Hibernate初探之单表映射

什么是ORM
hibernate简介
编写一个小列子的需要的东西

- ORM(Object/Relationship Mapping): 对象/关系映射

利用面向对象思想编写的数据库应用程序最终都是把对象信息保存在关系型数据库中，于是要编写很多和底层数据库相关的SQL语句。

写sql语句有什么不好吗？

1.不同的数据库使用的SQL语法不同。比如:PL/SQL与T/SQL

2.同样的功能在不同的数据库中有不同的实现方式。比如分页SQL。

3.程序过分依赖SQL对程序的移植及扩展，维护等带来很大的麻烦。

Hibernate是java领域开源的一款ORM框架技术
Hibernate对JDBC进行了非常轻量级的对象封装。

其他主流的ORM框架技术：
1、MyBatis：前身就是著名的iBatis。
2、Toplink：后被Oracle收购，并重新包装为Oracle AS TopLink。
3、EJB：本身是JAVAEE的规范。

开发工具：Eclipse Standard Kepler
Hibernate Tools for Eclipse Plugins

Hibernate Tools是由JBoss推出的一个Eclipse综合开发工具插件，该插件可以简化ORM框架Hibernate,以及JBoss Seam,EJB3等的开发工作。

编写Hibernate例子的步骤：
1.创建Hibernate的配置文件（Hibernate.cfg.xml）
2.创建持久化对象 （javaBean）
3.创建对象-关系映射文件(JavaBean.hbm.xml)
4.通过Hibernate API编写访问数据库的代码

JPA annotation mappings require at least one Session Factory

创建Hibernate的配置文件

```
<property name="connection.username">root</property>
<property name="connection.password"></property>
<property name="connection.driver_class">com.mysql.jdbc.Driver</property>
<property
name="connection.url">jdbc:mysql:///hibernate?useUnicode=true&amp;characterEncodi
ng=UTF-8</property>
<property name="dialect">org.hibernate.dialect.MySQLDialect</property>
```

java Bean 类 遵循的4个条件 public class student {

1.共有的类
2.提供公有的不带参数的默认的构造方法
3.属性私有
4.属性setter/getter 封装方法
}

Students.hbm.xml

对象-关系映射配置文件创建完成之后，要讲该文件引入到hibernate的配置文件中，通过<mapping resource=""/>标签引入，resource中填入对象关系映射文件名

使用Junit测试

@Test:测试方法
@Before:初始化方法
@After:释放资源

- **通过Hibernate API编写访问数据库的代码**

```
Configuration config = new Configuration().configure();//创建配置对象

ServiceRegistry serviceRegistry = new ServiceRegistryBuilder().
 applySettings(config.getProperties()).buildServiceRegistry();//创建服务注册对象

sessionFactory = config.buildSessionFactory(serviceRegistry);//创建会话工厂对象

session = sessionFactory.openSession();//打开会话

transaction = session.beginTransaction();//打开事务
```

hibernate.cfg.xml常用配置：

hibernate.cfg.xml常用配置

属性名字	含义
hibernate.show_sql	是否把Hibernate运行时的SQL语句输出到控制台，编码阶段便于测试。
hibernate.format_sql	输出到控制台的SQL语句是否进行排版，便于阅读。建议设置为true。
hbm2ddl.auto	可以帮助由java代码生成数据库脚本，进而生成具体的表结构。create|update|create-drop|validate
hibernate.default_schema	默认的数据库。
hibernate.dialect	配置Hibernate数据库方言，Hibernate可针对特殊的数据库进行优化。

1.hbm2ddl.auto的四个值：
a.create：表示启动的时候先drop，再create
b.create-drop: 也表示创建，只不过再系统关闭前执行一下drop
c.update: 这个操作启动的时候会去检查schema是否一致，如果不一致会做scheme更新
d.validate: 启动时验证现有schema与你配置的hibernate是否一致，如果不一致就抛出异常，并不做更新
2.hibernate.default_schema:给每个表名加前缀，一般属性值为当前表名的数据库名
3.hibernate.dialect:方言的使用，hibernate可针对特殊的数据库进行优化

下面是一个列子

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//com.Hibernate/com.Hibernate Configuration DTD//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="connection.url">jdbc:mysql://localhost:3306/spitter</property>
```

```
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
      <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
       <property name="hbm2ddl.auto">update</property>
      <property name="connection.username">root</property>
      <property name="connection.password">liuziye</property>
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>
      <property name="hibernate.current_session_context_class">thread</property>

      <mapping resource="hibernate/Account.hbm.xml"/>

      <mapping resource="hibernate/Address.hbm.xml"/>



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

注意：hibernate的前缀可以省略，即：hibernate.dialect等同于dialect

- **hibernate的执行流程**

1.创建Configuration对象，用来读取Hibernate.cfg.xml文档，通过这个对象可以创建SessionFactory对象
2.SessionFactory对象读取对象映射文件，通过这个对象可以创建Session对象
3.Session对象调用方法执行操作，并且创建一个事物，这些方法都得封装在事物当中，操作完成后先提交事物，再关闭连接
1.不建议直接使用jdbc的connection操作数据库，而是通过session操作数据库。
2.session可以了解为操作数据库的对象，操作数据库之前必须先获取session的实例
3.**session与connection，是多对一关系**，每个session都有一个与之对应的connection，一个connection不同时刻可以供多个session使用。
4.把对象保存到关系数据库中需要调用session的各种方法：save(),update(),delete(),createQuery

- **hibernate对数据的操作都是封装在事务当中，并且默认是非自动提交的方式。所以用session保存对象时，如果不开启事务，并且手工提交事务，对象并不会真正保存在数据库中。**

- **如果你想让hibernate像jdbc那样自动提交事务，必须调用session对象的doWork()方法,获得jdbc的connection后，设置其为自动提交事务模式。（注意：通常并不推荐这样做）**

transaction简介：
1.Hibernate推荐使用手工开启，提交事物的方式
a.transaction=session.beginTransaction();
b.transaction.commit();
2.如果使用自动提交的方式，需要调用doWork()方法,并且要求刷新session.flush();
a. session.doWork(new Work() {

@Override
public void execute(Connection connection) throws SQLException {
// TODO Auto-generated method stub

```
connection.setAutoCommit(true);
}
});
```

Sesson详解：
1.如何获得session对象：
a.openSession
b.getCurrentSession,使用这个方法需要在hibernate.cfg.xml文件中进行配置(注意一点 使用getCurrentSesson的时候 由于都是自动控制事务的提交的
不是手动提交事务 有的时候在本地操作成功之后但是数据库没有值的情况下 就是出错了 但是getCurrentSesson是不会显示为什么会出错的 因为在本地中没有提交上去也不会认为是出错 这个时候需要使用oppenSession自己手动提交事务来查看具体原因)

如果使用getCurrentSession需要在hibernate.cfg.xml文件中进行配置：
如果是本地事务（jdbc事务）
<propertyname="hibernate.current_session_context_class">thread</property>
如果是全局事务（jta事务）
<propertyname="hibernate.current_session_context_class">jta</property>

seesion详解

- openSession与getCurrentSession的区别

1. getCurrentSession在事务提交或者回滚之后会自动关闭，而openSession需要你手动关闭。如果使用openSession而没有手动关闭，多次之后会导致连接池溢出。

2. openSession每次创建新的session对象，getCurrentSession使用现有的session对象。

下面就是一个配置的Demo

```
public class hibernateDao {
static SessionFactory sessionFactory;
static Session session;

static {

// 获得配置对象
Configuration config = new Configuration().configure("hibernate/hibernate.cfg.xml");
// 服务注册对象
ServiceRegistry serviceRegistry = new ServiceRegistryBuilder().applySettings(config.getProperties())
                .buildServiceRegistry();

sessionFactory = config.buildSessionFactory(serviceRegistry);
session = sessionFactory.getCurrentSession();
session.beginTransaction();

/* 自动提交
        session.doWork(new Work() {
            @Override
            public void execute(Connection connection) throws SQLException {
                connection.setAutoCommit(true);
            }
        });*/
}


public static void main(String[] args) {
Account accout = new Account();
Name name = new Name();
name.setBigName("大名就是这个");
name.setSmallName("小名就是这个");
accout.setId(4);

accout.setMoney(6000.0);
accout.setName(name);

saveAccout(accout);
```

```
System.out.println(getAccout(4));

    }

public static void saveAccout(Account account) {

session.save(account);

    }


public  static  Account  getAccout(int accountid){
Account account= (Account) session.load(Account.class, accountid);
return account;
    }



/*在使用openSession需要*/
public static void closeSessionAndCommitTransaction() {
session.getTransaction().commit();
session.close();
    }
}
```

hbm配置文件常用设置

```
<hibernate-mapping
      schema="schemaName"
      catalog="catalogName"
      default-cascade="cascade_style"        //级联风格
      default-access="field|property|ClassName"    //访问策略
      default-lazy="true|false"              //加载策略
      package="packagename"
/>
<class
      name="ClassName"
      table="tableName"
      batch-size="N"
      where="condition"
      entity-name="EntityName"
/>
```

```
<id
      name="propertyName"
      type="typename"
      column="column_name"
      length="length"
      <generator class="generatorClass"/>
</id>
```

主键生成策略 是在xxx.hbm.xml中生成的

标识符生成器	描述
increment	适用于代理主键。由Hibernate自动以递增方式生成。
identity	适用于代理主键。由底层数据库生成标识符。
sequence	适用于代理主键。Hibernate根据底层数据库的序列生成标识符，这要求底层数据库支持序列。
hilo	适用于代理主键。Hibernate分局high/low算法生成标识符。
seqhilo	适用于代理主键。使用一个高/低位算法来高效的生成long，short或者int类型的标识符。
native	适用于代理主键。根据底层数据库对自动生成标识符的方式，自动选择identity、sequence或hilo。
uuid.hex	适用于代理主键。Hibernate采用128位的UUID算法生成标识符。
uuid.string	适用于代理主键。UUID被编码成一个16字符长的字符串。
assigned	适用于自然主键。由Java应用程序负责生成标识符。
foreign	适用于代理主键。使用另外一个相关联的对象的标识符。

1.native：自适应所使用的数据库
2.assignd:自己在程序中手工赋值

单一主键：
1.两种主键生成策略如图
2.使用assigned未给id赋值，默认为0
3.如果表中已经有数据了，使用native方式添加数据，id值在原有数据中最大id值上加1

下面俩张图是java中类型与SQL类型对应表

Hibernate映射类型	Java类型	标准SQL类型	大小
integer/int	java.lang.Integer/int	INTEGER	4字节
long	java.lang.Long/long	BIGINT	8字节
short	java.lang.Short/short	SMALLINT	2字节
byte	java.lang.Byte/byte	TINYINT	1字节
float	java.lang.Float/float	FLOAT	4字节
double	java.lang.Double/double	DOUBLE	8字节
big_decimal	java.math.BigDecimal	NUMERIC	
character	java.lang.Character/java.lang.String/char	CHAR(1)	定长字符
string	java.lang.String	VARCHAR	变长字符
boolean/ yes_no/true_false	java.lang.Boolean/Boolean	BIT	布尔类型
date	java.util.Date/java.sql.Date	DATE	日期
timestamp	java.util.Date/java.util.Timestamp	TIMESTAMP	日期
calendar	java.util.Calendar	TIMESTAMP	日期
calendar_date	java.util.Calendar	DATE	日期

映射类型	Java 类型	标准 SQL 类型	描述
date	java.util.Date 或 java.sql.Date	DATE	代表日期: yyy-MM-dd
time	java.util.Date 或 java.sql.Time	TIME	代表时间: hh:mi:ss
timestamp	java.util.Date 或 java.sql.Timestamp	TIMESTAMP	代表时间和日期: yyyymmddhhmiss
calendar	java.util.Calendar	TIMESTAMP	同上
calendar_date	java.util.Calendar	DATE	代表日期: yyy-MM-dd

mysql当中不支持sql的CLOB类型，在mysql中，用text，MEDIUMTEXT及LONGTEXT
类型来表示长度超过255 的长度文本数据

@实体类中存在一个实体类属性　　就是存javabean中存在一个类类型的属性

- **实体类中的某个属性属于用户自定义的类的对象。**

```
<component name="address" class="Address">
    <property name="postcode" column="POSTCODE"></property>
    <property name="phone" column="PHONE"></property>
    <property name="address" column="ADDRESS"></property>
</component>
```

如下配置

```
<class name="com.Hibernate.Dao.entity.Account" table="account" schema="" catalog="spitter">
    <id name="id">
        <column name="id" sql-type="int" not-null="true"/>
        <generator class="assigned"/>
    </id>
<!--  <property name="name">
        <column name="name" sql-type="varchar" length="20" not-null="true"/>
    </property>-->
<component name="name" class="com.Hibernate.Dao.entity.Name">
         <property name="bigName" column="bigName"></property>
         <property name="smallName" column="smallName"></property>
    </component>

    <property name="money">
        <column name="money" sql-type="double" precision="22"/>
    </property>

</class>
```

单表CRUD操作实例
1 save
2 update
3 delete
4 get/load(查询单个记录)

get方法有俩个参数，一个是（要查找的数据的类类型，查询的数据）；
get与load的区别
在不考虑缓存的情况下，**get方法会在调用之后立即向数据库发出sql语句**
返回持久化对象
load方法在调用后返回一个代理对象
该对象只保存了实体对象的id，直到使用对象的非主键属性时才会发出sql语句

查询数据库中不存在的数据时，get方法返回null，
load方法抛出异常org.hibernate.ObjectNotFoundException

```
session.get();
session.load();
```