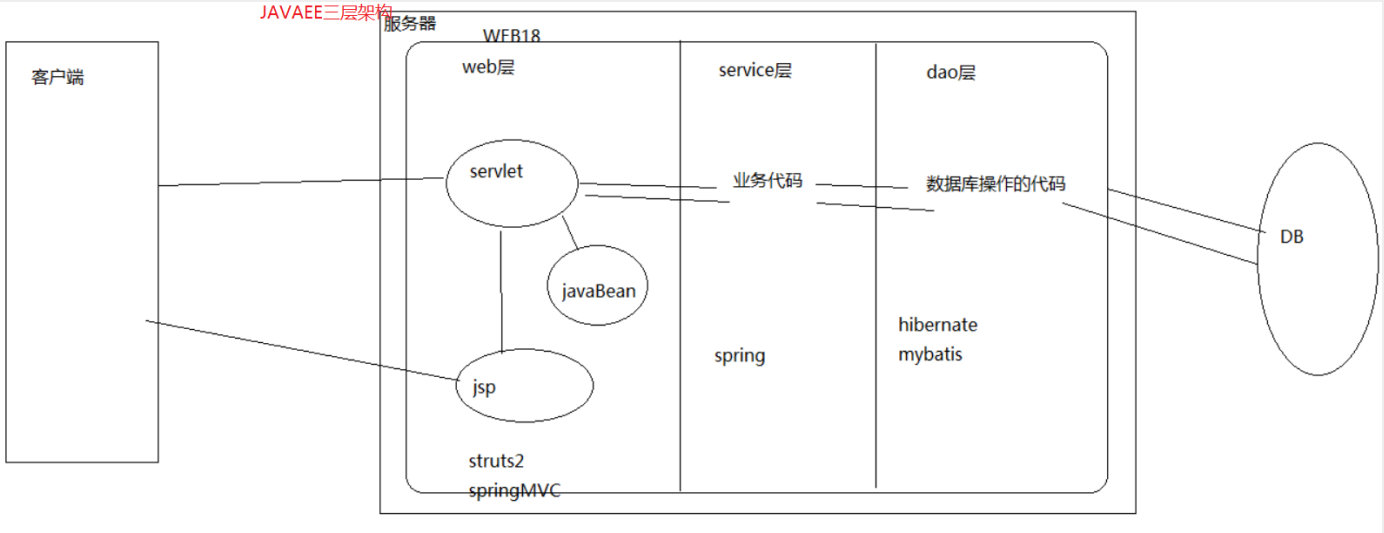
# Hibernate学习日志一

## 1 WEB内容回顾

### 1.1 JAVAEE三层架构




1. WEB层：Struts2框架
2. service层：spring框架（其实是一个项目容器，统筹三层）
3. dao层：hibernate框架，mybatis框架等

### 1.2 MVC设计模式

MVC设计模式:Model-View-Controller简写。
MVC是软件工程中的一种软件架构模式，它是一种分离业务逻辑和显示界面的设计方法。它把软件系统分为二个基本部分:模型（Model）、视图（View)和控制器（Controller）。
模型Model: 编写程序应用的功能(实现算法等等)、数据库管理;
视图View: 界面设计人员进行图形界面设计;
控制器Controller: 对请求进行处理，负责请求转发;

### 1.3 三层架构与MVC设计模式的关系

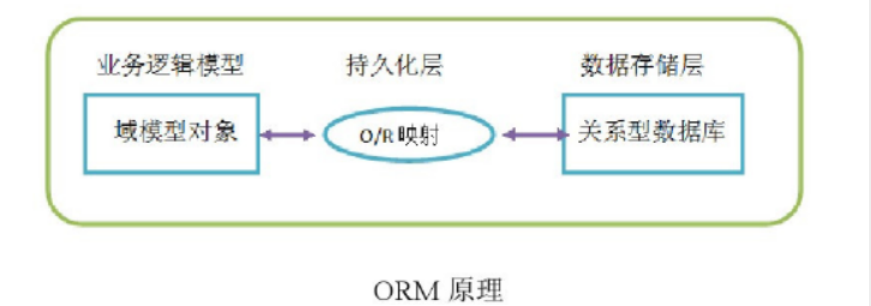
周末再总结，记着

## 2 Hibernate框架概述与orm思想

### 2.1 概述

使用传统的JDBC开发应用系统时，如果是小型应用系统，并不觉得有什么麻烦，但是对于大型应用系统的开发，使用JDBC就会显得力不从心。例如对几十、几百张包含几十个字段的表进行插入

操作时，编写的SQL语句不但很长，而且繁琐，容易出错;在读取数据时，需要写多条getXxx语句从结果集中取出各个字段的信息，不但枯燥重复，并且工作量非常大。为了提高数据访问层的编程效率，Gavin King开发出了一个当今最流行的的ORM框架，它就是Hibernate框架。



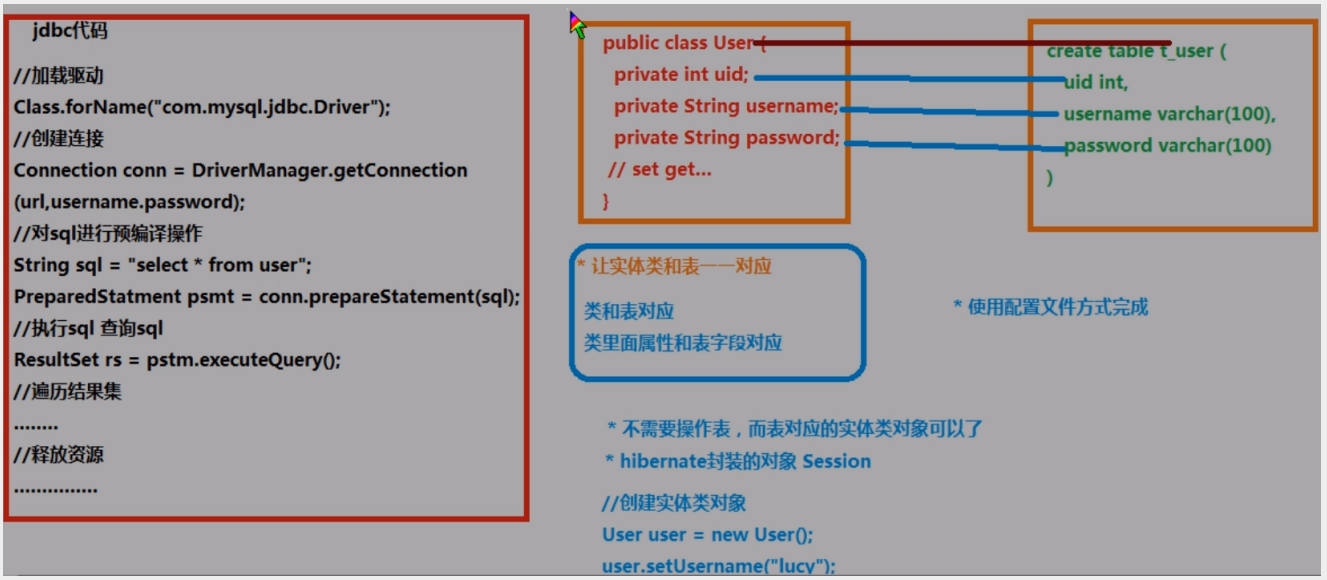
ORM 原理

框架：半成品，在这个基础上少写代码
Hibernate框架：

1. 应用在dao层
2. 操作数据库，底层封装的就是jdbc
3. 更偏向面向对象编程
4. 开源轻量级
5. 持久层的orm框架

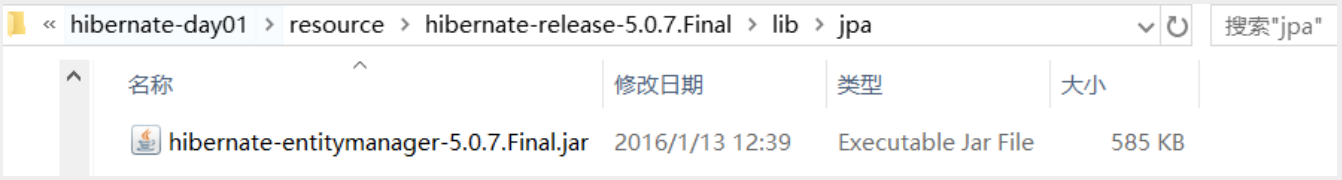
## 2.2 orm（Object Relational Mapping）思想

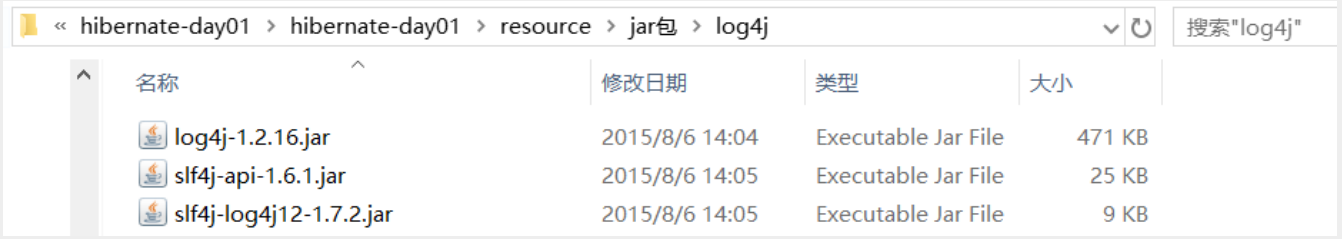
1. 在web阶段学习的javabean，在这里叫实体类
2. 对象关系映射，类似
   文字描述：
   （1）让实体类和数据库表进行一一对应关系，让实体类首先和数据表对应，让实体类的属性与数据表的字段对应。
   （2）不需要直接操作数据库，只需要操作对应表的实体类对象。




# 3 Hibernate环境搭建与入门

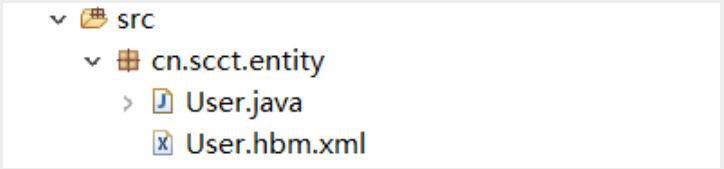
1. 导包(required、jpa、日志包、mysql驱动包)：

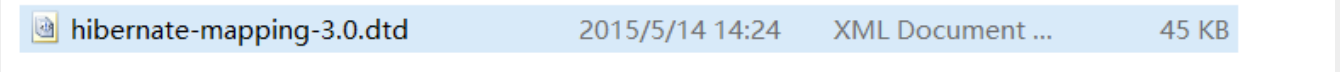


2. 创建实体类



```java
package cn.scct.entity;
                    要求每个实体类都有一个唯一的属性，一般用
public class User {
    private int uid;  int类型表示的id之类的
    private String username;
    private String password;
    private String address;
    public int getUid() {
        return uid;
    }                   需要为每个属性生成get和set方法
    public void setUid(int uid) {
        this.uid = uid;
    }                   alt+s
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
```

3. 创建数据库，并不需要创建表
4. 配置实体类与数据库表的映射关系（见配置文件详解）
   （1）创建xml格式的配置文件
   （2）建议写在实体类的包下，名字为实体类名称.hbm.xml
   （3）引入xml格式的约束，找得到就行




```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">
```

5. 创建hibernate核心配置文件（见配置文件详解）
   （1）核心配置文件格式为xml，名称和位置固定
   （2）在src下，名称为hibernate.cfg.xml
   （3）约束

```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">
    <!-- 本文件一定要创建在src下 -->
```

hibernate在操作过程中只会加载核心配置文件

6. 创建测试类并允许测试类即可创建表

第一步：加载hibernate核心配置文件

```
    @Test
    public void testAdd(){
//      第一步 加载hibernate核心配置文件
            /*
             * 到src下找名称为hibernate.cfg.xml
             * 在hibernate里面封装对象
             * */
//       Configuration cfg=new Configuration();
//       cfg.configure();
```

第二步：创建SessionFactory对象，这一步会创建表

```
//      第二步 创建SessionFactory对象
        //创建表
                                        // ctrl+2+L这个快捷键可自动补全代码，极大提升编码效率!
                                        //注：ctrl和2同时按完以后释放，再快速按L.
//       SessionFactory sessionFactory = cfg.buildSessionFactory();
```

第三步：创建Session对象，类似于连接

```
        Session session = sessionFactory.openSession();
```

第四步：开启事务

```
//       第四步 开启事务
        Transaction transaction = session.beginTransaction();
```

第五步：操作数据库，crud操作

```
//      第五步 写具体逻辑crud操作
           //添加功能
        User user=new User();
        user.setUsername("lisi");
        user.setAddress("beijing");
        user.setPassword("232323");
           //调用session的方法实现添加
        session.save(user);
```

第六步：提交事务

```
//      第六步 提交事务
        transaction.commit();
```

第七步：关闭资源

```
//      第七步 关闭资源
        session.close();
        sessionFactory.close();
```

# 4 Hibernate配置文件详解

## 4.1 Hibernate映射配置文件

1. 建议写在实体类的包下，名字为实体类名称.hbm.xml
2. class标签name属性写实体类的全包名，table属性写要创建的表的表名

```
<hibernate-mapping>
  <!-- 1.配置类和表对应
        class标签
        name属性：实体类全路径
        table属性：数据表名称 -->
  <class name="cn.scct.entity.User" table="t_User">
```

3. id标签与表的主键对应，并在generator标签中设置增长策略

```
        <!-- 2.配置实体类id和表id对应
             hibernate要求实体类有一个唯一的属性值
             hibernate要求表有唯一的字段
             -->
        <!-- id标签
            name属性：实体类里面id属性名称
            column属性：生成表字段名称
         -->
         <id name="uid" column="uid">
             <!-- 设置自增长策略
               native设置为主键自动增长
             -->
             <generator class="native"></generator>
         </id>
```

4. property代表实体类的属性，与表的字段对应，配置中为column标签，注意column不写默认与property或id标签值一样

```
        <!-- 3. 配置其他属性和表字段对应
           name属性：实体类属性名称
           column属性：生成表字段名称-->
         <property name="username" column="username"></property>
         <property name="password" column="password"></property>
         <property name="address" column="address"></property>
  </class>
</hibernate-mapping>
```

## 4.2 Hibernate核心配置文件

```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">
    <!-- 本文件一定要创建在src下 -->
<hibernate-configuration>
    <session-factory>

    <!-- 第一步：配置数据库信息
    #hibernate.dialect org.hibernate.dialect.MySQLDialect
    #hibernate.dialect org.hibernate.dialect.MySQLInnoDBDialect
    #hibernate.dialect org.hibernate.dialect.MySQLMyISAMDialect
    #hibernate.connection.driver_class com.mysql.jdbc.Driver
    #hibernate.connection.url jdbc:mysql:///test
    #hibernate.connection.username gavin
    #hibernate.connection.password
     -->
         <!-- 数据库驱动 -->
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
         <!-- 数据库url -->
        <property name="hibernate.connection.url">jdbc:mysql:///hibernate_day01</property>
         <!-- 数据库连接用户名 -->
        <property name="hibernate.connection.username">root</property>
         <!-- 数据库连接密码 -->
        <property name="hibernate.connection.password">201805</property>

    <!-- 第二步：配置hibernate信息 可选的
     -->

        <!-- 2.1输出底层sql语句，将hibernate生成的sql语句打印到控制台 -->
        <property name="hibernate.show_sql">true</property>
        <!-- 2.2输出底层sql语句的格式将hibernate生成的sql语句格式化(语法缩进) -->
        <property name="hibernate.format_sql">true</property>
        <!-- 2.3 建表的结构
        ## auto schema export  自动导出表结构. 自动建表
        #hibernate.hbm2ddl.auto create      自动建表.每次框架运行都会创建新的表.以前表将会被覆盖,表数据会丢失.(开发环境中测试使用)
        #hibernate.hbm2ddl.auto create-drop 自动建表.每次框架运行结束都会将所有表删除.(开发环境中测试使用)
        #hibernate.hbm2ddl.auto update(推荐使用) 自动生成表.如果已经存在不会再生成.如果表有变动.自动更新表(不会删除任何数据).
        #hibernate.hbm2ddl.auto validate    校验.不自动生成表.每次启动会校验数据库中表是否正确.校验失败.
         -->
        <property name="hibernate.hbm2ddl.auto">update</property>
        <!-- 2.4 数据库方言
            不同的数据库中,sql语法略有区别. 指定方言可以让hibernate框架在生成sql语句时.针对数据库的方言生成.
            在mysql里面实现分页 关键字为limit，只能使用在mysql里面
            在oracle数据库，实现分页fownum
            让hibernate框架识别不同数据可自己特有的语句
         -->
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

    <!-- 第三步：
        路径书写：填写src下的路径引入的映射文件
     -->
        <mapping resource="cn/scct/entity/User.hbm.xml" />


    </session-factory>
</hibernate-configuration>
```

# 5 Hibernate API

## 5.1 configuration对象

```
//      Configuration cfg=new Configuration();
//      cfg.configure();
```

**加载核心配置文件** 在使用Hibernate时，首先要创建Configuration实例，Configuration实例主要用于启动、加载、管理hibernate的配置文件信息。在启动Hibernate的过程中，Configuration实例首先确定Hibernate配置文件的位置，然后读取相关配置，最后创建一个唯一的SessionFactory实例。Configuration对象只存在于系统的初始化阶段，它将SessionFactory创建完成后，就完成了自己的使命。Hibernate通常使用`Configuration config=new Configuration().configure()`的方式创建实例，此种方式默认会去src下读取hibernate. cfg.xml配置文件。如果不想使用默认的hibernate. cfg. xml配置文件，而是使用指定目录下(或自定义)的配置文件，则需要向configure()方法中传递一个文件路径的参数，其代码写法如下: `Configuration config=new Configuration().configure("xml文件位置")`

**加载映射文件(很少使用)**

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

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

## 5.2 SessionFactory对象（重要）

1. 使用configuration对象创建SessionFactory对象

   （1）创建SessionFactory过程中，根据核心配置文件中的数据库配置，映射文件在核心文件中的配置，到数据库中直接创建表。

   即根据它：

```
    <!-- 第一步：配置数据库信息
    #hibernate.dialect org.hibernate.dialect.MySQLDialect
    #hibernate.dialect org.hibernate.dialect.MySQLInnoDBDialect
    #hibernate.dialect org.hibernate.dialect.MySQLMyISAMDialect
    #hibernate.connection.driver_class com.mysql.jdbc.Driver
    #hibernate.connection.url jdbc:mysql:///test
    #hibernate.connection.username gavin
    #hibernate.connection.password
     -->
        <!-- 数据库驱动 -->
       <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <!-- 数据库url -->
       <property name="hibernate.connection.url">jdbc:mysql:///hibernate_day01</property>
        <!-- 数据库连接用户名 -->
       <property name="hibernate.connection.username">root</property>
        <!-- 数据库连接密码 -->
       <property name="hibernate.connection.password">201805</property>
```

   和它：

```
    <!-- 第三步：
        路径书写：填写src下的路径引入的映射文件
     -->
       <mapping resource="cn/scct/entity/User.hbm.xml" />

```

   创建表：

```
        <!-- 2.3 建表的结构
        ## auto schema export  自动导出表结构. 自动建表
        #hibernate.hbm2ddl.auto create      自动建表.每次框架运行都会创建新的表.以前表将会被覆盖,表数据会丢失.(开发环境中测试使用)
        #hibernate.hbm2ddl.auto create-drop 自动建表.每次框架运行结束都会将所有表删除.(开发环境中测试使用)
        #hibernate.hbm2ddl.auto update(推荐使用) 自动生成表.如果已经存在不会再生成.如果表有变动.自动更新表(不会删除任何数据).
        #hibernate.hbm2ddl.auto validate     校验.不自动生成表.每次启动会校验数据库中表是否正确.校验失败.
         -->
        <property name="hibernate.hbm2ddl.auto">update</property>
```

（2）创建SessionFactory过程中，耗费资源，可以抽取SessionFactory对象的创建过程。

```
package cn.scct.hibernateUtils;

import org.hibernate.SessionFactory;

public class HibernateUtils {
    private static Configuration cfg;
    private static SessionFactory sessionFactory;

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

    public static SessionFactory getSessionFactory(){
        return sessionFactory;
    }

}
```

## 5.3 Session对象（重要）

1. Session类似于jdbc中的connection
2. 调用session里面的不同方法可以实现crud操作
   （1）添加save方法
   （2）修改update方法
   （3）删除delete方法
   （4）根据id 查询get方法
3. Session对象是一个单线程对象。

```
        Session session = sessionFactory.openSession();
//      第四步 开启事务
        Transaction transaction = session.beginTransaction();
//      第五步 写具体逻辑crud操作
            //添加功能
        User user=new User();
        user.setUsername("lisi");
        user.setAddress("beijing");
        user.setPassword("232323");
            //调用session的方法实现添加
        session.save(user);
```

## 5.4 Transaction对象

开启事务：`Session对象.beginTransaction()`
提交事务：`Transaction对象.commit()`
回滚事务：`Transaction对象.rollback()`