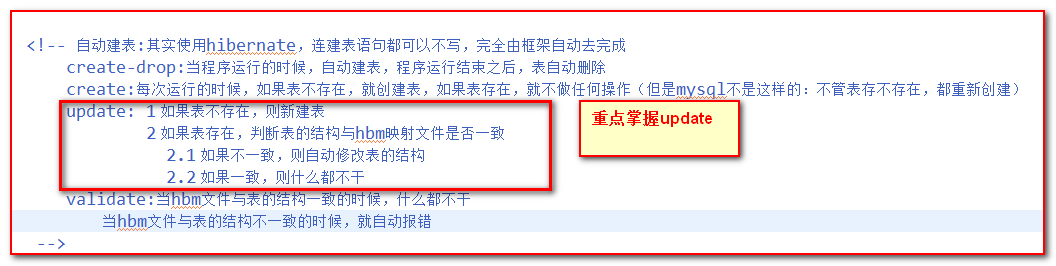


如果当表中的字段比配置文件多的时候，那么表中多的字段，hibernate将无法维护。（因为没有映射）

* validate:在程序运行时,如果表结构和类不一致,则报错!（表中没有相应的字段，则报错，表中有冗余字段，不报错）



【自动建表小结】



# 技术分析之Hibernate常用的接口和类

## Configuration类和作用

1. Configuration类  
  \* Configuration对象用于配置并且启动Hibernate。  
  \* Hibernate应用通过该对象来获得对象-关系映射文件中的元数据，以及动态配置Hibernate的属性，然后创建SessionFactory对象。  
          
  \* 简单一句话：加载Hibernate的配置文件，可以获取SessionFactory对象。  
      
2. Configuration类的其他应用（了解）  
  \* 加载配置文件的种类，Hibernate支持xml和properties类型的配置文件，在开发中基本都使用XML配置文件的方式。  
  \* 如果采用的是properties的配置文件，那么通过Configuration configuration = new Configuration();就可以加载配置文件  
  \* 但是需要自己手动加载映射文件  
  \* 例如：config.addResource("com/igeek/domain/Customer.hbm.xml");  
              
  \* 如果采用的XML的配置文件，通过Configuration configuration = new Configuration().configure();加载配置文件

## SessionFactory接口

1. 是工厂类，是生成Session对象的工厂类  
2. SessionFactory类的特点  
  \* 由Configuration通过加载配置文件创建该对象。  
  \* SessionFactory对象中保存了当前的数据库配置信息和所有映射关系以及预定义的SQL语句。同时，SessionFactory还负责维护Hibernate的二级缓存。  
  \* 预定义SQL语句  
  \* 使用Configuration类创建了SessionFactory对象是，已经在SessionFacotry对象中缓存了一些SQL语句  
  \* 常见的SQL语句是增删改查（通过主键来查询）  
  \* 这样做的目的是效率更高  
          
  \* 一个SessionFactory实例对应一个数据库，应用从该对象中获得Session实例。  
  \* SessionFactory是线程安全的，意味着它的一个实例可以被应用的多个线程共享。  
  \* SessionFactory是重量级的，意味着不能随意创建或销毁它的实例。如果只访问一个数据库，只需要创建一个SessionFactory实例，且在应用初始化的时候完成。  
  \* SessionFactory需要一个较大的缓存，用来存放预定义的SQL语句及实体的映射信息。另外可以配置一个缓存插件，这个插件被称之为Hibernate的二级缓存，被多线程所共享  
      
3. 总结  
  \* 一般应用使用一个SessionFactory,最好是应用启动时就完成初始化。

### 编写HibernateUtil的工具类

1. 在com.igeek.util包下新建工具类HibernateUtils,具体代码如下

|  |
| --- |
| **package** com.igeek.util;  **import** org.hibernate.Session;  **import** org.hibernate.SessionFactory;  **import** org.hibernate.cfg.Configuration;  **public** **class** HibernateUtils {  //定义获取配置文件的常量  **private** **static** **final** Configuration ***CONFIG***;  **private** **static** **final** SessionFactory ***FACTORY***;    **static**{  //给常量赋值  ***CONFIG*** = **new** Configuration().configure();  ***FACTORY*** = ***CONFIG***.buildSessionFactory();  }    //获取session对象  **public** **static** Session openSession(){  **return** ***FACTORY***.openSession();  }  } |

1. 在com.igeek.demo3中创建TestSessionFactory类，代码如下：

|  |
| --- |
| **package** com.igeek.demo3;  **import** org.hibernate.Session;  **import** org.hibernate.Transaction;  **import** org.junit.Test;  **import** com.igeek.demo1.Customer;  **import** com.igeek.util.HibernateUtils;  **public** **class** TestSessionFactory {  @Test  **public** **void** run1(){  //加载配置文件、创建SessionFactory、创建Session  Session session = HibernateUtils.*openSession*();  //开启事务  Transaction tr = session.beginTransaction();  //保存数据  Customer customer = **new** Customer();  customer.setCust\_name("美美");  customer.setCust\_mobile("133");  //提交事务  tr.commit();  //关闭session  session.close();  //web开发中不要关闭SessionFactory  }  } |

## Session接口

1. 概述  
    \* Session是在Hibernate中使用最频繁的接口。也被称之为持久化管理器。它提供了和持久化有关的操作，比如添加、修改、删除、加载和查询实体对象  
    \* Session 是应用程序与数据库之间交互操作的一个单线程对象，是 Hibernate 运作的中心  
    \* Session是线程不安全的  
    \* 所有持久化对象必须在 session 的管理下才可以进行持久化操作  
    \* Session 对象有一个一级缓存，显式执行 flush 之前，所有的持久化操作的数据都缓存在 session 对象处  
    \* 持久化类与 Session 关联起来后就具有了持久化的能力  
         
   2. 特点  
    \* 不是线程安全的。应避免多个线程使用同一个Session实例  
    \* Session是轻量级的，它的创建和销毁不会消耗太多的资源。应为每次客户请求分配独立的Session实例  
    \* Session有一个缓存，被称之为Hibernate的一级缓存。每个Session实例都有自己的缓存  
         
   3. 常用的方法  
    \* save(obj)               -- 保存数据

\* get(Class,id)             -- 通过主键查询该条记录  
 \* delete(obj)              -- 删除数据

\* update(obj)              -- 修改数据

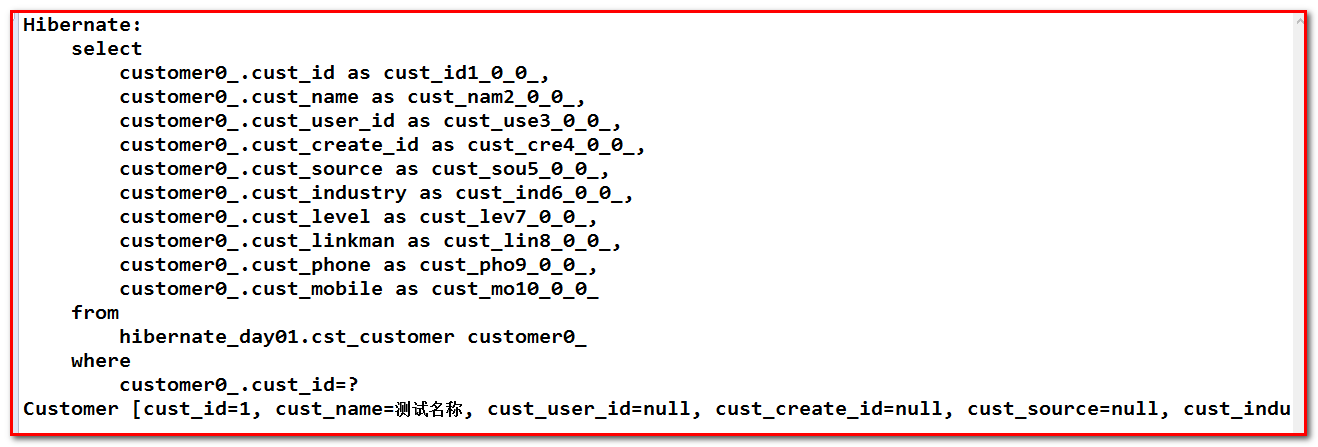
\* saveOrUpdate(obj)           -- 保存或者修改  
 \* createQuery(String hql)        -- 查询所有的数据

### session中的get方法

在com.igeek.demo3中创建TestSession类，添加get方法，代码如下：

|  |
| --- |
| **package** com.igeek.demo3;  **import** org.hibernate.Session;  **import** org.hibernate.Transaction;  **import** org.junit.After;  **import** org.junit.Before;  **import** org.junit.Test;  **import** com.igeek.demo1.Customer;  **import** com.igeek.util.HibernateUtils;  **public** **class** TestSession {  **private** Session session;  **private** Transaction tr;  //在测试方法之前运行  @Before  **public** **void** init(){  //获取session  session = HibernateUtils.*openSession*();  //开启事务  tr = session.beginTransaction();  }    /\*\*  \* 根据id获取数据  \*/  @Test  **public** **void** get(){  //第一个参数：返回对象的类型  //第二个参数：数据表中的数据id,注意数据类型，在Customer id为Long类型，所以这里要写1L  Customer customer = session.get(Customer.**class**,1L);  //在Customer类中添加toString()方法  System.***out***.println(customer);  }  //在测试方法之后运行  @After  **public** **void** destroy(){  //提交事务  tr.commit();  //释放资源  session.close();  }  } |

测试结果：

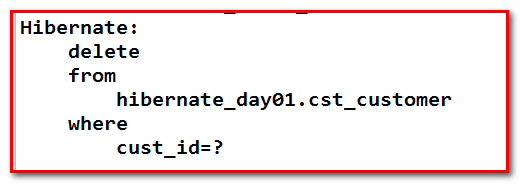


### session中的delete方法

在TestSession类，添加delete方法

|  |
| --- |
| /\*\*  \* 删除数据  \*/  @Test  **public** **void** delete(){  //1.根据id获取要删除的数据  Customer customer = session.get(Customer.**class**,1L);  //2.调用delete方法删除  session.delete(customer);  } |

测试结果

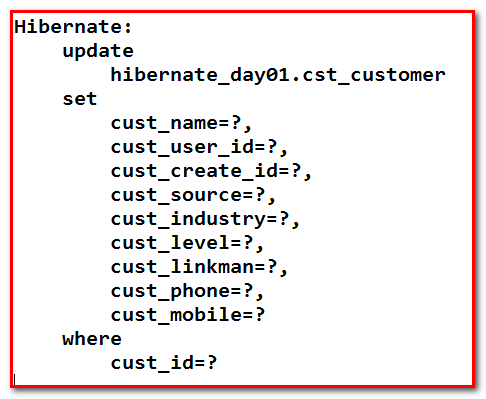


### session中的update方法

在TestSession类，添加update方法

|  |
| --- |
| @Test  **public** **void** update(){  //1.根据id获取数据，把数据封装在对象中  Customer customer = session.get(Customer.**class**, 2L);  //2.修改对象中的信息  customer.setCust\_name("美美");  session.update(customer);  } |

测试结果：



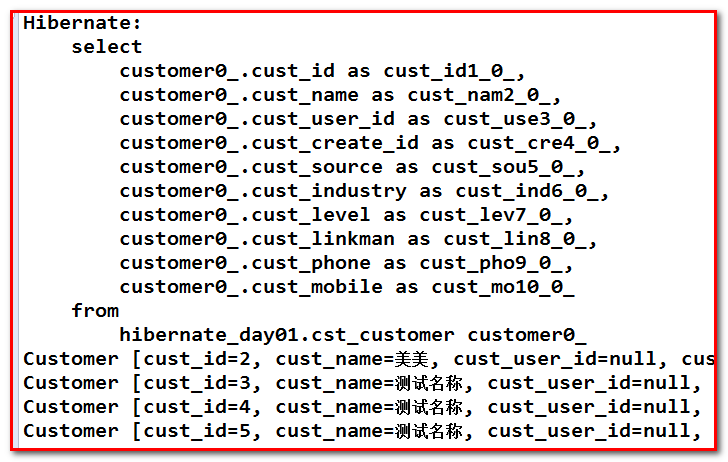
注意：hibernate会根据主键id更新所有字段

### session中的createQuery方法

在TestSession类，添加queryAll方法

|  |
| --- |
| /\*\*  \* 查询所有数据  \*/  @Test  **public** **void** queryAll(){  //1.创建查询所有数据的接口  //参数为hql语句，Customer为类名不是表名（即不能写成customer）  Query query = session.createQuery("from Customer");  //2.调用list方法获取数据  List<Customer> data = query.list();  //3.打印数据  **for** (Customer customer : data) {  System.***out***.println(customer);  }  } |

测试结果：



## Transaction接口

1. Transaction是事务的接口  
   2.常用的方法  
    \* commit()             -- 提交事务  
    \* rollback()            -- 回滚事务  
         
   3.特点  
    \* Hibernate框架默认情况下事务不自动提交.需要手动提交事务  
    \* 如果没有开启事务，那么每个Session的操作，都相当于一个独立的事务

### 事务代码编写

|  |
| --- |
| **package** com.igeek.demo3;  **import** org.hibernate.Session;  **import** org.hibernate.Transaction;  **import** org.junit.Test;  **import** com.igeek.demo1.Customer;  **import** com.igeek.util.HibernateUtils;  **public** **class** TestTransaction {  @Test  **public** **void** run1(){  //1.获取session  Session session = HibernateUtils.*openSession*();  //2.开启事务  Transaction tr = session.beginTransaction();  //3.在try...catch块中使用事务，开发中在业务层使用  **try** {  //新建customer对象  Customer c = **new** Customer();  c.setCust\_name("冠希");  session.save(c);  //提交事务  tr.commit();  } **catch** (Exception e) {  //回滚事务  tr.rollback();  }**finally** {  //释放资源  session.close();  }  }  } |

课后作业

使用hibernate框架完成crm系统中客户关系管理的增删改查操作