开发回顾：

SSH框架：

Struts框架， 基于mvc模式的应用层框架技术！

Hibernate, 基于持久层的框架(数据访问层使用)！

Spring, 创建对象处理对象的依赖关系以及框架整合！

Dao代码，如何编写？

-🡪 操作XML数据

-🡪 使用Jdbc技术

🡪原始的jdbc操作， Connection/Statement/ResultSet

🡪 自定义一个持久层框架, 封装了dao的通用方法

🡪 DbUtils组件， 轻量级的dao的组件；

🡪 Hibernate技术 【hibernate最终执行的也是jdbc代码！】

# 一、Hibernate框架

## ORM概念

O, Object 对象

R， Realtion 关系 (关系型数据库: MySQL, Oracle…)

M，Mapping 映射

ORM, 对象关系映射！

ORM, 解决什么问题？

**存储： 能否把对象的数据直接保存到数据库？**

**获取： 能否直接从数据库拿到一个对象？**

**想做到上面2点，必须要有映射！**

**总结：**

**Hibernate与ORM的关系？**

**Hibernate是ORM的实现！**

组件学习：

1. 源码，引入jar文件

**2. 配置**

3. Api

## Hibernate HelloWorld案例

搭建一个Hibernate环境，开发步骤：

1. 下载源码

版本：hibernate-distribution-3.6.0.Final

2. 引入jar文件

**hibernate3.jar核心 + required 必须引入的(6个) + jpa 目录 + 数据库驱动包**

**3. 写对象以及对象的映射**

**Employee.java 对象**

**Employee.hbm.xml 对象的映射 (映射文件)**

4. src/hibernate.cfg.xml 主配置文件

-🡪 数据库连接配置

-🡪 加载所用的映射(\*.hbm.xml)

5. App.java 测试

|  |
| --- |
| Employee.java 对象 |
| //一、 对象  **public** **class** Employee {  **private** **int** empId;  **private** String empName;  **private** Date workDate;    } |
| Employee.hbm.xml 对象的映射 |
| <?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 package=*"cn.itcast.a\_hello"*>  <!-- 对象名称 表名称 -->  <class name=*"Employee"* table=*"employee"*>    <!-- 主键 ，映射 （字段） -->  <id name=*"empId"* column=*"id"*>  <generator class=*"native"*/>  </id>    <!-- 非主键，映射 （属性） -->  <property name=*"empName"* column=*"empName"*></property>  <property name=*"workDate"* column=*"workDate"*></property>    </class>  </hibernate-mapping> |
| hibernate.cfg.xml 主配置文件 |
| <!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>  <!-- 数据库连接配置 -->  <property name=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>  <property name=*"hibernate.connection.url"*>jdbc:mysql:///hib\_demo</property>  <property name=*"hibernate.connection.username"*>root</property>  <property name=*"hibernate.connection.password"*>root</property>  <!—- 数据库方言 –->  <property name=*"hibernate.dialect"*>org.hibernate.dialect.MySQL5Dialect</property>    <property name=*"hibernate.show\_sql"*>true</property>    <!-- 加载所有映射 -->  <mapping resource=*"cn/itcast/a\_hello/Employee.hbm.xml"*/>  </session-factory>  </hibernate-configuration> |
| App.java 测试类 |
| **public** **class** App {  @Test  **public** **void** testHello() **throws** Exception {  // 对象  Employee emp = **new** Employee();  emp.setEmpName("班长");  emp.setWorkDate(**new** Date());    // 获取加载配置文件的管理类对象  Configuration config = **new** Configuration();  config.configure(); // 默认加载src/hibenrate.cfg.xml文件  // 创建session的工厂对象  SessionFactory sf = config.buildSessionFactory();  // 创建session (代表一个会话，与数据库连接的会话)  Session session = sf.openSession();  // 开启事务  Transaction tx = session.beginTransaction();  //保存-数据库  session.save(emp);  // 提交事务  tx.commit();  // 关闭  session.close();  sf.close();  }  } |

## Hibernate Api

|-- Configuratio配置管理类对象（读取配置文件以及根据读取配置的文件创建SessionFactory）

config.configure(); 加载主配置文件的方法(hibernate.cfg.xml)

默认加载src/hibernate.cfg.xml

config.configure(“cn/config/hibernate.cfg.xml”); 加载指定路径下指定名称的主配置文件

config.buildSessionFactory(); 创建session的工厂对象

|-- SessionFactory session的工厂（或者说代表了这个hibernate.cfg.xml配置文件）

sf.openSession(); 创建一个sesison对象

sf.getCurrentSession(); 创建session或取出session对象

|--**Session** session对象维护了一个连接(Connection), 代表了与数据库连接的会话。

Hibernate最重要的对象： 只用使用hibernate与数据库操作，都用到这个对象

session.beginTransaction(); 开启一个事务； hibernate要求所有的与数据库的操作必须有事务的环境，否则报错！

更新：

session.save(obj); 保存一个对象

session.update(emp); 更新一个对象

session.saveOrUpdate(emp); 保存或者更新的方法：

🡪没有设置主键，执行保存；

🡪有设置主键，执行更新操作;

🡪如果设置主键不存在报错！

“”-----四种查询方式--------------------------------------------------------------------------------------

主键查询：

session.get(Employee.class, 1); 主键查询

session.load(Employee.class, 1); 主键查询 (支持懒加载)

HQL查询：

HQL查询与SQL查询区别：

**SQL**: (结构化查询语句)查询的是表以及字段; 不区分大小写。

**HQL**: hibernate query language 即hibernate提供的面向对象的查询语言

查询的是对象以及对象的属性。

区分大小写。

Criteria查询：(kuiti erli)

完全面向对象的查询。

本地SQL查询：

复杂的查询，就要使用原生态的sql查询，也可以，就是本地sql查询的支持！

(缺点： 不能跨数据库平台！)

|  |
| --- |
| **package** cn.itcast.a\_hello;  **import** java.util.List;  **import** org.hibernate.Criteria;  **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.hibernate.criterion.Restrictions;  **import** org.junit.Test;  **public** **class** App3 {    **private** **static** SessionFactory *sf*;  **static** {    // 创建sf对象  *sf* = **new** Configuration().configure().buildSessionFactory();  }  //HQL查询 【适合有数据库基础的】  @Test  **public** **void** testQuery() **throws** Exception {    Session session = *sf*.openSession();  Transaction tx = session.beginTransaction();    // 主键查询  //Employee emp = (Employee) session.get(Employee.class, 1);    // HQL查询，查询全部  Query q = session.createQuery("from Employee where empId=1 or empId=2");  List<Employee> list = q.list();    System.***out***.println(list);    tx.commit();  session.close();  }    //QBC查询 , query by criteria 完全面向对象的查询  @Test  **public** **void** testQBC() **throws** Exception {  Session session = *sf*.openSession();  Transaction tx = session.beginTransaction();    Criteria criteria = session.createCriteria(Employee.**class**);  // 条件  criteria.add(Restrictions.*eq*("empId", 1));  // 查询全部  List<Employee> list = criteria.list();    System.***out***.println(list);    tx.commit();  session.close();  }    //sQL  @Test  **public** **void** testSQL() **throws** Exception {  Session session = *sf*.openSession();  Transaction tx = session.beginTransaction();    // 把每一行记录封装为对象数组，再添加到list集合  // SQLQuery sqlQuery = session.createSQLQuery("select \* from employee");  // 把每一行记录封装为 指定的对象类型  SQLQuery sqlQuery = session.createSQLQuery("select \* from employee").addEntity(Employee.**class**);  List list = sqlQuery.list();    System.***out***.println(list);    tx.commit();  session.close();  }  } |

|-- Transaction hibernate事务对象

共性问题1：

ClassNotFoundException…., 缺少jar文件！

问题2：

如果程序执行程序，hibernate也有生成sql语句，但数据没有结果影响。

问题一般是事务忘记提交…….

遇到问题，一定看错误提示!

## Hibernate crud (增删改查)

|  |
| --- |
| **public** **class** EmployeeDaoImpl **implements** IEmployeeDao{  @Override  **public** Employee findById(Serializable id) {  Session session = **null**;  Transaction tx = **null**;  **try** {  // 获取Session  session = HibernateUtils.*getSession*();  // 开启事务  tx = session.beginTransaction();  // 主键查询  **return** (Employee) session.get(Employee.**class**, id);  } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally** {  tx.commit();  session.close();  }  }  @Override  **public** List<Employee> getAll() {  Session session = **null**;  Transaction tx = **null**;  **try** {  session = HibernateUtils.*getSession*();  tx = session.beginTransaction();  // HQL查询  Query q = session.createQuery("from Employee");  **return** q.list();  } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally** {  tx.commit();  session.close();  }  }  @Override  **public** List<Employee> getAll(String employeeName) {  Session session = **null**;  Transaction tx = **null**;  **try** {  session = HibernateUtils.*getSession*();  tx = session.beginTransaction();  Query q =session.createQuery("from Employee where empName=?");  // 注意：参数索引从0开始  q.setParameter(0, employeeName);  // 执行查询  **return** q.list();  } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally** {  tx.commit();  session.close();  }  }  //分页查询  @Override  **public** List<Employee> getAll(**int** index, **int** count) {  Session session = **null**;  Transaction tx = **null**;  **try** {  session = HibernateUtils.*getSession*();  tx = session.beginTransaction();  Query q = session.createQuery("from Employee");  // 设置分页参数  q.setFirstResult(index); // 查询的其实行  q.setMaxResults(count); // 查询返回的行数    List<Employee> list = q.list();  **return** list;  } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally** {  tx.commit();  session.close();  }  }  @Override  **public** **void** save(Employee emp) {  Session session = **null**;  Transaction tx = **null**;  **try** {  session = HibernateUtils.*getSession*();  tx = session.beginTransaction();  // 执行保存操作  session.save(emp);  } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally** {  tx.commit();  session.close();  }    }  @Override  **public** **void** update(Employee emp) {  Session session = **null**;  Transaction tx = **null**;  **try** {  session = HibernateUtils.*getSession*();  tx = session.beginTransaction();  session.update(emp);    } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally** {  tx.commit();  session.close();  }    }  @Override  **public** **void** delete(Serializable id) {  Session session = **null**;  Transaction tx = **null**;  **try** {  session = HibernateUtils.*getSession*();  tx = session.beginTransaction();  // 先根据id查询对象，再判断删除  Object obj = session.get(Employee.**class**, id);  **if** (obj != **null**) {  session.delete(obj);  }  } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally** {  tx.commit();  session.close();  }  }  } |

# 二、Hibernate.cfg.xml 主配置

Hibernate.cfg.xml

主配置文件中主要配置：数据库连接信息、其他参数、映射信息！

|  |
| --- |
| <!-- 1. 数据库连接配置 connection的前缀可写可不写-->  <propertyname=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>  <property name=*"hibernate.connection.url"*>jdbc:mysql:///hib\_demo</property>  <property name=*"hibernate.connection.username"*>root</property>  <property name=*"hibernate.connection.password"*>root</property>  <!-- 数据库方法配置hibernate在运行的时候，会根据不同的方言生成符合当前数据库语法的sql-->  <property name=*"hibernate.dialect"*>org.hibernate.dialect.MySQL5Dialect</property> |

常用配置查看源码：

hibernate-distribution-3.6.0.Final\project\etc**\hibernate.properties**

|  |
| --- |
| <!-- 2. 其他相关配置 -->  <!-- 2.1 显示hibernate在运行时候执行的sql语句 -->  <property name=*"hibernate.show\_sql"*>true</property>  <!-- 2.2 格式化sql -->  <property name=*"hibernate.format\_sql"*>true</property> |

## 数据库连接参数配置

**例如：**

## MySQL

#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

## 自动建表

Hibernate.properties

|  |
| --- |
| <!-- 2.3 自动建表 -->  <!--每次在创建sessionFactory时候执行创建表；  当调用sesisonFactory的close方法的时候，删除表！ -->  <property name=*"hibernate.hbm2ddl.auto"*>create-drop</property>  <!-- 每次都重新建表； 如果表已经存在就先删除再创建 -->  <property name=*"hibernate.hbm2ddl.auto"*>create</property>  <!-- 如果表不存在就创建； 表存在就不创建 --> 常用  <property name=*"hibernate.hbm2ddl.auto"*>update</property>  <!-- (生成环境时候) 执行验证： 当映射文件的内容与数据库表结构不一样的时候就报错！ -->  <property name=*"hibernate.hbm2ddl.auto"*>validate</property> |

通常：测试中常用 但是在安全要求较高就不能有

代码自动建表：

|  |
| --- |
| **public** **class** App\_ddl {  // 自动建表  @Test  **public** **void** testCreate() **throws** Exception {  // 创建配置管理类对象  Configuration config = **new** Configuration();  // 加载主配置文件  config.configure();    // 创建工具类对象  SchemaExport export = **new** SchemaExport(config);  // 建表  // 第一个参数： 是否在控制台打印建表语句  // 第二个参数： 是否执行脚本  export.create(**true**, **true**);  }  } |

# 三、映射配置

XXX.hbm.xml

1. 普通字段类型

2. 主键映射

单列主键映射

多列作为主键映射

主键生成策略，查看api： 5.1.2.2.1. Various additional generators

数据库：

一个表能否有多个主键？ 不能。

为什么要设置主键？ 数据库存储的数据都是有效的，必须保持唯一。

(为什么把id作为主键？)

因为表中通常找不到合适的列作为唯一列即主键，所以为了方法用id列，因为id是数据库系统维护可以保证唯一，所以就把这列作为主键!

联合/复合主键

如果找不到合适的列作为主键，出来用id列以外，我们一般用联合主键，即多列的值作为一个主键，从而确保记录的唯一性。

## 映射配置

|  |
| --- |
| <?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">  <!-- 映射文件: 映射一个实体类对象； 描述一个对象最终实现可以直接保存对象数据到数据库中。 -->  <!--  package: 要映射的对象所在的包(可选,如果不指定,此文件所有的类都要指定全路径)  auto-import 默认为true， 在写hql的时候自动导入包名  如果指定为false, 再写hql的时候必须要写上类的全名；  如：session.createQuery("fromcn.itcast.c\_hbm\_config.Employee").list();  -->  <hibernate-mapping package=*"cn.itcast.c\_hbm\_config"* auto-import=*"true"*>    <!--  class 映射某一个对象的(一般情况，一个对象写一个映射文件，即一个class节点)  name 指定要映射的对象的类型  table 指定对象对应的表；  如果没有指定表名，默认与对象名称一样  -->  <class name=*"Employee"* table=*"employee"*>    <!-- 主键 ，映射-->  <id name=*"empId"* column=*"id"*>  <!--  主键的生成策略  identity 自增长(mysql,db2)  sequence 自增长(序列)， oracle中自增长是以序列方法实现  native 自增长【会根据底层数据库自增长的方式选择identity或sequence】  如果是mysql数据库, 采用的自增长方式是identity  如果是oracle数据库， 使用sequence序列的方式实现自增长    increment 自增长(会有并发访问的问题，一般在服务器集群环境使用会存在问题。)    assigned 指定主键生成策略为手动指定主键的值  uuid 指定uuid随机生成的唯一的值  foreign (外键的方式， one-to-one讲)  -->  <generator class=*"uuid"*/>  </id>    <!--  普通字段映射  property  name 指定对象的属性名称  column 指定对象属性对应的表的字段名称，如果不写默认与对象属性一致。  length 指定字符的长度, 默认为255  type 指定映射表的字段的类型，如果不指定会匹配属性的类型  java类型： 必须写全名  hibernate类型： 直接写类型，都是小写  -->  <property name=*"empName"* column=*"empName"* type=*"java.lang.String"* length=*"20"*></property>  <property name=*"workDate"* type=*"java.util.Date"*></property>  <!-- 如果列名称为数据库关键字，需要用反引号或改列名。 -->  <property name=*"desc"* column=*"`desc`"* type=*"java.lang.String"*></property>    </class>    </hibernate-mapping> |

## 复合主键映射

|  |
| --- |
| // 复合主键类  **public** **class** CompositeKeys **implements** Serializable{  **private** String userName;  **private** String address;  // .. get/set  } |
| **public** **class** User {  // 名字跟地址，不会重复  **private** CompositeKeys keys;  **private** **int** age;  } |
| User.hbm.xml |
| <?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 package=*"cn.itcast.d\_compositeKey"* auto-import=*"true"*>    <class name=*"User"*>    <!-- 复合主键映射 -->  <composite-id name=*"keys"*>  <key-property name=*"userName"* type=*"string"*></key-property>  <key-property name=*"address"* type=*"string"*></key-property>  </composite-id>    <property name=*"age"* type=*"int"*></property>    </class>    </hibernate-mapping> |
| App.java |
| **public** **class** App2 {    **private** **static** SessionFactory *sf*;  **static** {  // 创建sf对象  *sf* = **new** Configuration()  .configure()  .addClass(User.**class**) //（测试） 会自动加载映射文件：Employee.hbm.xml  .buildSessionFactory();  }  //1. 保存对象  @Test  **public** **void** testSave() **throws** Exception {  Session session = *sf*.openSession();  Transaction tx = session.beginTransaction();    // 对象  CompositeKeys keys = **new** CompositeKeys();  keys.setAddress("广州棠东");  keys.setUserName("Jack");  User user = **new** User();  user.setAge(20);  user.setKeys(keys);    // 保存  session.save(user);      tx.commit();  session.close();  }    @Test  **public** **void** testGet() **throws** Exception {  Session session = *sf*.openSession();  Transaction tx = session.beginTransaction();    //构建主键再查询  CompositeKeys keys = **new** CompositeKeys();  keys.setAddress("广州棠东");  keys.setUserName("Jack");    // 主键查询  User user = (User) session.get(User.**class**, keys);  // 测试输出  **if** (user != **null**){  System.*out*.println(user.getKeys().getUserName());  System.*out*.println(user.getKeys().getAddress());  System.*out*.println(user.getAge());  }      tx.commit();  session.close();  }  } |

练习：

1. 搭建hibernate开发环境

2. Api

Session类的相关方法

3. 主配置文件/映射配置/主键映射