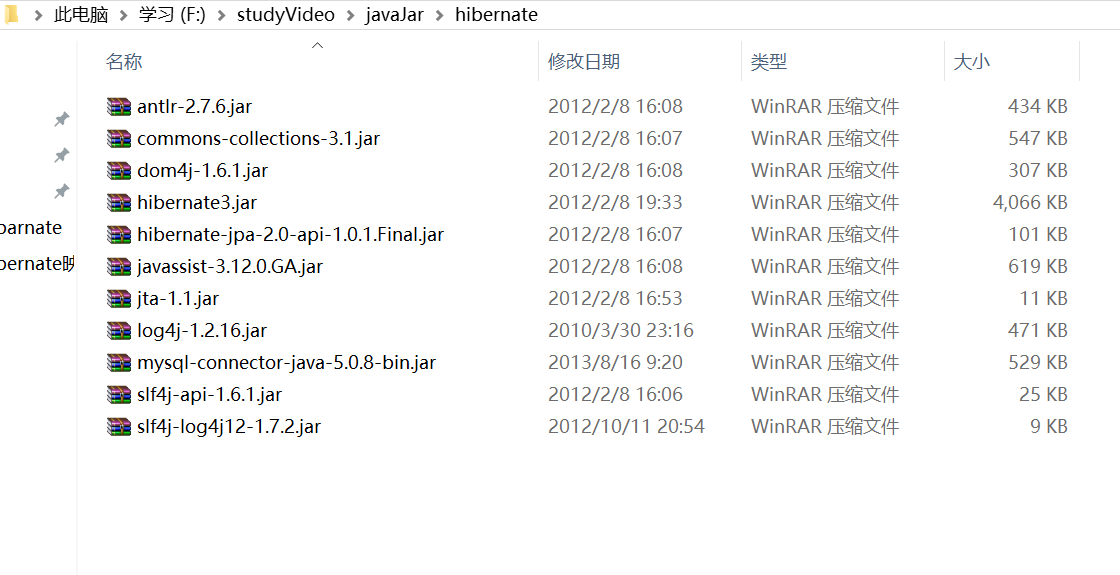
一．Hibernate是什么？

Hibernate是javaweb项目dao层的一个框架，主要是为了方便dao层的操作。通过ORM（Object Relational Mapping）完成java对象和 数据库表的映射。

通过这个框架，可以很简明的 把对象和数据库的表 对应起来。

二．Hibernate开发环境配置：

1.引入jar包：



2.编写核心配置文件

在src目录下编写：hibernate.cfg.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"***>***<**hibernate-configuration**>  
 <!--会话连接工厂，建立数据哭连接-->  
 <**session-factory**>  
 <!--必须的参数：连接数据库-->  
 <**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">0807</**property**>  
 <!--因为每个数据库sql语句稍微有差别，所以要配置这样来明确使用哪个数据库语言-->  
 <**property** name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</**property**>  
  
 <!--控制sql语句-->  
 <**property** name="hibernate.show\_sql">true</**property**>  
 <**property** name="hibernate.format\_sql">true</**property**>  
  
 <**property** name="hbm2ddl.auto">update</**property**>

通过hbm2ddl.auto 的参数来设置：

\* create :每次执行的时候,创建一个新的表.(如果以前有该表,将该表删除重新创建.) 一般测试的时候的使用.

\* create-drop :每次执行的时候,创建一个新的表,程序执行结束后将这个表,删除掉了. 一般测试的时候使用.

\* update :如果数据库中没有表,创建一个新的表,如果有了,直接使用这个表.可以更新表的结构.

\* validate :会使用原有的表.完成校验.校验映射文件与表中配置的字段是否一致.不一致报错.  
  
 <!--引入orm映射文件-->  
 <**mapping** resource="bean/User.hbm.xml"></**mapping**>  
 <**mapping** resource="bean/Order.hbm.xml"></**mapping**>  
 <**mapping** resource="bean/Customer.hbm.xml"></**mapping**>  
 </**session-factory**>  
</**hibernate-configuration**>

3.映射文件的配置

*<?*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="bean.Order" table="orders"> <!--如果table="order" 就会出现错误，原因是，order不能作为表名，因为order是数据库关键字-->  
 <!--主键属性：id-->  
 <**id** name="oid" column="oid">  
 <**generator** class="native"></**generator**>  
 </**id**>  
 <**property** name="addr" column="addr"></**property**>  
 <!--建立实体关系对应-->  
 <**many-to-one** name="customer" column="cid" class="bean.Customer" />  
 <!--  
 cascade="save-update" 这个级联属性，是有方向的，因为在程序中，关联也是有方向的，比如：  
  
 customer1.getOrders().add(order1);  
 session.save(customer1); 这必须在 one 中的set标签配置级联属性。 表示customer->order 这样一起保存。  
  
 order.setCustomer(customer1);  
 session.save(order); 表示order->customer 这个方向，因此必须在order这个多中，配置cascade ，且保存必须保存order。  
 -->  
</**class**>  
</**hibernate-mapping**>

映射文件配置三个关键点：

1.主键要用id标签来配置，并且可以通过generator确定生成方法

2.property中name等于实体属性名 column等于数据库表名。

3.关于多表关联的：

1）一对多：实体类中 一 设置一个set集合，配置如下：

<!--配置映射-->  
 <**set** name="orders" cascade="save-update" >  
 <**key column**="cid"></**key**>  
 <**one-to-many** class="bean.Order" ></**one-to-many**>  
 </**set**>

这里：name等于 one 实体中set集合名字 cascade：设置级联

Key 中column 表示 many 实体中那个外键的名字。

one-to-many 中class表示 many的实体。

实体类中 多 设置一个 一 的对象，表示它属于哪个。

配置如下：

<many-to-one name="customer" column="cid" class="bean.Customer" />

Colum表示添加的外键，name表示many实体中对应的对象名。Class表示具体的类。

2）多对多：就是两个实体的映射配置文件中，都要加set集合：

<!-- <set>中name:对应当前类中的学生的集合的名称 table:中间表的名称-->

<set name="students" table="stu\_cour">

<!-- <key>中column:当前类在中间表中外键 -->

<key column="cno"></key>

<!-- <many-to-many>中class:另一方的类全路径. column:另一方在中间表中外键名称 -->

<many-to-many class="cn.itcast.hibernate3.demo3.Student" column="sno"/>

</set>

三．Hibernate的核心api：

1.核心配置文件的加载api以及获得session

**public class SessionUtils** {  
  
 **private static Configuration** *configuration*;  
 **private static** SessionFactory *sessionFactory*;  
  
 **static**{  
 //通过Configuration来加载核心配置文件以及映射配置文件。  
 *configuration* = **new** Configuration().configure();  
 //通过Configuration来获取得到Session的工厂。  
 *sessionFactory* = *configuration*.buildSessionFactory();  
 }  
  
 **public static** Session openSession(){  
 //通过工厂直接获取到session ，session像是JDBC中的Connection对象。代表一个连接对象。  
 **return** *sessionFactory*.openSession();  
 }  
  
 **public static void** main(**String**[] args) {  
 openSession();  
 }  
}

2.session相关：

Session session= **SessionUtils**.openSession();  
 //2.获取事务  
 Transaction transaction=session.beginTransaction();  
 //3.执行业务逻辑。

主要业务代码也在这里

//4.事务提交  
 transaction.commit();

//5.关闭资源  
 session.close();

Session中对于实体对象的操作：

save()/persist() :添加.

update( ) :修改

saveOrUpdate() :增加和修改对象

delete() :删除对象

get()/load() :根据主键查询

createQuery() :创建一个Query接口,编写HQL语句

createSQLQuery() :创建一个SQLQuery接口,编写SQL语句数据库操作对象

createCriteria() :返回一个Criteria接口.条件查询

四．Hibernate的对象状态以及一级缓存

4.1对象的三种状态：

\* 将持久化对象分成三种状态:

\* 瞬时态:

\* 没有唯一标识OID,且没有与session关联.

\* 持久态:

\* 有唯一的标识OID,且与session关联.

\* 脱管态:

\* 有唯一的标识OID,没有与session关联.

\* 瞬时态对象:

\* Book book = new Book();

\* 瞬时-->持久:

\* save()/saveOrUpdate();

\* 瞬时-->脱管:

\* book.setBid(1);

\* 持久态对象:

\* get()/load()/find()/lock();

\* 持久态-->瞬时:

\* delete();

\* 持久态-->脱管:

\* close()/clear()/evict();

\* 脱管态对象:

\* Book book = new Book();

\* book.setBid(1);

\* 脱管-->瞬时:

\* book.setBid(null);

\* 脱管-->持久:

\* update();/saveOrUpdate();

4.2 hibernate的一级缓存

Session中内置了很多集合，以使得在从数据库中获取数据的时候，同时把对象存入相应的集合，这样的话就可以优化hibernate，这就是一级缓存，与session的生命周期一致。

同时，session中存在一个快照区：

向一级缓存存入数据的时候,（）放入一级缓存区和一级缓存快照区,

Customer customer=session.get(Customer.class,1);

当更新了一级缓存的数据的时候,事务一旦提交,比对一级缓存和快照区,如果数据一致,不更新,如果数据不一致,自动更新数据库.

customer.setName(“xxx”); 改变了的话就自动更新，没有就不更新。

这就是一级缓存可以自动更新的原理。

五．Hibernate的检索方式：

检索方式:查询的方式:

**导航对象图检索方式: 根据已经加载的对象导航到其他对象**

**\* Customer customer = (Customer)session.get(Customer.class,1);**

**\* customer.getOrders();// 获得到客户的订单**

**OID 检索方式: 按照对象的 OID 来检索对象**

\* get()/load();方法进行检索.

**HQL 检索方式: 使用面向对象的 HQL 查询语言**

**\* Query query = session.createQuery(“HQL”);**

**QBC 检索方式: 使用 QBC(Query By Criteria) API 来检索对象. 这种 API 封装了基于字符串形式的查询语句, 提供了更加面向对象的查询接口.**

**\* Criteria criteria = session.createCriteria(Customer.class);**

**本地 SQL 检索方式: 使用本地数据库的 SQL 查询语句**

**\* SQLQuery query = session.createSQLQuery(“SQL”);**

### HQL:

HQL:Hibernate Query Language:

\* 特点:

\* 面向对象的查询:

\* 支持方法链编程:

\* 使用:

1. 创建Query接口.
2. 查询所有记录:

List<Customer> list = session.createQuery("from Customer").list();

for (Customer customer : list) {

System.out.println(customer);

}

1. 查询使用别名:

// 使用别名

// 别名as可以省略

/\* List<Customer> list =

session.createQuery("from Customer c").list();

System.out.println(list);\*/

// 使用别名:带参数

/\*List<Customer> list = session

.createQuery("from Customer as c where c.cname = ?")

.setString(0, "小沈").list();

System.out.println(list);\*/

// 不支持 select \* from Customer写法.可以写成 select 别名 from Customer as 别名;

List<Customer> list = session.createQuery("select c from Customer c").list();

System.out.println(list);

1. 排序:

List<Customer> list = session.createQuery(

"from Customer c order by c.id desc").list();

for (Customer customer : list) {

System.out.println(customer);

}

1. 分页查询:

Query query = session.createQuery("from Order");

query.setFirstResult(20);

query.setMaxResults(10);

List<Order> list = query.list();

for (Order order : list) {

System.out.println(order);

}

1. 单个对象查询:

Customer customer = (Customer) session

.createQuery("from Customer where cname = ?")

.setString(0, "小明").uniqueResult();

System.out.println(customer);

1. 参数绑定:

// 1.使用?号方式绑定

/\*Query query = session.createQuery("from Customer where cname = ?");

query.setString(0, "小沈");

List<Customer> list = query.list();

System.out.println(list);\*/

/\*Query query = session.createQuery("from Customer where cname = ? and cid =?");

query.setString(0, "小沈");

query.setInteger(1,3);

List<Customer> list = query.list();

System.out.println(list);\*/

// 2.使用名称的方式绑定

Query query = session.createQuery("from Customer where cname=:name and cid=:id");

query.setString("name", "小沈");

query.setInteger("id", 3);

List<Customer> list = query.list();

System.out.println(list);

// 3.绑定实体

List<Order> list = session

.createQuery("from Order o where o.customer = ?")

.setEntity(0, customer).list();

for (Order order : list) {

System.out.println(order);

}

1. 投影操作:

// 查询客户的名称:

/\*

\* List<Object> list = session.createQuery(

\* "select c.cname from Customer c").list(); System.out.println(list);

\*/

/\*

\* List<Object[]> list = session.createQuery(

\* "select c.cid,c.cname from Customer c").list(); for (Object[] objects

\* : list) { System.out.println(Arrays.toString(objects)); }

\*/

List<Customer> list = session.createQuery(

"select new Customer(cname) from Customer").list();

System.out.println(list);

1. 模糊查询:

Query query = session.createQuery("from Customer where cname like ?");

query.setParameter(0, "小%");

List<Customer> list = query.list();

System.out.println(list);

SQL多表查询:

\* 连接:

\* 交叉连接:

\* select \* from A,B;

\* 内连接:查询的是两个表的交集!

\* select \* from A inner join B on A.字段 = B.字段;

\* 隐式内连接:

\* select \* from A,B where A.字段 = B.字段;

\* 外连接:

\* 左外连接:

\* select \* from A left outer join B on A.字段 = B.字段;

\* 右外连接:

\* select \* from A right outer join B on A.字段 = B.字段;

HQL多表的查询:

\* 连接:

\* 交叉连接:

\* 内连接:

\* 隐式内连接:

**\* 迫切内连接:**

\* 左外连接:

**\* 迫切左外连接:**

\* 右外连接:

\* HQL的内连接和迫切内连接区别:

\* 内连接查询 :将数据封装一个List<Object[]>中.

\* 迫切内连接 :将数据封装一个List<Customer>中.但是迫切内连接,得到会有重复记录 ,需要使用distinct排重.

### QBC:

1. 查询所有记录:

List<Customer> list = session.createCriteria(Customer.class).list();

for (Customer customer : list) {

System.out.println(customer);

}

1. 排序:

List<Customer> list = session.createCriteria(Customer.class)

.addOrder(org.hibernate.criterion.Order.desc("id")).list();

for (Customer customer : list) {

System.out.println(customer);

}

1. 分页:

Criteria criteria = session.createCriteria(Order.class);

criteria.setFirstResult(10);

criteria.setMaxResults(10);

List<Order> list = criteria.list();

for (Order order : list) {

System.out.println(order);

}

1. 获取单个对象:

Customer customer = (Customer) session.createCriteria(Customer.class)

.add(Restrictions.eq("cname", "小明")).uniqueResult();

System.out.println(customer);

1. 带参数的查询:

/\*

\* List<Customer> list = session.createCriteria(Customer.class)

\* .add(Restrictions.eq("cname", "小明")).list();

\* System.out.println(list);

\*/

List<Customer> list = session.createCriteria(Customer.class)

.add(Restrictions.eq("cname", "小明"))

.add(Restrictions.eq("cid", 2)).list();

System.out.println(list);

1. 模糊查询:

Criteria criteria = session.createCriteria(Customer.class);

criteria.add(Restrictions.like("cname", "大%"));

List<Customer> list = criteria.list();

System.out.println(list);

### SQL:

1. SQL语句查询所有记录:

List<Object[]> list = session.createSQLQuery("select \* from customer").list();

for (Object[] objects : list) {

System.out.println(Arrays.toString(objects));

}

List<Customer> list = session.createSQLQuery("select \* from customer")

.addEntity(Customer.class).list();

for (Customer customer : list) {

System.out.println(customer);

}