# Hibernate第一天

## 本日重点

* 了解什么是ORM
* 完成Hibernate开发环境搭建
* 掌握Hibernate的执行流程
* 掌握Hibernate开发中常用的API

## 第一节课

### 框架概述

框架：是来解决复杂问题的一种结构或者是一系列的组件

为什么要使用框架？

简化开发、提高开发效率

框架是分领域的：

Hibernate/Mybatis/Apache ORM/Oralce TopLink：主要解决持久层的问题（数据如何保存、数据查询…）

Struts2/Spring MVC：一个MVC框架

Spring：它是一种IOC、AOP框架

Shiro：权限框架

Apache CXF：Web服务框架

到底该如何去学习框架？

1. 这个框架**主要**是来解决什么问题
2. 快速入门（HelloWorld）
   1. 搭建环境（导入JAR包 + 配置文件）
   2. 编写HelloWorld代码（非常简单的、可以运行的一个程序）
3. 熟悉框架的各个核心模块（API）

### JavaEE开发中常用框架

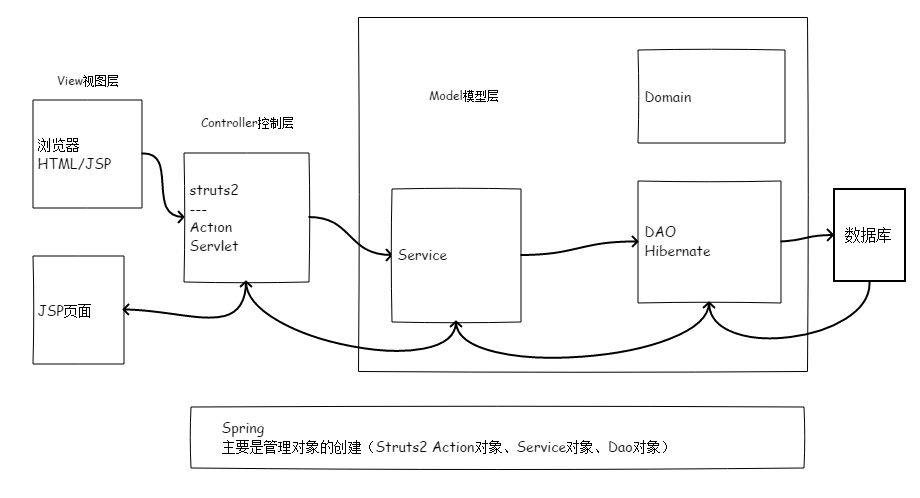
SSH（SSH2）：struts2 hibernate spring

主要应用于企业开发

SSM（SSI）：spring MVC、spring、Mybatis

在互联网公司、在创业公司

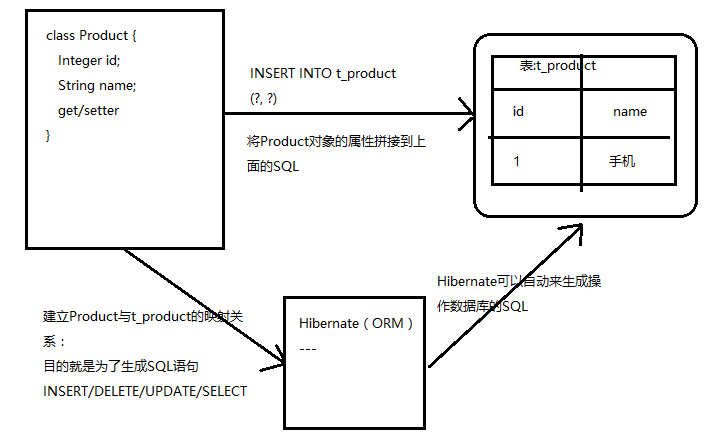
MVC



### Hibernate框架介绍

#### 什么是Hibernate？

ORM： Object Relation Mapping



#### Hibernate的学习重点

* 学习搭建Hibernate环境，编写入门程序
* 学习Hibernate常用API（增删改查）
* 关联映射（一对一、一对多、多对多）
* 检索策略（**LEFT JOIN**、RIGHT JOIN、自然连接）
* Hibernate优化

### Hibernate环境准备

* Hibernate 5.0.7

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

### Hibernate快速入门（准备数据库、实体类）

包名：com/cn/org . **itheima** . 模块名（使用小写）

类名：以大写驼峰式命名（Person/Product）——**P**roduct**C**ategory

字段名：小写驼峰式命名（id/name）

方法名：小写驼峰式命名（getOrder）

常量名：全部用大写，用下划线分隔（**SESSION\_USER** = 1）

* 编写实体类

**public** **class** User {

**private** Integer id;

**private** String name;

**private** String address;

**public** Integer getId() {

**return** id;

}

**public** **void** setId(Integer id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getAddress() {

**return** address;

}

**public** **void** setAddress(String address) {

**this**.address = address;

}

@Override

**public** String toString() {

**return** "User [id=" + id + ", name=" + name + ", address=" + address

+ "]";

}

}

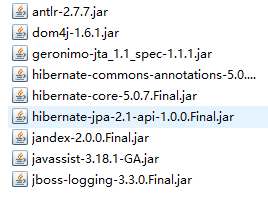
* 编写代码的几个阶段：

1. 复制粘贴代码——能够看懂老师的代码（不要自己造轮子）
2. 参考老师的笔记（代码）自己去编写
3. 离开所有的笔记、代码独立编写

## 第二节课

### Hibernate快速入门（导JAR包）

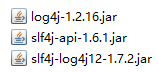
* Hibernate框架目录中的lib/required所有的JAR包



* 数据库驱动JAR包



* 日志JAR包



* 拷贝日志配置文件

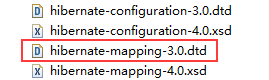
将 hibernate框架目录的project/etc/log4j.properties文件拷贝到项目的**src**目录下

### Hibernate快速入门（映射文件）

* xxx.hbm.xml：实体类与数据库表的映射文件

位置：实体类所在的包下，名字：类名.hbm.xml

DTD约束可以去hibernate-core-5.0.7.Final.jar的org.hibernate包下拷贝。注意：不要拷贝错了。



catelog：数据库名

<?xml version=*"1.0"*?>

<!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.itheima.domain.User"* table=*"t\_d1h01\_user"*>

<id name=*"id"* column=*"id"*>

<generator class=*"native"*></generator>

</id>

<property name=*"name"* length=*"20"*></property>

<property name=*"address"* length=*"50"*></property>

</class>

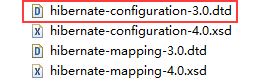
</hibernate-mapping>

### 配置Hibernate核心配置文件

类名、映射文件的路径都从Eclipse去拷贝，尽量不要手敲

* hibernate.cfg.xml：Hibernate框架的核心配置文件

Hibernae核心配置文件的dtd约束也可以从hibernate-core-5.0.7.Final.jar的org.hibernate包下拷贝。



Hibernate中的属性可以从Hibernate框架目录中的project/etc/hibernate.properties尽量从该文件中拷贝，避免因为拼写错误导致程序无法执行的问题。

<!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</property>

<property name=*"hibernate.connection.username"*>root</property>

<property name=*"hibernate.connection.password"*>000000</property>

<property name=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</property>

<property name=*"hibernate.show\_sql"*>true</property>

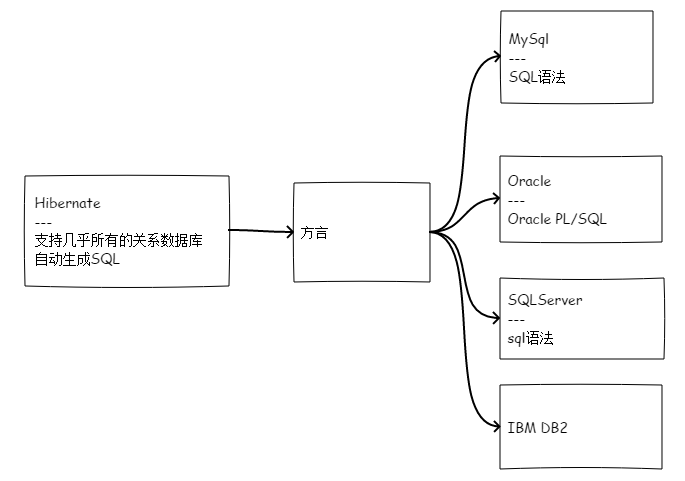
<property name=*"hibernate.format\_sql"*>true</property>

<mapping resource=*"com/itheima/domain/User.hbm.xml"*/>

</session-factory>

</hibernate-configuration>

* 方言



### 编写测试代码

// 新增

@Test

**public** **void** testAdd01() {

Configuration cfg = **new** Configuration().configure();

SessionFactory sessionFactory = cfg.buildSessionFactory();

Session session = sessionFactory.openSession();

Transaction t = session.beginTransaction();

User u = **new** User();

u.setName("王二狗");

u.setAddress("中国南京");

session.save(u);

t.commit();

session.close();

sessionFactory.close();

}

Hibernate执行的SQL语句：

insert

into

t\_d1h01\_user

(name, address)

values

(?, ?)

## 第三节课

### 测试CRUD操作

注意：使用Hibernate来完成操作数据库是通过**直接操作对象**来完成。

使用Query去查询所有数据的时候，需要传入的是HQL（”from Customer”）,注意不要写成表名，而需要写成对应的实体名称。

* 根据ID查询

// 根据ID查询

@Test

**public** **void** testGetById01() {

Configuration cfg = **new** Configuration().configure();

SessionFactory sessionFactory = cfg.buildSessionFactory();

Session session = sessionFactory.openSession();

Transaction t = session.beginTransaction();

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

System.***out***.println(user);

t.commit();

session.close();

sessionFactory.close();

}

Hibernate执行的SQL语句：

select

user0\_.id as id1\_0\_0\_,

user0\_.name as name2\_0\_0\_,

user0\_.address as address3\_0\_0\_

from

t\_d1h01\_user user0\_

where

user0\_.id=?

* 更新操作

// 更新

@Test

**public** **void** testUpdate01() {

Configuration cfg = **new** Configuration().configure();

SessionFactory sessionFactory = cfg.buildSessionFactory();

Session session = sessionFactory.openSession();

Transaction t = session.beginTransaction();

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

user.setName("王吉祥");

session.update(user);

t.commit();

session.close();

sessionFactory.close();

}

Hibernate执行的SQL语句

update

t\_d1h01\_user

set

name=?,

address=?

where

id=?

* 查询所有数据

// 列表查询

@Test

**public** **void** testFind01() {

Configuration cfg = **new** Configuration().configure();

SessionFactory sessionFactory = cfg.buildSessionFactory();

Session session = sessionFactory.openSession();

Transaction t = session.beginTransaction();

Query q = session.createQuery("from User");

System.***out***.println(q.list());

t.commit();

session.close();

sessionFactory.close();

}

Hibernate执行的SQL语句

select

user0\_.id as id1\_0\_,

user0\_.name as name2\_0\_,

user0\_.address as address3\_0\_

from

t\_d1h01\_user user0\_

* 删除数据

// 删除

@Test

**public** **void** testDelete01() {

Configuration cfg = **new** Configuration().configure();

SessionFactory sessionFactory = cfg.buildSessionFactory();

Session session = sessionFactory.openSession();

Transaction t = session.beginTransaction();

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

session.delete(user);

t.commit();

session.close();

sessionFactory.close();

}

Hibernate执行的SQL语句

delete

from

t\_d1h01\_user

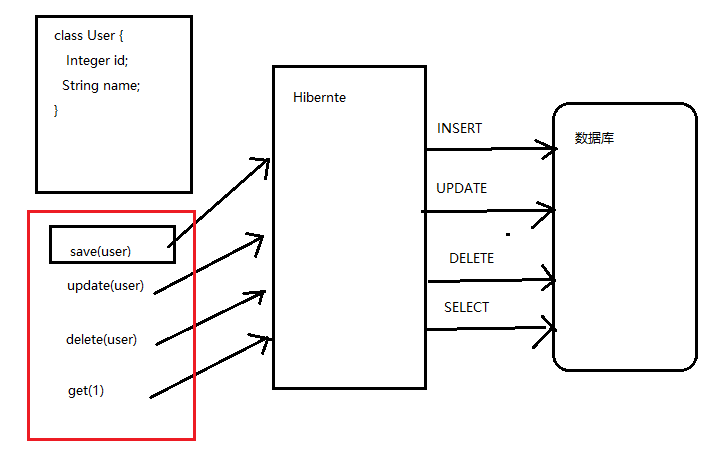
where

id=?

### Hibernate快速入门（总结）

Hibernate的工作原理：

* 通过Configuration().confige()读取hibernate.cfg.xml配置文件
* 由hibernate.cfg.xml中的<mapping resource=”….hbm.xml” />读取解析映射文件
* 通过config.buildSessionFactory()来得到SessionFactory
* 通过sessionFactory.openSession()来得到Session
* 通过session.beginTransaction()来开启事务
* 通过session.save/update/delete/get/load来执行数据库操作
* 通过session.getTransaction().commit()来提交事务
* 关闭session
* 关闭sessionFactory



### 详解Hibernate核心配置文件

* hibernate.cfg.xml（开发中常用，需掌握）

数据库相关配置

<property name=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<property name=*"hibernate.connection.url"*>jdbc:mysql:///hibernate</property>

<property name=*"hibernate.connection.username"*>root</property>

<property name=*"hibernate.connection.password"*>000000</property>

<property name=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</property>

Hibernate属性配置

<property name=*"hibernate.hbm2ddl.auto"*>create</property>

<property name=*"hibernate.show\_sql"*>true</property>

<property name=*"hibernate.format\_sql"*>true</property>

hibernate.hbm2ddl.auto配置

#hibernate.hbm2ddl.auto create-drop

#hibernate.hbm2ddl.auto create

#hibernate.hbm2ddl.auto update

#hibernate.hbm2ddl.auto validate

加载映射文件配置

<mapping resource=*"com/itheima/domain/User.hbm.xml"*/>

* hibernate.properties

### 详解Hibernate映射文件

<class name=*"com.itheima.domain.User"* table=*"t\_d1h02\_user1"*>

<id name=*"id"* column=*"id"*>

<generator class=*"native"*></generator>

</id>

<!-- 默认类型 -->

<!--<property name="name" length="20"></property> -->

<!-- JAVA类型 -->

<!-- <property name="name" length="20" type="java.lang.String"></property> -->

<!-- Hibernate类型 -->

<!--<property name="name" length="20" type="string"></property> -->

<!-- SQL类型 -->

<property name=*"name"*>

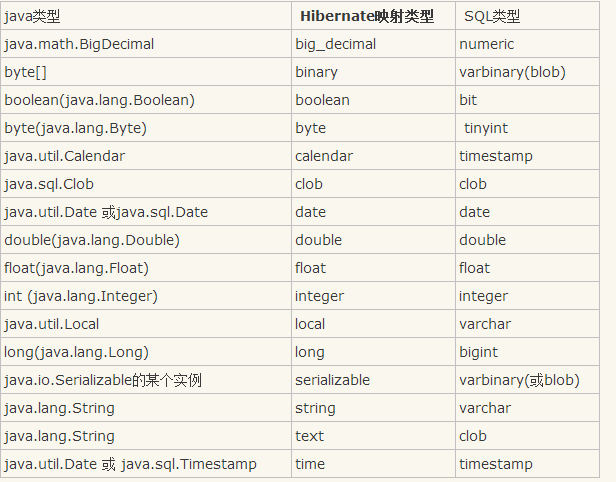
<column name=*"name"* sql-type=*"longtext"*></column>

</property>

<property name=*"address"* length=*"50"*></property>

</class>

* Hibernate数据类型映射



## 第四节课

### Hibernate API（介绍Configuration）

Configuration、SessionFactory、Session、Transaction、Query、Criteria

#### Configuration

主要用于加载Hibernate的配置文件。

new Configuration.configure()加载的是src/hibernate.cfg.xml文件。

new Configuration()加载的是src/hibernate.properties文件

* 加载指定的配置文件

Configuration cfg = **new** Configuration().configure("my.hibernate.cfg.xml");

SessionFactory factory = cfg.buildSessionFactory();

* 编程方式加载hbm映射文件

文件路径方式加载：

/\*\*

\* 编程加载hbm映射文件(文件路径方式)

\*/

@Test

**public** **void** test02() {

Configuration cfg = **new** Configuration().configure("my.hibernate.cfg.xml");

cfg.addResource("com/itheima/domain/User.hbm.xml");

SessionFactory factory = cfg.buildSessionFactory();

factory.close();

}

类方式加载

/\*\*

\* 编程加载hbm映射文件(类方式)

\*/

@Test

**public** **void** test03() {

Configuration cfg = **new** Configuration().configure("my.hibernate.cfg.xml");

cfg.addClass(User.**class**);

SessionFactory factory = cfg.buildSessionFactory();

factory.close();

}

### Hibernate API（介绍SessionFactory）

SessionFactory是通过Configuration得到。

Configuration cfg = **new** Configuration().configure();

SessionFactory sessionFactory = cfg.buildSessionFactory();

通过SessionFactory可以得到Session。一个项目中有一个SessionFactory即可。

* Hibernate工具类HibernateUtil

**public** **class** HibernateUtils {

**private** **static** SessionFactory *sessionFactory*;

**static** {

Configuration cfg = **new** Configuration().configure();

*sessionFactory* = cfg.buildSessionFactory();

}

**public** **static** Session getSession() {

**return** *sessionFactory*.openSession();

}

**public** **static** **void** close() {

*sessionFactory*.close();

}

}

* 使用HibernateUtils来进行数据操作

// 测试HibernateUtils保存

@Test

**public** **void** test01() {

Session session = HibernateUtils.*getSession*();

session.beginTransaction();

User u = **new** User();

u.setName("测试");

u.setAddress("地址");

session.save(u);

session.getTransaction().commit();

Query q = session.createQuery("from User");

System.***out***.println(q.list());

session.close();

HibernateUtils.*close*();

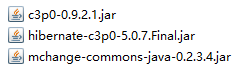
}

### 配置C3P0连接池

Connection Pool

* 导入C3P0 JAR包

可以从Hibernate框架目录中的lib\optional\c3p0中复制



* 在hibernate.cfg.xml核心配置中添加C3P0配置

C3P0相关的配置可以从project\etc\hibernate.properties中找到

<!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</property>

<property name=*"hibernate.connection.username"*>root</property>

<property name=*"hibernate.connection.password"*>000000</property>

<property name=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</property>

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

<property name=*"hibernate.show\_sql"*>true</property>

<property name=*"hibernate.format\_sql"*>true</property>

<!-- C3P0连接池 -->

<property name=*"hibernate.connection.provider\_class"*>org.hibernate.connection.C3P0ConnectionProvider</property>

<property name=*"hibernate.hibernate.c3p0.max\_size"*>10</property>

<property name=*"hibernate.c3p0.min\_size"*>2</property>

<mapping resource=*"com/itheima/User.hbm.xml"*/>

</session-factory>

</hibernate-configuration>

### Hibernate API（介绍Session）

Session接口是线程不安全的，需要在方法内部创建。

通过SessionFactory.openSession/getCurrentSession来获取。

#### 常用API

save：保存

update：更新

delete：删除

**get：根据ID查询（立即加载）**

**load：根据ID查询（懒加载）**

saveOrUpdate：没有ID、有ID执行update

createQuery：获取Query对象，来执行查询（HQL语句）

createSqlQuery：获取可以执行SQL的Query对象（SQL语句）

createCriteria：获取一个Criteria，来进行QBC查询

## 第五节课

### Hibernate API（介绍Transaction）

Transaction是Hibernate的事务接口，使用它可以进行事务提交、回滚操作。

使用Session.beginTransaction()获取事务。默认每一个数据库操作都会开启事务，但不会自动提交。

可以配置让事务自动提交（不建议使用）

<!-- 配置自动提交 -->

<property name=*"hibernate.connection.autocommit"*>true</property>

### Hibernate API（介绍Query）

Query主要完成查询操作。

Query query = session.createQuery(HQL)

Query query = session.createSqlQuery(SQL)

按住Ctrl + 2,马上按L，自动生成局部变量

### 使用Query查询全部和分页查询

* 查询所有数据

/\*\*

\* HQL查询

\*/

@Test

**public** **void** test02() {

String hql = "From User";

Session s = HibernateUtils.*getSession*();

Query q = s.createQuery(hql);

System.***out***.println(q.list());

s.close();

HibernateUtils.*close*();

}

* 分页查询

/\*\*

\* HQL分页查询

\*/

@Test

**public** **void** test03() {

String hql = "From User";

Session s = HibernateUtils.*getSession*();

Query q = s.createQuery(hql);

q.setFirstResult(0);

q.setMaxResults(10);

System.***out***.println(q.list());

s.close();

HibernateUtils.*close*();

}

### HQL投影查询

/\*\*

\* 投影查询（取出数据、返回的是Object[]）

\*/

@Test

**public** **void** test04() {

String hql = "SELECT o1.id, o1.name, o1.address FROM User o1";

Session s = HibernateUtils.*getSession*();

Query q = s.createQuery(hql);

q.setFirstResult(0);

q.setMaxResults(10);

Object[] row0 = (Object[])q.list().get(0);

System.***out***.println((Integer)row0[0]);

s.close();

HibernateUtils.*close*();

}

* 将查询的列封装为对象

/\*\*

\* 投影查询（取出数据封装为对象）

\*/

@Test

**public** **void** test06() {

String hql = "SELECT new User(o1.name, o1.address) FROM User o1";

Session s = HibernateUtils.*getSession*();

Query q = s.createQuery(hql);

q.setFirstResult(0);

q.setMaxResults(10);

List<User> users = q.list();

System.***out***.println(users);

s.close();

HibernateUtils.*close*();

}

## 第六节课

### HQL条件查询

HQL：不要写成~~SELECT \* FROM User…~~

* 使用无名称参数条件查询查询单条数据

/\*\*

\* 使用unqiueResult查询一条数据

\*/

@Test

**public** **void** test01() {

Session s = HibernateUtils.*getSession*();

s.beginTransaction();

String hql = "FROM User o1 WHERE o1.id = ?";

Query q = s.createQuery(hql);

q.setParameter(0, 35);

User u = (User) q.uniqueResult();

System.***out***.println(u);

s.getTransaction().commit();

s.close();

HibernateUtils.*close*();

}

* 使用有名称参数进行查询

/\*\*

\* 使用名称参数

\*/

@Test

**public** **void** test03() {

Session s = HibernateUtils.*getSession*();

s.beginTransaction();

String hql = "FROM User o1 WHERE o1.name LIKE :name";

Query q = s.createQuery(hql);

q.setParameter("name", "%2%");

List<User> us = q.list();

System.***out***.println(us);

s.getTransaction().commit();

s.close();

HibernateUtils.*close*();

}

### 本地SQL查询

#### 本地SQL查询列表

/\*\*

\* 使用原生SQL查询

\*/

@Test

**public** **void** test01() {

Session s = HibernateUtils.*getSession*();

s.beginTransaction();

String sql = "SELECT \* FROM t\_d1h02\_user1";

SQLQuery q = s.createSQLQuery(sql);

q.addEntity(User.**class**);

List<User> us = q.list();

System.***out***.println(us);

s.getTransaction().commit();

s.close();

HibernateUtils.*close*();

}

#### 使用SqlQuery进行条件查询

/\*\*

\* 条件查询（名称参数）

\*/

@Test

**public** **void** test02() {

Session s = HibernateUtils.*getSession*();

s.beginTransaction();

String sql = "SELECT \* FROM t\_d1h02\_user1 t1 WHERE t1.name LIKE :name";

SQLQuery q = s.createSQLQuery(sql);

q.addEntity(User.**class**);

q.setParameter("name", "%3%");

List<User> us = q.list();

System.***out***.println(us);

s.getTransaction().commit();

s.close();

HibernateUtils.*close*();

}

### QBC查询

允许以面向对象的方式来执行查询。

* 基本的条件查询：Restrictions.eq/like/lt/gl/le/…( name = 111, name like …)
* 逻辑运行条件：Restrictions.and/or

#### 查询所有数据

/\*\*

\* 使用Criteria查询所有

\*/

@Test

**public** **void** test01() {

Session s = HibernateUtils.*getSession*();

s.beginTransaction();

Criteria c = s.createCriteria(User.**class**);

List<User> us = c.list();

System.***out***.println(us);

s.getTransaction().commit();

s.close();

HibernateUtils.*close*();

}

#### 分页查询

/\*\*

\* 使用Criteria分页查询

\*/

@Test

**public** **void** test02() {

Session s = HibernateUtils.*getSession*();

s.beginTransaction();

Criteria c = s.createCriteria(User.**class**);

c.setFirstResult(0);

c.setMaxResults(5);

List<User> us = c.list();

System.***out***.println(us);

s.getTransaction().commit();

s.close();

HibernateUtils.*close*();

}

* 条件查询

/\*\*

\* 使用Criteria进行多条件查询

\*/

@Test

**public** **void** test03() {

Session s = HibernateUtils.*getSession*();

s.beginTransaction();

Criteria c = s.createCriteria(User.**class**);

c.add(Restrictions.*or*(Restrictions.*eq*("address", "地址33"), Restrictions.*eq*("name", "测试1")));

List<User> us = c.list();

System.***out***.println(us);

s.getTransaction().commit();

s.close();

HibernateUtils.*close*();

}

#### 排序

/\*\*

\* 使用Criteria进行排序

\*/

@Test

**public** **void** test04() {

Session s = HibernateUtils.*getSession*();

s.beginTransaction();

Criteria c = s.createCriteria(User.**class**);

c.addOrder(Order.*desc*("id"));

List<User> us = c.list();

System.***out***.println(us);

s.getTransaction().commit();

s.close();

HibernateUtils.*close*();

}

* 断点调试（执行单步）
* 打log（System.out.println、console.log）
* 排除法（把代码注释掉、逐个去放开注释）

## 作业

* 使用Hibernate完成增删改查功能
  + 编写并使用HibernateUtils
  + 使用C3P0数据源