Struts2：

1. 简化Servlet开发，用Action取代之前的Servlet

2. 开发流程

3. 请求数据自动封装 (params拦截器)

3. struts 数据处理

Map---》域对象

4. 文件上传/下载

5. 数据效验、国际化

6. Ognl表达式语言、Struts标签

(OgnlContext/ValueStack)

传统的开发模式：

基于MVC模式项目体系结构！

# 1. Hibernate框架

## ORM 规则

ORM, Object Relation Mapping 对象关系映射!

目标：

通过orm（对象关系映射）, 可以做到：

a. 不写sql，包对象直接保存到数据库中

b. 不写sql， 从数据库中读取，直接获取的是封装好的对象！

ORM是一种思想，或者说是一种规则的定义， 基于orm的应用：

1. hibernate 框架

2. mybatis 框架

3. 自定义一套程序！

## Hibernate框架

### 概念

基于ORM的持久层框架，对jdbc操作进行了封装！

Hibernate与具体数据库无关的技术，即可以跨数据库平台！

### Hibernate开发环境搭建

Hibernate版本： 3.6版本

整体步骤：

1. 下载源码，引入jar文件

2. 写配置

\* hibernate.cfg.xml 主配置

\* Users.hbm.xml 映射配置

3. Api

Configuration

SessionFactory

Transaction

Session

Query

步骤实现：

|  |
| --- |
| 1. 建库、建表 |
| -- 删除数据库  DROP DATABASE hib\_demo;  -- 创建数据库  CREATE DATABASE hib\_demo DEFAULT CHARACTER SET utf8;  -- 建表  CREATE TABLE users(  id INT PRIMARY KEY AUTO\_INCREMENT,  NAME VARCHAR(20),  age INT  ) |
| 2. 引入hibernate核心jar文件 |
| 源码中：  hibernate3.jar + required 目录中所有 + jpa + 数据库驱动包  antlr-2.7.6.jar  commons-collections-3.1.jar  dom4j-1.6.1.jar  hibernate-jpa-2.0-api-1.0.0.Final.jar  hibernate3.jar  javassist-3.12.0.GA.jar  jta-1.1.jar  mysql-connector-java-5.1.12-bin.jar  slf4j-api-1.6.1.jar |
|  |
| 3. 主配置文件: hibernate.cfg.xml |
| <!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>  <!-- 数据库连接的参数配置 -->  <property name=*"hibernate.connection.url"*>jdbc:mysql:///hib\_demo</property>  <property name=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>  <property name=*"hibernate.connection.username"*>root</property>  <property name=*"hibernate.connection.password"*>root</property>  <!-- 通过数据库方言，告诉hibernate如何生产sql。 hibernate会根据配置的方言，生产符合当前数据库语言的SQL语句 -->  <property name=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</property>  </session-factory>  </hibernate-configuration> |
| 4. javabean/ 映射文件 |
| <?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">  <!--  通过XML映射，把对象映射到数据库的表中！  -->  <hibernate-mapping package=*"cn.itcast.a\_config"*>    <class name=*"Users"* table=*"users"*>  <id name=*"userId"* column=*"id"*>  <generator class=*"native"*></generator>  </id>  <property name=*"userName"* column=*"name"*></property>  <property name=*"age"* column=*"age"*></property>  </class>  </hibernate-mapping> |
|  |
| 5. App |
| // 测试类  **public** **class** App {  @Test  // 保存  **public** **void** testSave() **throws** Exception {  // 对象  Users users = **new** Users();  users.setUserName("Jack");  users.setAge(30);  // 1. 创建配置管理器对象  Configuration config = **new** Configuration();  // 2. 加载主配置文件， 默认加载src/hibernate.cfg.xml  config.configure();  // 3. 根据加载的主配置文件，创建对象  SessionFactory sf = config.buildSessionFactory();  // 4. 创建Session对象  Session session = sf.openSession();  // 5. 开启事务  Transaction tx = session.beginTransaction();  // --- 保存  session.save(users);  // 6. 提交事务/关闭session  tx.commit();  session.close();  }  @Test  // 获取对象  **public** **void** testGet() **throws** Exception {  // 1. 创建配置管理器对象  Configuration config = **new** Configuration();  // 2. 加载主配置文件， 默认加载src/hibernate.cfg.xml  config.configure();  // 3. 根据加载的主配置文件，创建对象  SessionFactory sf = config.buildSessionFactory();  // 4. 创建Session对象  Session session = sf.openSession();  // 5. 开启事务  Transaction tx = session.beginTransaction();  // --- 获取 (根据主键查询)  Users users = (Users) session.get(Users.**class**, 1);    System.*out*.println(users);  // 6. 提交事务/关闭session  tx.commit();  session.close();  }  } |

# 2. Hibernate框架Api

**|-- Session**  一个与数据库连接的会话信息

Sesison里面维护了一个连接对象，且对常用操作进行封装！

更新：

session.save(obj); 保存一个对象

session.update(obj) 更新一个对象, 注意修改对象的主键一定要在数据库存在!

session.saveOrUpdate(obj) 保存或更新

如果有设置主键且主键存在执行更新！

没有设置主键执行保存！

session.delete(obj) 删除一个对象; 注意修改对象的主键一定要在数据库存在!

主键查询：

Session.get(clazz,id); 根据主键查询

Session.load(clazz,id); 根据主键查询

HQL查询：

Hibernate Query language

Hibernate提供的面向对象的查询语言！

查询的是对象、对象的属性！

HQL与SQL区别：

Hql 查询的是对象、对象的属性， 区分大小写！

Hql查询的对象一定要有映射！

SQL 查询的是表、字段，不区分大小写！

# 3. Hibernate执行流程

# 4. hibernate.cfg.xml 配置详解

在主配置文件配置的时候，hibernate 前缀可以省略！

d

查看配置提示：

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

里面有常用配置，如：

#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.hbm2ddl.auto create-drop 每次在创建sessionFactory的时候创建

表；执行sf.close()删除表。

#hibernate.hbm2ddl.auto create 每次都先删除表，再创建新的表

#hibernate.hbm2ddl.auto update 如果表不存在则创建，存在就不创建！

#hibernate.hbm2ddl.auto validate 检查映射配置与数据库结构是否一致，不一致就报错! (严格)

也可以通过代码的方式，自动建表：

|  |
| --- |
| /\*\*  \* 代码方式自动建表  \* **@param** args  \*/  **public** **static** **void** main(String[] args) {  // 创建配置管理器对象，加载主配置文件(会加载映射)  Configuration cfg = **new** Configuration().configure();  // 自动建表工具类  SchemaExport export = **new** SchemaExport(cfg);  // 创建表  // 第一个参数： 是否打印建表语句到控制台  // 第二个参数： 是否执行脚本，生成表  export.create(**true**, **true**);  } |

# 5. 映射配置

映射文件,

命名: \*.hbm.xml

作用： 描述“对象”与“表”的映射关系！

通过映射文件可以描述一张完整的表！

|  |
| --- |
| <?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">  <!--  通过XML映射，把对象映射到数据库的表中！    package 表示包名； 可选，如果没有写，后面的类必须指定类的全名！  auto-import="true" 默认为true，即在写hql的时候，会自动引入包名；  如为false，hql中对象要写上包名称  Query q = session.createQuery("from Users");  -->  <hibernate-mapping package=*"cn.itcast.b\_api"* auto-import=*"true"*>  <!--  class 表示映射的一个javabean对象  (可以有多个class节点，但一般一个映射文件对应一个class)    name 表示映射的类的名称;  table (可选)类的名称，对应的表名, 如果不写默认与类名称一样    -->  <class name=*"Users"* table=*"t\_users"*>  <!--    id 表示映射的是主键  注意： 在hibernate的映射文件中，必须要有主键映射!  那就要求表必须要有主键！    generator 表示的是主键生成策略 (Api : 5.1.2.2.1. Various additional generators)  class    identity 主键自增长， mysql/sqlservlet等数据库使用的自增长的方式  sequence 以序列的方式实现自增长；  native 表示主键自增长: 根据底层数据库的能力选择 identity、sequence等其中一个。    assigned 手动指定主键的值  uuid uuid的值作为主键，确保一定唯一  -->  <id name=*"userId"* column=*"id"*>  <generator class=*"native"*></generator>  </id>      <!--  property 表示普通字段的映射  name 属性的名称  column 可选，如果没有指定，与属性名称一致  length 表示字符长度，只对字符类型有效  type 数据库中字段的类型 （如果没有指定，会匹配属性的类型）  hibernate中指定的类型: 小写开头  java中类型: 写类的全名  unique 设置为true，表示给当前列添加唯一约束  (主键约束 = 唯一约束 + 非空)  -->  <property name=*"userName"* type=*"java.lang.String"* column=*"username"* length=*"50"* unique=*"true"*></property>    <property name=*"age"* column=*"age"* type=*"int"*></property>    <!-- 注意：如果列名称对应的是数据库的关键字，需要处理 -->  <property name=*"desc"* column=*"`desc`"* length=*"200"*></property>  </class>    </hibernate-mapping> |

# 6. 联合主键映射

联合主键，

多列共同作为主键！

|  |
| --- |
| 数据库，联合主键： |
| -- 联合主键 (hbm: <composite-id></composite-id> )  CREATE TABLE test(  id1 INT,  id2 INT,  **PRIMARY KEY(id1,id2)**  );  SELECT \* FROM test;  -- 插入  INSERT INTO test VALUES(1,1);  INSERT INTO test VALUES(1,2);  INSERT INTO test VALUES(1,2); -- 错误 |

需求：

保存员工信息: 名称、地址、部门、年龄

(联合主键：名称、地址)

设计javabean：

写映射：

App:

|  |
| --- |
| /\*\*  \* 联合主键对象, 必须要实现可课序列化标记！  \*/  **public** **class** CompositeKeys **implements** Serializable{  **private** String name;  **private** String address;  ..  } |
| // 员工  **public** **class** Employee {  // 名称、地址、部门、年龄    // 联合主键对象  **private** CompositeKeys keys;  **private** String dept;  **private** Date birth;  ..  } |
| Employee的映射： |
| <?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 package=*"cn.itcast.c\_composite"*>  <class name=*"Employee"* table=*"t\_employee"*>  <composite-id name=*"keys"*>  <key-property name=*"name"*></key-property>  <key-property name=*"address"*></key-property>  </composite-id>    <property name=*"dept"* length=*"50"*></property>  <property name=*"birth"* type=*"date"*></property>  </class>    </hibernate-mapping> |
|  |
| 测试 |
| **public** **class** App {  **private** **static** SessionFactory *sf*;  **static** {  *sf* = **new** Configuration().configure().buildSessionFactory();  }  @Test  **public** **void** testSave() **throws** Exception {  Session session = *sf*.openSession();  session.beginTransaction();    // 联合主键对象  CompositeKeys keys = **new** CompositeKeys();  keys.setName("Jack");  keys.setAddress("棠东...");    // 对象  Employee emp = **new** Employee();  emp.setKeys(keys);  emp.setDept("研发部");  emp.setBirth(**new** Date());    // 保存  session.save(emp);      session.getTransaction().commit();  session.close();  }    @Test  **public** **void** testGet() **throws** Exception {  Session session = *sf*.openSession();  session.beginTransaction();    // 联合主键对象  CompositeKeys keys = **new** CompositeKeys();  keys.setName("Jack");  keys.setAddress("棠东...");    Employee employee = (Employee) session.get(Employee.**class**, keys);  System.*out*.println(employee.getKeys().getName());  System.*out*.println(employee.getKeys().getAddress());  System.*out*.println(employee.getDept());  System.*out*.println(employee.getBirth());    session.getTransaction().commit();  session.close();  }  } |