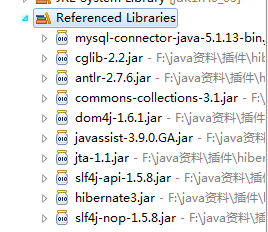
# Hibernate基础

Hibernate意思是“冬眠”，整个架构意思是持久化，Hibernate最重要的理解ORM，Object-Relation Mapping，即对象——关系映射，这主要负责业务对象的持久化，封装了数据库访问细节，这就是hibernate产生的核心原因，因为项目的业务逻辑模型是面向对象，而关系型数据库模型就是面向关系的，所以Hibernate把一个持久化的类和数据库中的一张表对应，类的每一个实例对应这张表是一条记录，类的每一个属性，对应这表里的一个字段，这样就把一个对象和一个关系数据库映射在一起，这就是ORM 模式

## 第一个Hibernate架构：

1. 新建Test project，java
2. 导包:
3. 导入hibernate-distribution-3.5.5-Final/hibernate3.jar
4. 导入hibernate-distribution-3.5.5-Final\lib\bytecode\cglib\包
5. 导入hibernate-distribution-3.5.5-Final\lib\required\全部的包
   1. 注意： hibernate自带的是：slf4-api-1.5.8中 而需要的是：自己去下载slf4.org官网中下载相应版本的slf4j-1.5.8 nop：no dependent
6. 在导入数据库包



1. 导配置：

\project\etc中其中一个，最好是hibernate.cfg.xml

把该文件导入到src下

基本文件信息：

<hibernate-configuration>

<session-factory>

<property name=*"hbm2ddl.auto"*>update</property>

<property name=*"show\_sql"*>true</property>

<property name=*"format\_sql"*>true</property>

<property name=*"myeclipse.connection.profile"*>MySql</property>

<property name=*"connection.url"*>

jdbc:mysql://localhost:3306/test

</property>

<property name=*"connection.username"*>root</property>

<property name=*"connection.password"*>root</property>

<property name=*"connection.driver\_class"*>

com.mysql.jdbc.Driver

</property>

<property name=*"dialect"*>

org.hibernate.dialect.MySQL5Dialect

</property>

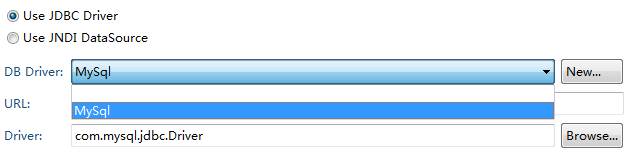
<mapping resource=*"com/hibernate/po/User.hbm.xml"* />

</session-factory>

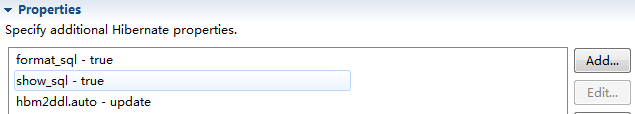
</hibernate-configuration>

#一个表配置一个session-factory name=”表名”

1. 自动配置数据库



1. 自动显示数据

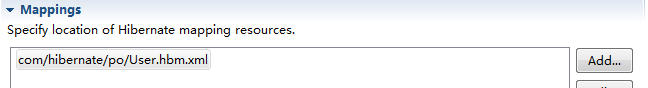


Format\_sql – true 表示显示格式化数据库

Show\_sql – true 表示显示数据库

**Hbm2ddl-auto – update 当不存在使用表的时候，比写**

1. 自动配置实体映像文件（）



1. 复制映像文件：

对象与关系之间的映像文件，配置文件根据不同映像文件来实现对象与关系转换

在project\testing\src\main\java\org\hibernate\test\cache\ Item.hbm.xml

<hibernate-mapping>

<class name=*"com.hibernate.po.User"* table=*"t\_users"*>

<id name=*"id"*>

<generator class=*"native"*/>

</id>

<property name=*"username"* type=*"java.lang.String"* update=*"true"* insert=*"true"*

access=*"property"*  column=*"username"* length = *"32"* />

<property name=*"password"*

type=*"java.lang.String"*

/>

</class>

</hibernate-mapping>

类名加包名叫全限定名

Xml映像文件中各种细节：

1.<hibernate-mapping/>查文档，都是偏僻的

2. <class

name="ClassName" 可选，当这个属性不存在时，则会假定这是非pojo实体映像（不写name说明就不能实现操作）com.hibernate.pojo.User

table="tableName" 对应的数据库名 t\_user

lazy="true|false" (16) 通过设置 lazy="false"，所有的延迟加载（Lazy fetching）功能将被全部禁

/>

3. <id

name="propertyName" 必写，（主键中poio属性名）标示符

type="typename" 数据类型（int）

column="column\_name" 设置主键字段

<generator class="generatorClass"/

其中属性名： increment 用于long int short类型生成唯一标示符，只有没有其他进程插入该表时，才能使用

**Identity** 对 DB2，MySQL，MS SQL Server，Sybase 和 HypersonicSQL 的内置标识字段提供支持。返回的标识符是 long，short 或者 int 类型的。

**sequence**在 DB2，PostgreSQL，Oracle，SAP DB，McKoi 中使用序列（sequence）， 而在 Interbase中使用生成器（generator）。返回的标识符是 long，short 或者 int 类型的。

**foreign**使用另外一个相关联的对象的标识符。它通常和 <one-to-one> 联合起来使用。

</id

> <class name=”实体类名” table=”表名”

<generator class=*"native"*/>

Native是系统自动根据数据库来选择自动填充字段

1. po：User

**public** **class** User {

**private** **int** id;

**private** String username;

**private** String password;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

}

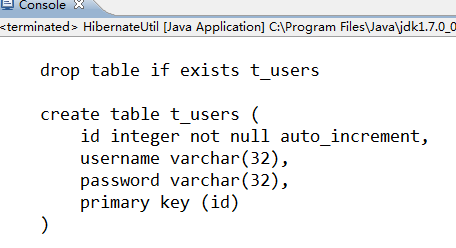
1. 创建表示例

Configuration cfg = **new** Configuration().configure();

//读取配置文件，如果configuration是读取properties文件，configuration（）.configure(),才是读取配置文件hibernate.cfg.xml

SchemaExport se = **new** SchemaExport(cfg);

se.create(**true**, **true**);



显示出，先把原来的表删除，才创建

1. 增加：

**import** org.hibernate.HibernateException;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.Configuration;

**import** com.hibernate.po.User;

**public** **class** T {

**public** **static** **void** main(String[] args) {

Configuration cfg = **new** Configuration().configure();

SessionFactory sf = cfg.buildSessionFactory();

Session session = sf.openSession();

Transaction ts = session.getTransaction();

//Transaction ts = session.beginTransaction();实现getTransaction与begin()

**try** {

ts.begin();//开始

User user = **new** User();

user.setUsername("admin1");

user.setPassword("12345");

session.save(user);

ts.commit();

} **catch** (HibernateException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

ts.rollback();

}**finally**{

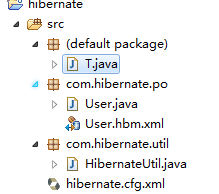
session.clear();

}

}

}

第一次写入：

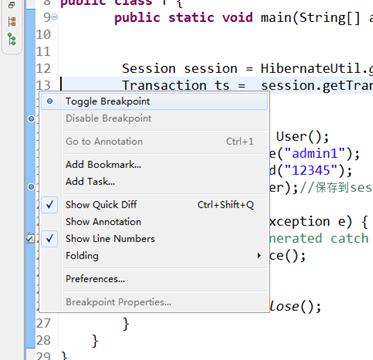


## 断点模式debug

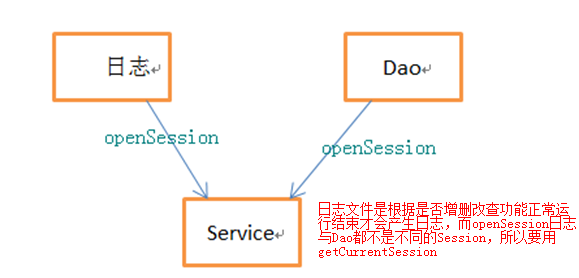
双击界面序号（或者如下图）--进入debug模式



1.进入方法 2.进行下一步



### Session



### getCurrentSession()

1. 如果没有Session存在，就openSession一个新的
2. 如果存在Session直接拿来使用
3. 要在配置文件中加

<property name=*"current\_session\_context\_class"*>thread</property>

1. 不需要关闭（如果关闭，那么就和openSession一样）

其中配置文件中值可以是thread，在该电脑线程中找到，也可以JTA（Java[事务](http://baike.baidu.com/view/121511.htm)API）

JAT：

JTA允许应用程序执行分布式事务处理——在两个或多个网络计算机资源上访问并且更新数据。[JDBC](http://baike.baidu.com/view/25611.htm" \t "_blank)[驱动程序](http://baike.baidu.com/view/1048.htm)的JTA支持极大地增强了数据访问能力。

## Hibernate 对象三种状态及状态之间转换

### 瞬时状态 (Transient)

1. 没有被Session管理
2. 数据库不在改数据

User user = **new** User();

user.setUsername("tom");

user.setPassword("12345");上面对象是瞬时状态

当我们通过java的new关键字生成一个实体对象，这个实体对象就是处于自由状态，为啥是自由状态，因为user只是在jvm获取到一个内存空间，并没有通过Session对象来Save（）保存到数据库中，没有进入hibernate的缓冲管理中

### 持久化状态

1） 已经被Session管理

2） 数据库存在该对象

### 3.离线状态

1）没有被Session管理

2）存在数据库中

将离线对象更新成持久化对象

## Hql语言

### Update，修改数据中示例

User user = **new** User();

user.setId(1);//条件

session.update(user);

user.setUsername("999");//set username=?

但是这样就只能改了复制值，其余null

Sql语句：局部修改

Query query = *session*.createQuery("update User set username =? where id =?");

query.setString(0,"4444");

query.setInteger(1, 4);

query.executeUpdate();

### 查询：

###### 用get和load查询出

(User)session.load(User.class, 2);

get不会马上发出sql语句，[代理模式](Hql.docx)，而是当真正使用这个对象的时候才会发出这个对象的值（延迟加载）

User user = (User)session.get(User.class, 3);

load方法马上发出sql语句，返回的真实对象，对象中赋值

System.out.println(user.getUsername());

User user = (User) session.get(User.**class**,1);

System.*out*.println(user.getUsername());

session.evict(user);

// session.clear();

System.*out*.println(user.getUsername());

//从断点中可以看出 查询过程中，先从Session缓冲中找到，所以效率高

//当evict清除Session，所以必须commit才能得到

所以hibernate效率

Evict和clear区别是：evict是清除某一个对象的Session，而clear是所有的Session

###### 2.Hql语句：查询语句可以返回多个对象和（或）属性，存放在 Object[]队列中

Hql语句得到多个时候，query.list();

单个是：query. uniqueResult()

Query query2 = session.createQuery("select username,password from User");

List<Object[]> list2 = query2.list();

**for**(Object[] object : list2){

String name = (String)object[0];

String passwd = (String)object[1];

System.out.println(name + " : " + passwd);

}

**3.条件查询**

Query query4 = *session*.createQuery("from User where username=? and password=?");

**第一种方式：**

query4.setString(0, "tom");

面向对象的原则，所以需要从0开始

// query4.setString(1, "12345");

**第二种方式**setParameter

1.Query query = session.createQuery("select age from Test where name=?");

query.setParameter(0, name);

String name = "t1";

2. Query query = session.createQuery("select age from Test where age in(?,?,?)");

query.setParameter(0, 11);

query.setParameter(1, 13);

query.setParameter(2, 15);

3. Query query = session.createQuery("select age from Test where age in(:ages)");

query.setParameterList("ages", **new** Object[]{11,13,15});

List<Integer> i = query.list();

**for**(**int** age : i ){

System.*out*.println(age);

}

Query query5 = *session*.createQuery("from User where username=:name and password=:pwd");

query5.setString("name", "tom");

query5.setString("pwd", "12345");

List<User> list5 = query5.list();

**for**(User u : list5){

System.*out*.println(u.getUsername());

}

Query query6 = *session*.createQuery("from User where username like '%t%' ");

##### 4.Itertor与list区别：

**List:**

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

List<Test> tests1 = query.list();

**for**(Test t : tests1){

System.*out*.println(t.getName());

}

Query query2 = session.createQuery("from Test");

List<Test> tests2 = query2.list();

**for**(Test t : tests2){

System.*out*.println(t.getName());

}

Hibernate:

select

test0\_.id as id0\_,

test0\_.age as age0\_,

test0\_.name as name0\_

from

t\_test test0\_

t0

t1

Hibernate:

select

test0\_.id as id0\_,

test0\_.age as age0\_,

test0\_.name as name0\_

from

t\_test test0\_

t0

t1

**iterate:**

Query query1 = session.createQuery("from Test");

Iterator<Test> t1 = query1.iterate();

**for**(;t1.hasNext();){

Test test =t1.next();

System.*out*.println(test.getName());

}

Query query2 = session.createQuery("from Test");

Iterator<Test> t2 = query2.iterate();

**for**(;t2.hasNext();){

Test test =t2.next();

System.*out*.println(test.getName());

}

Hibernate:

select

test0\_.id as col\_0\_0\_

from

t\_test test0\_

Hibernate:

select

test0\_.id as id0\_0\_,

test0\_.age as age0\_0\_,

test0\_.name as name0\_0\_

from

t\_test test0\_

where

test0\_.id=?

t0

Hibernate:

select

test0\_.id as id0\_0\_,

test0\_.age as age0\_0\_,

test0\_.name as name0\_0\_

from

t\_test test0\_

where

test0\_.id=?

t1

Hibernate:

select

test0\_.id as id0\_0\_,

test0\_.age as age0\_0\_,

test0\_.name as name0\_0\_

from

t\_test test0\_

where

test0\_.id=?

t0

t1

**由上可知：**

**Iterate使用了缓冲，出现N+1问题，先是发出查询所有的id，根据id查询出第一次结果，下一查询是根据缓存中id来查询出缓存中所有数据；注意只有当缓存中没有对应的id是，才会发出sql语言**

**List默认情况下没有缓存，可以人工设置**

**Iterate出现缓存只是在查询全部实体属性**

Query query1 = session.createQuery("select name from Test ");

Iterator<String> t1 = query1.iterate();

**for**(;t1.hasNext();){

String str =t1.next();

System.*out*.println(str);

}

Query query2 = session.createQuery("select name from Test");

Iterator<String> t2 = query2.iterate();

**for**(;t2.hasNext();){

String str =t2.next();

System.*out*.println(str);

}

Hibernate:

select

test0\_.name as col\_0\_0\_

from

t\_test test0\_

t0

t1

Hibernate:

select

test0\_.name as col\_0\_0\_

from

t\_test test0\_

t0

t1

发出两次sql语句，没有缓存

#### Flush：在批量使用时使用

**if**(i%10==0){

session.flush();

//flush有两个作用：1.清空Session，2.放入数据库

//就不要ts.commit();

}

1. 设置分页：

query.setMaxResults(2);

query.setFirstResult(0);

**注意：**

1. Hql：不能使用“\*”
2. 支持对象导航

user.group.name = ?

##### 增加：

用save方法就可以了

##### 删除：

user.setId(1);

// session.delete(user);//删除整个对象

Query query = *session*.createQuery("delete User where username=?");

query.setString(0, "index");

query.executeUpdate();

## 悲观锁

数据库的隔离级别：

悲观锁的实现，通常依赖于数据库机制，在整个过程中将数据锁定，其他任何用户不能读取或修改。适用场景：短事务。

使用重载的load(Class class,int id,LockMode.UPGRADE)方法

### 乐观锁

大多数基于数据版本记录机制，一般在数据库表中加入一个version字段，读取数据时将版本号一同读出，之后更新数据将版本号加一，如果提交数据时版本号小于或等于数据库中表中的版本号，则认为数据过期，否则更新。适合于长事务，并发性好。

在配置文件的<class optimistic-lock="version">标签上配置属性optimistic-lock="version"

在<id>标签后配置<version name="version">

## 一级缓存

一级缓存很短和Session的生命周期一致，session间不能共享缓存，一级缓存也称为Session级别的缓存或事务级缓存。

**一级缓存缓存实体对象**，**get与load方法使用缓存，iterate只有在查询结果为实体结果**集时才使用缓存，查询普通属性不使用缓存。

save同样使用缓存，**一级缓存不能取消，但是可以管理，如：session.clear();sesson.evict();**

常用优化：**在批量更新或保存时，每更新或保存多少条数据强制数据持久化，同时清理缓存，避免内存溢出。Session.flush();session.clear();**

### 二级缓存

经常被访问且不经常改动的对象应该放在二级缓存当中

使用第三方缓存产品EHCache

二级缓存也称为进程级缓存或SessionFactory级缓存，二级缓存可以被所有session共享。二级缓存的生命周期和SessionFactory的生命周期一致，SessionFactory管理二级缓存。

二级缓存的配置

1. 将ehcache.xml文件复制到src文件夹下。
2. 在hibernate.cfg.xml配置文件中配置，打开二级缓存，默认为打开。

<property name="hibernate.cache.use\_second\_level\_cache">true</property>

1. 指定产品提供商

<property name="hibernate.cache.use.provider\_class">

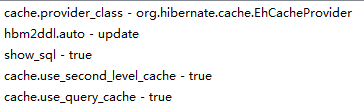
org.hibernate.cache.EhCacheProvider

</property>

1. 在实体配置文件中使用<cache usage="read-only">标签配置实体缓存策略

也可以在hibernate.cfg.xml中配置<class-cache class="实体类" usage="read-only">

@Cache load默认使用二级缓存，iterator默认使用二级缓存 list默认往二级缓存加数据但是查询不使用，query要使用二级缓存需要在<property name=”chache.use\_query\_cache”>true</propery> 然后调用Query.setCacheable(true)



**那个对象需要使用二级缓存：**

**就需要再实体类中写上：**

**@Cache(usage=CacheConcurrencyStrategy.*READ\_ONLY*)**

#### 查询缓存：

针对普通属性结果集的缓存，对实体对象的结果集只保存id，默认为关闭，打开查询缓存，在hibernate.cfg.xml中配置：

<property name="hibernate.cache.use\_query\_cache">true</property>

生命周期，当关联表发生修改，查询缓存的生命周期结束。

启用：query.setCacheable(true);

查询缓存只对query.list()起作用，对iterate()不起作用。

Query query = session.createQuery("");

query.setCacheable(**true**);

//查询中1.必须配置config cache.use\_query\_cache = true

// 2.必须启动,query.setCacheable(true);

Hibernate难点：

如果知道懒加载lazy

抓取策略：？？？

**多对多是必须使用懒加载（因为在查询的时候，使用缓存时，会导致其他实体表的数据查询出，浪费，所以只需要查询相应的表才会出现相应缓存，这就是懒加载的好处）**

**@ManyToMany(fetch=FetchType.*LAZY*)**

**对需要对象导航时使用**