Hibernate基本配置

# hibernate.cfg.xml常用配置

hibernate.cfg.xml文件主要用于配置数据库连接和Hibernate运行时所需的各种属性，这个配置文件应该位于应用程序或Web程序的类文件夹 classes中，我们根据上一篇中的示例来简要分析以下这个文件中的一些常用配置。

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">  <hibernate-configuration>  <session-factory>  <!-- 对应的数据库驱动 -->  <property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>  <!-- 数据库密码 -->  <property name="hibernate.connection.password">123</property>  <!-- 数据库用户名 -->  <property name="hibernate.connection.username">root</property>  <!-- 数据库url，类似于JDBC中连接数据库的url -->  <property name="hibernate.connection.url">jdbc:mysql://172.16.129.212:3306/test?useUnicode=true&amp;characterEncoding=UTF-8</property>  <!-- 数据库使用的方言 -->  <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>    <!-- 是否把输出到控制台的sql进行格式化 -->  <property name="format\_sql">true</property>  <!-- 是否将Hibernate运行时的sql输出到控制台 -->  <property name="show\_sql">true</property>  <!-- 自动创建|更新|验证数据库表结构 -->  <property name="hbm2ddl.auto">create</property>    <mapping resource="User.hbm.xml"/>  </session-factory>  </hibernate-configuration> |

在这个文件中属性里的hibernate前缀可以省略，例如hibernate.dialect是等同于dialect的。从这个例子中我们可以看到，hibernate.cfg.xml这个配置文件主要配置了数据库连接的一些信息，包括

* 数据库的驱动：connection.driver\_class、
* 数据库的用户名：connection.username、
* 数据库的密码：connection.password和、
* 数据库连接的JDBC的URL：connection.url，
* 同时Hibernate还为不同的数据库提供了不同的方言dialect。
* show\_sql：是否会将SQL打印到控制台，
* format\_sql：是否要进行格式化，
* hbm2ddl.auto：用于帮助由Java生成数据库脚本进而生成数据库表结构的，这个属性的值包括create、update、create-drop和validate，create表示每次加载Hibernate时都会删除上一次的生成的表，然后根据持久化类再重新生成新表，即使两次没有任何改变也要这样执行，这种操作会导致之前运行时产生的数据丢失；update表示第一次加载Hibernate时会根据持久化类自动建立表结构，以后加载Hibernate时根据 持久化类自动更新表结构，即使表结构改变了但表中的行仍然存在不会删除以前的行，要注意的是当部署到服务器后，表结构是不会被马上建立起来的，是要等 应用第一次运行起来后才会；create-drop表示每次加载Hibernate时根据持久化类生成表，但是sessionFactory一关闭,表就自动删除，因此这个值平时要慎用；validate表示每次加载Hibernate时，验证创建数据库表结构，只会和数据库中的表进行比较，不会创建新表，但是会插入新值。
* current\_session\_context\_class是在使用getCurrentSession()获取session时用到的，这在下面会进行介绍。

# Hibernate的执行流程

|  |
| --- |
| **package** com.immoc.test;  **import** java.util.Date;  **import** org.hibernate.Session;  **import** org.hibernate.SessionFactory;  **import** org.hibernate.Transaction;  **import** org.hibernate.cfg.Configuration;  **import** org.junit.After;  **import** org.junit.Before;  **import** org.junit.Test;  **import** com.immoc.entity.User;  **public** **class** UserTest {  **private** SessionFactory sessionFactory;  **private** Session session;  **private** Transaction transaction;  @Before  **public** **void** init() {  //创建配置对象  Configuration config = **new** Configuration().configure();  //创建会话工厂对象  sessionFactory = config.buildSessionFactory();  //创建会话对象  session = sessionFactory.openSession();  //开启事务  transaction = session.beginTransaction();  }  @After  **public** **void** destory() {  //提交事务  transaction.commit();  //关闭会话  session.close();  //关闭会话工厂  sessionFactory.close();  }  @Test  **public** **void** saveNewUser() **throws** Exception {  User s = **new** User(1, "张三", 12, "男", **new** Date(), "上海市");  session.save(s);  }  } |

类中理解Hibernate的执行流程：首先执行的是init()方法，这个方法里主要包含一些操作数据库之前的初始化动作，首先使用Configuration对象读取配置文件，获取Hibernate的配置信息，然后使用Configuration对象生成会话工厂SessionFactory，SessionFactory是线程安全的，创建和销毁SessionFactory都是很消耗资源的，所以一般情况下一个系统中只有一个SessionFactory，创建了SessionFactory之后就可以创建一个会话Session了，Session是操作数据库对象，类似于JDBC中的Connection，但是不建议使用JDBC的Connection操作数据库，而要通过Session操作数据库，Hibernate在操作数据库之前必须要获得Session对象，Session与Conection是多对一的关系，每一个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。

在上面的这个例子中我们使用的是SessionFactory的openSession()获得Session对象的，其实还可以通过SessionFactory的getCurrentSession()获得Session对象，但是需要注意的是如果要使用getCurrentSession()获得Session对象时需要在hibernate.cfg.xml文件中配置

|  |
| --- |
| <!-- 本地事务（jdbc事务） -->  <property name="hibernate.current\_session\_context\_class">thread</property>  或者  [html] view plain copy 在CODE上查看代码片派生到我的代码片  <!-- 全局事务（jta事务） -->  <property name="hibernate.current\_session\_context\_class">jta</property> |

全局事务：资源管理器管理和协调的事务，可以跨越多个数据库和进程；

本地事务：在单个 EIS 或数据库的本地并且限制在单个进程内的事务，本地事务不涉及多个数据来源。

除了配置上的不同外，openSession()和getCurrentSession()还有什么不同呢？首先getCurrentSession()在事务提交或者回滚之后session会自动关闭，而openSession()则需要手动调用close()方法关闭session，如果没有手动关闭，当调用多次openSession()之后就会导致数据库连接池溢出；第二点不同是指openSession()每次都会创建一个新的session对象，而getCurrentSession()每次都是获取当前的session对象。

再继续看例子中的init()方法，当创建了会话Session之后，我们就需要开启事务了，Hibernate中对事务的操作都是封装在事务中的，并且默认的是非自动提交的方式，这一点与JDBC默认是自动提交事务的方式是不同的，所以在使用HIbernate时如果未手动提交事务，那么实际的数据并不会保存进数据库中。如果想要在Hibernate中实现自动提交事务，那么就需要调用Session的doWork()方法，在获得JDBC的Connection之后，可以设置事务为自动提交，但是平常的编码中我们不推荐这样使用。下面通过改写以下上面的小程序来具体讲解一下讲的Hibernate中事务的使用。

首先如果我们没有手动提交事务，即在destory()方法中删除Transaction的commit()方法。

|  |
| --- |
| @After  public void destory() {  //提交事务  //transaction.commit();  //关闭会话  session.close();  //关闭会话工厂  sessionFactory.close();  }  运行程序，观察数据库里的数据，我们会发现张三这个用户信息并没有插入到表中，这说明事务并没有自动提交。 |
| 接下来我们再设置事务自动提交，仍然不显式的调用Transaction的commit()方法，只调用Session的doWork()方法，在doWork()里设置JDBC的Connection为自动提交事务，然后运行程序，查看数据库表，我们会发现数据插入成功了，说明手动设置数据库自动提交成功了。  @Test  public void saveNewUser() throws Exception{  User s = new User(1, "张三", 12, "男", new Date(), "上海市");  session.doWork(new Work() {  public void execute(Connection arg0) throws SQLException {  arg0.setAutoCommit(true);  }  });  session.save(s);  //设置事务自动提交时在保存数据之后要调用flush()方法  session.flush();  } |

# 映射文件hbm.xml的常用配置

在上一篇的例子中我们用到了映射文件User.hbm.xml，下面我们对里面一些基本的配置做一个简单的介绍。

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">  <hibernate-mapping package="com.immoc.entity">  <class name="User" table="USER">  <id name="id" type="int">  <column name="ID" />  <generator class="assigned" />  </id>  <property name="name" type="java.lang.String">  <column name="NAME" />  </property>  <property name="age" type="int">  <column name="AGE" />  </property>  <property name="sex" type="java.lang.String">  <column name="SEX" />  </property>  <property name="birthday" type="java.util.Date">  <column name="BIRTHDAY" />  </property>  <property name="address" type="java.lang.String">  <column name="ADDRESS" />  </property>    </class>  </hibernate-mapping> |

首先class表明了POJO与实际映射的表，其中name指的是映射的POJO类名，而table指的是映射的表名。id表示表的主键，name指的是主键在POJO类里面对应的属性，type为主键的类型，column子标签表示映射到表里面的列，genertor是指生成主键的策略，Hibernate支持很多种生成主键的策略，具体的可以参考下表。



我们常用的设置方法是native和assigned，其中native是根据底层的数据库自动采用不同的方式生成主键，代理主键指的是指与业务无关且能唯一标识数据库中记录，一般是数据库自动生成的，比如MySQL可以使用auto\_increment，Sql2000可以使用identity生成方式，Oracle可以使用sequence生成方式。assigned是我们在程序中手动设置的主键值，通常是与业务相关的，能够唯一标识数据的主键。这里还有一个注意点，如果使用assigned赋值的话，如果实体类中设置的主键id是基本类型int的话，则可以不用赋值，系统默认值为0；如是引用类型Integer话，则默认值为null，不赋值系统则报错。使用native时，系统会自动选择该数据库对应的自动增值方式，从1开始，即使手动给他赋值，也不会起作用，但也不会报错。

在hbm.xml文件中我们还用到了type来映射Java与数据库的类型，Hibernate的基本映射类型可以参考下表（表格摘自慕课网）。我们在type属性里既可以填写Hibernate的映射类型，也可以填写Java类型。



在这里简单讲一下时间类型，时间类型有下面这几种，其中date类型主要包含的是年月日的信息，即yyyy-MM-dd，不包含时分秒，而time只包含时分秒的信息。



对于大的文件类型，Hibernate也有相应的映射，其中clob和text对应大文本文件，blob对应大的二进制文件，例如视频音频图片，Java中的Clob对应存储大文本文件，Java中的Blob对应存储大二进制文件,这里要注意MySQL不支持标准SQL的CLOB类型，在Mysql中，用TEXT，MEDIUMTEXT及LONGTEXT类型来表示长度超过255的长文本数据



我们修改User来测试一下大文件类型的使用，首先我们在User类中添加一个大文件类型picture

|  |
| --- |
| import java.sql.Blob;  import java.util.Date;  public class User {  private int id;  private String name;  private int age;  private String sex;  private Date birthday;  private String address;  //大文件类型  private Blob picture;  //无参数构造器，set ,get ,toString方法... |
| 然后我们修改映射文件，添加大文件类型的映射。  <property name="picture" type="blob">  <column name="PICTURE" />  </property> |
| @Test  public void saveNewUser() throws Exception{  User s = new User(1, "张三", 12, "男", new Date(), "上海市");    InputStream input = new FileInputStream(new File("/home/user/test.png"));  Blob picture = Hibernate.getLobCreator(session).createBlob(input, input.available());  s.setPicture(picture);    session.save(s);  } |

我们还可以在hbm.xml文件中设置组件属性，所谓的组件属性就是指在实体类中的某个属性是用户的自定义类型的，我们通过一个例子来分析，首先我们定义一个自定义类型Address，保存用户的地址信息。

|  |
| --- |
| public class Address {  private String postCode;  private String street;  private String city;  //构造器，set,get,toString方法 |
| public class User {  private int id;  private String name;  private int age;  private String sex;  private Date birthday;  private Blob picture;  //组件类型  private Address address;  //构造器，set,get,toString方法 |

|  |
| --- |
| 然后修改映射文件里的address的映射，添加一个component标签表示为组件属性，同时指向这个组件属性的类。<component name="address" class="com.imooc.vo.Address">  <property name="postCode" column="POSTCODE"></property>  <property name="street" column="STREET"></property>  <property name="city" column="CITY"></property>  </component> |
| @Test  public void saveNewUser() throws Exception{  User s = new User(1, "张三", 12, "男", new Date());  Address address = new Address("224400", "街道", "城市");  s.setAddress(address);    InputStream input = new FileInputStream(new File("/home/cuixin/test.png"));  Blob picture = Hibernate.getLobCreator(session).createBlob(input, input.available());  s.setPicture(picture);  session.save(s);  } |

# 单表的增删改查操作

Hibernate为数据库的增删改查提供了相应的API支持，这里我们先简单介绍一下对于单表的增删改查操作。

## 增加方法save()

save()方法可以说已经很熟悉了，因为在前面的测试方法中都使用了save()方法进行数据库的插入，在这里就不再重复介绍了。

## 查询方法get()/load()

Hibernate对于查询数据库提供了两种方法，get()和load()，我们仍然根据上面使用的User类来进行测试，首先我们使用save()方法向数据库中插入一个张三的用户，然后我们分别用get()和load()方法查询数据库。

|  |
| --- |
| @Test  public void testGetUser() {  User s = session.get(User.class, 1);  System.out.println(s);  }    @Test  public void testLoadUser() {  User s = session.load(User.class, 1);  System.out.println(s);  } |
| get()方法和load()方法有什么不同呢？  首先在不考虑缓存的情况下，get()方法在调用后立即向数据库发出sql语句，返回持久化对象，而load()方法在调用后则会返回一个代理对象，这个代理对象只保存了实体对象的id，一直等到要使用实体对象的非主键属性时才会发出sql语句；  第二点不同是当查询数据库不存在的数据时，get()方法会返回null，而load()方法会抛出异常org.hibernate.ObjectNotFoundException。 |

## 更新方法update()

Hibernate使用update()来更新数据库的数据

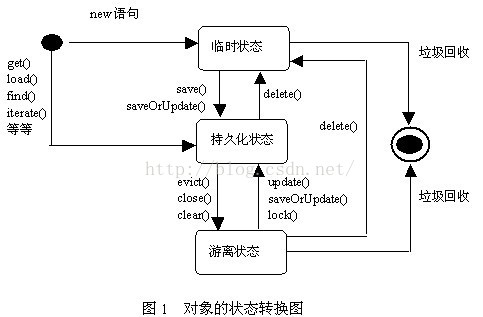
|  |
| --- |
| @Test  public void testUpdateUser() {  User s = session.get(User.class, 1);  s.setSex("女");  session.update(s);  } |

## 删除方法delete()

Hibernate使用delete()方法删除数据，例如下面这个例子就是删除主键为1的用户。

|  |
| --- |
| @Test  public void testDeleteUser() {  User s = session.get(User.class, 1);  session.delete(s);  } |

## hibernate中对象的状态有三种



session.save()：该方法可以把一个对象从临时装填转换成持久化状态

session.get()：从数据库中根据主键提取出一个对象，该对象就是一个持久化状态的对象

session.update()：把一个对象变为持久化对象

session.evict()：把一个持久化状态的对象变为脱管状态

session.clear()：把所有Hibernate中的持久化状态的对象变为脱管状态的对象

注意：每次进行transaction.commit()时,都会进行flush操作,进而进行对象与副本的数据对比,决定是否发送sql语句，比如取出的user的age是10,又改为10,user.setAge(10);那么这样是不会发送sql语句的。实际上sql的发送时刻,发生在session.flsuh()，

在transaction.commit()之前,如果有session.flush()操作,就会发送sql语句,而不再等待transaction.commit().如果持久化对象是由临时状态转换过来的就发出 insert 语句；如果持久化对象是由 get 等方法得到的；再次查看副本，如果与副本对照后，一致就不发送sql，如果不一致，就发生update语句

这也就是为什么，不使用session.update() 同样可以发送 update 语句。