# hibernate 学习笔记

前言hibernate 准备

1.视频说明

hibernate 网站: [www.hibernate.org](http://www.hibernate.org)

本视频版本： hibernate core 3.3.2 hibernate annotations 3.4.0

2日志框架 ：SLF4J

3需要的jar文件：

a\_) Hibernate JAR:

hibernate-distribution-3.3.2.GA/hibernate3.jar

hibernate-distribution-3.3.2.GA/lib/required/文件夹下jar（可选的必须的）

b\_) Annotation JAR

hibernate-annotations-3.4.0.GA/hibernate-annotations.jar

hibernate-annotations-3.4.0.GA/lib/ejb3-persistence.jar

hibernate-annotations-3.4.0.GA/lib/hibernate-commons-annotations.jar

c\_) SLF4J 实现 JAR (可选的日志系统):

slf4j-1.5.8/slf4j-nop-1.5.8.jar

d\_) Log4j: （slf4j 的log4j的实现）（练习中用到的日志系统）

log4j-1.2.11.jar

slf4j-1.5.8/slf4j-log4j12-1.5.8-sources.jar

e\_)二级缓存用到的jar

hibernate-distribution-3.3.2.GA\lib\optional\ehcache\ehcache-1.2.3.jar

commons-logging-1.1.1.jar

4 配置文件模版目录：

hibernate-distribution-3.3.2.GA\project\etc

5数据库版本： ysql-essential-5.0.22-win32.msi

6 JDBC驱动：mysql-connector-java-5.1.6-bin.jar

代码练习部分0

1 知识点：

基本配置

2 代码工程ES0000：

hibernate\_0000\_HelloWorld

3 详解：

a\_).导入：jre包（core库:hibernate3.jar lib文件夹下所有jar包）

b\_).导入mysql数据库驱动

c\_).导入hibernate.cfg.xml 配置文件到 src下

d\_).在hibernate.cfg.xml配置数据库 （参考 hibernate.properties 文件）

<hibernate-configuration>

<session-factory>

<!-- Database connection settings -->

<property name="connection.driver\_class">com.mysql.jdbc.Driver</property>

<property name="connection.url">jdbc:mysql://localhost/hibernate</property>

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

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

<!-- JDBC connection pool (use the built-in)

<property name="connection.pool\_size">1</property>-->

<!-- SQL dialect -->

<property name="dialect">org.hibernate.dialect.MySQLDialect</property> //方言

<!-- Enable Hibernate's automatic session context management

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

<!-- Disable the second-level cache -->

<property name="cache.provider\_class">org.hibernate.cache.NoCacheProvider</property>

<!-- Echo all executed SQL to stdout -->

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

<!-- Drop and re-create the database schema on startup -->

<property name="hbm2ddl.auto">create</property> //vaildata | update | create | create

<mapping resource="com/baishui/hibernate/model/User.hbm.xml"/>

</session-factory>

</hibernate-configuration>

e\_).配置:log4j.properties 便于调试，（log4j.properties 文件考入src下）

f\_).创建实体类User

g\_).配置实体映射文件（描述实体类 元数据）命名 User.hbm.xml

<hibernate-mapping package="com.baishui.hibernate.model">

<class name="User" table="User" >

<id name="id" column="id">

</id>

<property name="name"/>

<property name="age"/>

</class>

</hibernate-mapping>

注： Java 中的数据类型hibernate中会自动转换成数据库对应的类型，这是hibernate 自己的类型转换机制

h\_).在hibernate.cfg.xml 中 配置 对应的实体映射文件 加载配置

<mapping resource="com/baishui/hibernate/model/User.hbm.xml"/> //映射 实体映射文件位置，然后加载文件

i\_).创建 对象模型生成关系模型的工具类 UserTest

public class UserTest {

public static void main(String[] args) {

User user1 = new User();

user1.setId(1);

user1.setName("user1");

user1.setAge(9);

Configuration cfg = new Configuration().configure(); //根据hibernate.cfg.xml创建Configuration

SessionFactory sf = cfg.buildSessionFactory();

Session session = sf.openSession(); //打开链接

session.beginTransaction(); //开始事务

session.save(user1); //存储对象

session.getTransaction().commit(); //提交

session.close(); //关闭连接会话

sf.close(); //关闭连接工厂

}

}

（小技巧：关联源代码 选中方法 按F3， 选择文件系统， hibernate-3.2.0.ga.zip 一般都是zip包）

o\_).手动创建数据库（MySql）

create database hibernate;

user hibernate;

show tables;

create table user(id int primary key,name varchar(20),age int);

代码练习部分1

1知识点：

建立Annotation版本的 HelloWorld （支持JPA）（思想：标注 控制数据库）

2 代码工程ES0100：

hibernate\_0100\_HelloWorld

3 详解：

a\_).加入支持Annotation 的jar包

hibernate-annotations-3.4.0.GA/hibernate-annotations.jar

hibernate-annotations-3.4.0.GA/lib/ejb3-persistence.jar

hibernate-annotations-3.4.0.GA/lib/hibernate-commons-annotations.jar

b\_).创建Teacher 实体类

1.标注实体 ： @Entity (标识实体类 javax.persistence.Entity)

2.标注ID ： @Id （标识主键 在get方法上添加）

c\_).在hibernate.cfg.xml 中添加 实体映射的实体类说明

<mapping class="com.baishui.hibernate.model.Teacher"/>

4.创建hibernate会话

Teacher tr = new Teacher();

tr.setId(2);

tr.setName("tr1");

tr.setTitle("english");

Configuration cfg = new AnnotationConfiguration().configure(); // 必须要用AnnotationConfiguration

SessionFactory sf = cfg.buildSessionFactory();

Session session = sf.openSession();

session.beginTransaction(); //开始事务

session.save(tr); //存储对象

session.getTransaction().commit(); //提交

session.close(); //关闭连接会话

sf.close(); //关闭连接工厂

5.创建表

create table teacher(id int primary key,name varchar(20),title varchar(20));

（小技巧： 敲 @ 符号 提示：preferences->java->content assist 中设置 提示 . @）

代码练习部分2

1 知识点：

hibernate 模拟

2 代码工程ES0200：

hibernate\_0200\_OR\_Mapping\_Simulation

3 详解：----------------------------------

代码练习部分

1 知识点：

其他类型的 O/R映射框架

2 框架：

a\_). Hibernate b\_).toplink

c\_). Jbo d\_).ibatis

e\_)..JPA (意愿统一天下)

代码练习部分3

1 知识点：

hibernate 基础配置

2 代码工程ES0300：

hibernate\_0300\_BasicConfiguration

3 详解：

a\_).配置ddl

属性： <property name="hbm2ddl.auto">create</property>

值 ： vaildata | update | create | create-drop

含义： 数据类型验证 |表结构自动更新 | 自动创建表 | 关闭SessionFactory时drop 表

b\_). 加入 log4j log

a.添加log4j jar 文件 : log4j-1.2.11.jar

b.添加slf4j 转换jar : slf4j-1.5.8/slf4j-log4j12-1.5.8-sources.jar

c.添加log4j 配置文件 : log4j.properties

c\_). 加入Junit 4

a.添加Junit jar 文件 : junit-4.7.jar

b.创建SourcesFolder : test (测试代码跟文件夹)

d\_) .JUnit中的bug (有时出有时不出，用try catch 或 main 解决)

hibernate : hibernate.cfg.xml配置小错误 导致 SessionFactory 创建异常，但是不会报异常

（小技巧：myeclipce自动创建 junit 测试代码 ： 右键->JUnit Test Case ）

e\_) .关于sql 配置

<property name="show\_sql">true</property> //显示sql 语句

<property name="format\_sql ">true</property> // 格式化sql语句

f\_) .hibernate基础配置（Annotation 和xml两种不同情况的 配置）

(注：属性标注 写在get方法上面，写在属性上会破坏Java 的封装)

1表明与类名不同时（相同时什么都不用写）

a.Annotation: 添加 @table（name="\_teacher"） 标注

b.xml : <class name="Student" table="\_teacher">

2列名与属性名不同时（ 相同时 @Basic 或不写）

a.Annotation: @Column(name="\_name")

b.xml : <property name="name" column="\_name")/>

3当属性不想存入数据库时

a.Annotation: @ Transient (透明的)

b.xml : 不写对应的 property

4时间和日期的类型配置（默认会直接转换）

a.Annotation : @ Temporal(TemporalType.DATE) (控制数据的精度)

b.xml : 不写对应的 property

5枚举类型数据的配置

a.Annotation:

@Enumerated(EnumType.STRING) (STRING - varchar ORDINAL - Integer )

b.xml :特复杂

代码练习部分

1 知识点：

ID 生成策略

2 代码工程ES0400：

hibernate\_0400\_ID\_Generator

3 详解：

1\_)xml配置方法

<id name="id" column="id">

<generator class="native"></generator>

</id>

1. uuid (不会重复的一种算法 数据类型：String)

2. native (自动生成 auto increment 数据类型：int)

oracle : sequence

mysql : hilo

sqlserver : dentity

3.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。

4.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。

2\_)Annotation配置方法

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

（默认是AUTO,其他：SEQUENCE，IDENTITY,TABLE）

c\_)如果是oracle数据库时，要求指定不同的 Sequence 时，配置如下

1 .在实体类@ Entity 下添加标注

@SequenceGenerator(name="teacherSEQ",squenceName="teacher\_SEQ\_DB")

2.在@ Id 下添加标注

@GeneratedValue(strategy=GenerationType.SEQUENCE,generator="teacherSEQ")

d\_)配置 table\_Generator (优点跨数据库平台)

1 .在实体类@ Entity 下添加标注

@javax.persistence.TableGenerator(

name="Teacher\_Tab\_GEN",

table="GENERATOR\_Table",

pkColumnName="key",

valueColumnName="value",

pkColumnValue="TeacherID",

allocationSize=1 )

2.在@ Id 下添加标注

@GeneratedValue(strategy=GenerationType.TABLE,generator="Teacher\_Tab\_GEN")

(注：原理 select max(value)+1 from GENERATOR\_Table where key = TeacherID)

代码练习部分4-1

1 知识点：

联合主键

2 代码工程ES0410 ：

hibernate\_0410\_UniteID\_Generator

3 详解：

1\_)xml 配置实现

1：添加主键类（组件）implements java.io.Serializable (实现序列化)

2：添加逐渐属性 id name

3 : 重写 equals() 和 hashCode() 方法 实现 唯一

@Override

public boolean equals(Object obj) {

if(obj instanceof StudentPK){

StudentPK pk = (StudentPK)obj;

if(this.id == pk.getId() && this.name.equals(pk.getName())){

return true;

}

}

return false;

}

@Override

public int hashCode() {

return this.name.hashCode();

}

4 ：配置 xml

<composite-id name="pk" class="StudentPK">

<key-property name="id"></key-property>

<key-property name="name"></key-property>

</composite-id>

2\_)Annotation 配置实现

方法一：添加主键类（组件）implements java.io.Serializable

将组件类注解为 @Embeddable 并将组件的属性注解为 @Id

方法二： 添加主键类（组件）implements java.io.Serializable（常用）

直接将组建类主键属性 注解为 @EmbeddId

方法三：添加主键类（组件）implements java.io.Serializable（常用）

在组件中 定义属性，在类中 引入get set方法

在 类上 标注 @IdClass(TeacherPK.class)

在联合主键 get方法上 分别 注解： @Id

（注：load 时会明显看出 IdClass 作用）

注意重写 equals() 和 hashCode() 方法 实现 唯一

代码练习部分5

1 知识点：

核心开发接口

2 代码工程ES0500 ：

hibernate\_0500\_CoreAPI

3 详解：

1. org.hibernate.cfg.AnnotationConfiguration;

new AnnotationConfiguration().configure() //根hibernate.cfg.xml创Configuration

2. org.hibernate.SessionFactory;

相当于创建数据库 连接工厂

a\_) Session session = sf.openSession(); //每次打开新连接 必须close

b\_) Session session = sessionFactory.getCurrentSession(); //如果有已打开的session取出，否则打开新的 不必用close

getCurrentSession 与 openSession() 的区别

1 getCurrentSession创建的session会和绑定到当前线程,而openSession不会。

2 getCurrentSession创建的线程会在事务回滚或事物提交后自动关闭,而openSession必须手动关闭

这里getCurrentSession本地事务(本地事务:jdbc)时 要在配置文件里进行如下设置

\* 如果使用的是本地事务（jdbc事务）

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

\* 如果使用的是全局事务（jta事务）

\*手动管理事物（managed）

\*自定义事务管理（customer.Class）

3.org.hibernate.Session

三种状态 ： Transaction Persistent Detached

a) translent 内存中有 ，缓存没有 ,没ID

b) Persistent 内存中有 ，缓存有 ，数据库有

c) Detached 内存中有 ，缓存没有 ，数据库有

三种状态的区别在于：

1 有没有ID

2 ID在数据库中有没有

3 在内存中有没有（session缓存）

Session方法（CRUB）

1 save()

2 delete()

3 load() :返回的是代理对象

4 get() ：得到的是真实对象

5 update hibernate 根据 id 更新，默认更新所有列值，注意 同一个session

不更新的字段配置：

a)-Annotation : 在对应的 属性get上 添加标注 Column(updateble=false)

b)-xml : <perprotory update="true\false">

只更新修改的属性值

a)-Annotation : 建议使用 HQL EJBQL 语句

b)-xml : <class dynamic-update="true">

6 saveOrUpdate 根据session缓存中对象的数据ID 判断是save or update

7 clear 清除session缓存

（注：无论是load 还是 get 都是先查缓存（一级缓存），再去查数据库）

8 flush() 强制 执行从 内存到数据库 的同步

9 merge() 整合

4.org.hibernate.tool.hbm2ddl.SchemaExport

数据库操作 对象 CreateTable

SchemaExport se = new SchemaExport(new AnnotationConfiguration().configure());

se.create(false, false); (参数一：是否输出执行语句，参数二：是否执行数据库操作)

5.Query （HQL EJBQL）

代码练习部分

1 知识点：

实体映射关系模型

2关系映射：

1\_）.对象之间的关系

a)\_一对一（单向双向 ）

b)\_一对多（单向双向 ）

c)\_多对一（单向双向 ）

d)\_多对多（单向双向）

2\_）.每种关联的具体实现

（注：主键和外键关联的区别，主键关联是指 在主键上添加外键约束，

外键关联 是指 添加一个外键的字段，用于外键约束;

单向和双向的区别：数据库都不会 有任何差别，只是在java 程序中

建立了对象之间的 单向或双向的关联）

代码练习部分6

1 知识点：

一对一——单向——外键——关联： （很少用）

2代码工程ES0600：

hibernate\_0600\_one2one\_uni\_fk

3 详解：

1\_)Annotation:

在实体中对象属性get添加 标注

@OneToOne

@JoinColumn(name="wifeId")

2\_)xml:

<many-to-one name="student" class="Student" column="studentid" unique="true"/>

（注： unique="true" 指定 唯一）

代码练习部分7

1 知识点：

一对一——双向——外键——关联： (不常用)

2代码工程ES0700：

hibernate\_0700\_one2one\_bi\_fk

3 详解：

1\_）Annotation:

在两个实体类中都添加双向的对象属性

在实体Husband中对象属性get添加标注

@OneToOne

@JoinColumn(name="wifeId")

在实体Wife中对象属性get添加

@OneToOne (mappby="wife"),添加JoinColumn(name="hand\_id")标注

（注：双向外键 必填mappby，否则 两个表都会有外键 不合理）

2\_）xml:

在StudentIDCard.hbm.xml

<many-to-one name="student" class="Student" column="studentid" unique="true"/>

（注： unique="true" 指定 唯一）

在Student.hbm.xml

<one-to-one name="card" property-ref="student"></one-to-one>

（注：Student 被 StudentIDCard 中 student 设置，否则 两个表都会有外键 不合理）

代码练习部分8

1 知识点：

一对一——单向——主键——关联： (不重要)

2代码工程ES0800：

ES hibernate\_0800\_one2one\_uni\_pk

3 详解：

1\_）Annotation:

在实体Husband中对象属性get添加 标注

@OneToOne

@PrimaryKeyJoinColumn

(注： hibernate annotation不会生成两表的主键关联bug ，xml 没有详解)

2\_）xml:

在StudentIDCard.hbm.xml

主键声称策略 for

<id name="id">

<generator class="foreign"> //当程序存储时自动 id 会 去 student 去取，实现主键一致

<param name="property">student</param>

</generator>

</id>

<one-to-one name="student" constrained="true" />

（注： constrained="true" 两表添加外键关联）

Student.hbm.xml不需要配置其他

代码练习部分9

1 知识点：

一对一——双向——主键——关联： (了解)

2代码工程ES0900：

ES hibernate\_0900\_one2one\_bi\_pk

3 详解：

1\_)Annotation:

在实体Husband中对象属性get添加标注

@OneToOne

@PrimaryKeyJoinColumn

(注： hibernate annotation不会生成两表的主键关联bug ，xml 没有详解)

2\_)xml:

在StudentIDCard.hbm.xml

主键声称策略 foreign

<id name="id">

<generator class="foreign"> //当程序存储时自动 id 会 去 student 去取，实现主键一致

<param name="property">student</param>

</generator>

</id>

<one-to-one name="student" constrained="true" />

(注： constrained="true" 两表添加外键关联）

在Student.hbm.xml 不需要配置其他

代码练习部分10

1 知识点：

一对一——单向——联合主键的外键——关联:(了解)

2代码工程ES1000：

ES hibernate\_1000\_one2one\_uni\_UniteID\_fk

3 详解：

1\_)Annotation:

添加联合主键类 WifePk 属性 id name

Wife实体类标注 @ IdClass(WifePk.class);并在主键get 标注 @Id

Husband实体类 在对应 对象getWife() 上标注

@OneToOne

@JoinColumns(

{ @JoinColumn(name="wifeId",referencedColumnName="id"),

@JoinColumn(name="wifeName",referencedColumnName="name")

})

2\_)xml:

在StudentIDCard.hbm.xml

<id name="id">

<generator class="native"></generator>

</id>

<property name="studentName"/>

<property name="num"/>

<many-to-one name="student" unique="true">

<column name="id"></column>

<column name="studentName"></column>

</many-to-one>

在Student.hbm.xml

<composite-id name="pk" class="StudentPk">

<key-property name="id"></key-property>

<key-property name="name"></key-property>

</composite-id>

<property name="name"/>

(注： 如果是双向 添加one-to-one ES hibernate\_1010\_one2one\_bi\_UniteID\_fk)

<one-to-one name="studentIdCard" property-ref="student"></one-to-one>

代码练习部分11

1 知识点：

一对一 另一种解决办法（表合并） -------组件映射

2代码工程ES1100：

ES hibernate\_1100\_component

3 详解：

1\_)Annotation:

Husband 类中包含 Wife ,Wife 就不用标注实体类

在Husband 类中 getWife 上 添加标注 @ @Embedded 含义包含

2\_)xml:

在StudentIDCard.hbm.xml

<id name="id">

<generator class="native"></generator>

</id>

<property name="studentName"/>

<property name="num"/>

<many-to-one name="student" unique="true">

<column name="id"></column>

<column name="studentName"></column>

</many-to-one>

在Student.hbm.xml

<composite-id name="pk" class="StudentPk">

<key-property name="id"></key-property>

<key-property name="name"></key-property>

</composite-id>

<property name="name"/>

<one-to-one name="studentIdCard" property-ref="student"></one-to-one>

代码练习部分12

1 知识点：

多对一——单向——外键——关联： Group 1 ----N User

2代码工程ES1200：

ES hibernate\_1200\_many2one\_uni\_fk （常用）

3 详解：

1\_)Annotation:

在实体类User getGroup() 上添加 @ManyToOne 标注

2\_)xml:

在User.hbm.xml

<many-to-one name="group" colunm=" groupId"/>

代码练习部分13

1 知识点：

一对多——单向——外键——关联： Group 1 ----N User

2代码工程ES1300：

ES hibernate\_1300\_one2many\_uni\_fk （常用）

3 详解：

1\_)Annotation:

Group实体类中Set集合属性 <User>getUsers() 上添加 标注

@OneToMany

@JoinColnum(name="groupId")

（注： 在User表添加外键 ; @ 注：不添加 joincolnum标注时 默认会建立中间表)

2\_)xml:

在Group.hbm.xml

<set name="users" >

<key column="groupId"></key> （@注： 在User表添加外键列 groupId）

<one-to-many class="User"/> （@注： 指定说明被外键关联的对象）

</set>

代码练习部分14

1 知识点：

一对多(多对一)——双向——外键——关联： Group 1 ----N User

2代码工程ES1400：

ES hibernate\_1400\_one2many\_bi\_fk （最常用）

3 详解：

1\_)Annotation:

两面都添加对方的属性

Group实体类中Set集合属性 <User>getUsers() 上添加 标注

@OneToMany(mappedBy="group") （@注： 主要根据Group 表映射 (双向必填)

User实体类中属性getGroup()上添加 标注

@ManyToOne()

2\_)xml:

在Group.hbm.xml

<set name="users" >

<key column="groupId"></key> （@注： 在User表添加外键列 groupId）

<one-to-many class="User"/> （@注： 指定说明被外键关联的对象）

</set>

在User.hbm.xml

<many-to-one name="group" column="groupId"></many-to-one>

（@注： 两个对应 column必须保持一致，否则会创建两个外键列）

代码练习部分15

1 知识点：

多对多——单向——外键——关联： Teacher N ----N Student

2代码工程ES1500：

ES hibernate\_1500\_many2many\_uni\_fk （很少用）

3 详解：

1\_)Annotation:

Teacher 实体类中 添加 Set Students

在getStudent() 上添加 标注：

@ManyToMany

@JoinTable( （@注： 创建中间表 时 制定 表名和列名

name="joinTable",

joinColumns={ @JoinColumn(name="teacher\_id")},

inverseJoinColumns={ @JoinColumn(name="student\_id")}

)

单向Student 不用添加 teacher 属性

2\_)xml:

在Teacher.hbm.xml

<set name="students" table="joinTable"> （@注： 创建中间表 时 制定 表名和列名

<key column="teacher\_id"></key>

<many-to-many class="Student" column="student\_id"/>

</set>

在Student.hbm.xml 单向不需要特殊设置

代码练习部分16

1 知识点：

多对多)——双向——外键——关联： Teacher N ----N Student

2代码工程ES1600：

ES hibernate\_1600\_many2many\_bi\_fk （相当少用）

3 详解：

1\_)Annotation:

Teacher 实体类中 添加 Set Students

在getStudent() 上添加 标注：

@ManyToMany

@JoinTable( （@注： 创建中间表 时 制定 表名和列名

name="joinTable",

joinColumns={ @JoinColumn(name="teacher\_id")},

inverseJoinColumns={ @JoinColumn(name="student\_id")}

)

双向Student添加 set teacher 属性

@ManyToMany(mappedBy="students")

2\_)xml:

在Teacher.hbm.xml

<set name="students" table="joinTable"> （@注： 创建中间表 时 制定 表名和列名）

<key column="teacher\_id"></key>

<many-to-many class="Student" column="student\_id"/>

</set>

在Student.hbm.xml

<set name="teachers" table="xml\_joinTable" > （@注： 表名必须保持一致）

<key column="student\_id"></key>

<many-to-many class="Teacher" column="teacher\_id"/>

</set>

代码练习部分17

1 知识点：

一对多（多对一）——双向——级联关系映射： Group 1 ----N User

2代码工程ES1700：

ES hibernate\_1700\_one2many\_many2one\_bi\_crud

3 详解：

1\_)Annotation:

1．Group实体类标注

@OneToMany(mappedBy="group"，cascade={CascadeType.ALL},fetch=FetchType.LAZY)

2．User实体类标注

@ManyToOne(cascade={CascadeType.ALL}, fetch=FetchType.EAGER)

2\_)xml:

1．在Group.hbm.xml

<set name="users" cascade="all" inverse=”true” fetch=”join” ) （@注： 级联属性）

<key column="groupId"></key>

<one-to-many class="User"/>

</set>

2．User.hbm.xml

<many-to-one name="group" column="groupId" cascade="all" fetch=”join”/>

（注： 在级联增删改时cascade 级联属性，级联读取 fetch属性；并且设定 mappby）

Fetch：一的这一方默认是LAZY ; 多的这一方默认是EAGER (根据SQL区分)

代码练习部分18

1 知识点：

集合映射： Group 1 ----N User

2代码工程ES1800：

hibernate\_1800\_Collection\_Mapping

3 详解：

1\_)Annotation (List集合) 注（与set 基本一样 但是有序: @OraderBy(“username ASC”)）

private Set<User> users = new HashSet(User)();

标注：@OneToMany(mappedBy="group"，cascade={CascadeType.ALL})

@OraderBy(“username ASC”)

2\_)Annotation (Map集合)

private Map<Integer,User> users = new HashMap<Integer,User>();

标注：@OneToMany(mappedBy="group"，cascade={CascadeType.ALL})

@MapKey(name=”id”)

注： map 集合取值的方法

for(Map.Entry<Integer, User> entry : group.getUsers().entrySet()){

System.out.println(entry.getKey()+"--"+entry.getValue().getUserName());

}

代码练习部分19

1 知识点：

继承映射 1 Single\_Table ： Person -------( Student Teacher)

2代码工程ES1900：

hibernate\_1900\_Inheritence\_Mapping\_Single\_Table

3 详解：

1\_）Annotation

Persion 实体类中：

@Entity

@Inheritance(strategy=InheritanceType.SINGLE\_TABLE)

@DiscriminatorColumn(name="discriminator",discriminatorType=DiscriminatorType.STRING)

@DiscriminatorValue("persion")

Student 实体类中：

@Entity

@DiscriminatorValue("student")

Teacher 实体类中：

@Entity

@DiscriminatorValue("teacher")

代码练习部分20

1 知识点：

继承映射 2 Table\_per\_class ： Person -------( Student Teacher)

2代码工程ES2000：

hibernate\_2000\_Inheritence\_Mapping\_Table\_Per\_Class

3 详解：

1\_）Annotation

Persion 实体类中：

@Entity

@Inheritance(strategy=InheritanceType.TABLE\_PER\_CLASS)

@TableGenerator(

name="t\_gen",

table="t\_gen\_table",

pkColumnName="t\_pk",

valueColumnName="t\_value",

pkColumnValue="person\_pk",

initialValue=1,

allocationSize=1

)

@Id

@GeneratedValue(generator="t\_gen",strategy=GenerationType.TABLE)

Student 实体类中：

@Entity

Teacher 实体类中：

@Entity

代码练习部分21

1 知识点：

继承映射 3 Joined ： Person -------( Student Teacher)

2代码工程ES2100：

hibernate\_2100\_Inheritence\_Mapping\_Table\_Per\_Class

3 详解：

1\_）Annotation

Persion 实体类中：

@Entity

@Inheritance(strategy=InheritanceType.JOINED)

Student 实体类中：

@Entity

Teacher 实体类中：

@Entity

代码练习部分22

1 知识点：

Tree作业练习 ： Tree ----( Parent 1---N Children )

2代码工程ES2200：

hibernate\_2200\_Tree

3 详解：

1.声明成员变量

private int id;

private String name;

private Set<Org> children = new HashSet<Org>();

private Org parent;

2.设置关系模型

@OneToMany(mappedBy="parent",cascade={CascadeType.ALL},fetch=FetchType.EAGER)

public Set<Org> getChildren() {

return children;

}

@ManyToOne

@JoinColumn(name="parent\_id")

public Org getParent() {

return parent;

}

3.存储 (设置 一对多 级联cascade属性)

@OneToMany(mappedBy="parent",cascade={CascadeType.ALL},fetch=FetchTyp.EAGER)

4读取 （设置一对多 级联fetch属性）

（注 tree的递归算法； 注意tree非常小时 可以用 EAGER 否则影响效率）

代码练习部分23

1 知识点：

学生-课程-分数 关系映射练习

2代码工程ES2300：

hibernate\_2300\_Student\_Course\_Score

3 详解：

1．课程实体类

private int id;

private String name;

@Id

@GeneratedValue

public int getId() {

return id;

}

2.分数实体类 (标注 @Table(name=”score”))

private int id;

private int score;

private Student student;

private Course course;

@Id

@GeneratedValue

public int getId() {

return id;

}

@ManyToOne

@JoinColumn(name="student\_id")

public Student getStudent() {

return student;

}

@ManyToOne

@JoinColumn(name="course\_id")

public Course getCourse() {

return course;

}

3.学生实体类

通过 中间表连接 学生 – 课程，表名为：score 表，亦为 分数表

private int id;

private String name;

private Set<Course> courses = new HashSet<Course>();

@Id

@GeneratedValue

public int getId() {

return id;

}

@ManyToMany

@JoinTable(name="score",

joinColumns=@JoinColumn(name="student\_id", referencedColumnName="id"),

inverseJoinColumns=@JoinColumn(name="course\_id", referencedColumnName="id")

)

public Set<Course> getCourses() {

return courses;

}

（注：hibernate 小bug ,创建中间表时把 student\_id 和 course\_id 作为主键了，这样是不对的，所以手动修改表）

Create table score(

id int primary key auto\_increment,

student\_id int references student(id),

course\_id int references course(id),

score int

)

代码练习部分24

1 知识点：

Hibernate 查询（Query　Language）

2 HQL vs EJBQL语言

1\_) NativeSQL (本地语言)

2\_)HQL (hibernate自定义语言)

3\_)EJBQL （hibernate子集）

4\_)QBC (Query By Cretira)

5\_)QBE (Query By Example)

3 代码工程

1\_）: hibernate\_2400\_QL1

2\_）: hibernate\_2500\_QL2

3\_）: hibernate\_2600\_QBC

//标准约束

Criteria c = session.createCriteria(Topic.class) //相当于HQL from Topic t

// .add(Restrictions.gt("id", 1)) //greater than id

// .add(Restrictions.lt("id", 2)) //little than id

// .add(Restrictions.like("title", "%2"))

.add(Restrictions.between("id", 4, 6))

.createCriteria("category")

.add(Restrictions.between("id", 0, 2))

;

//DetachedCriteria

for (Object o: c.list()) {

Topic t = (Topic)o;

System.out.println(t.getTitle()+"-"+t.getCategory().getPlate());

}

4\_）: hibernate\_2700\_QBE

@Test

/\*\*

\* Query By Criteria Query By Example

\*/

public void testQBE(){

Session session= sessionFactory.getCurrentSession();

session.beginTransaction();

Topic tExample = new Topic();

tExample.setTitle("topic1");

Example e = Example.create(tExample)

.ignoreCase().enableLike();

Criteria c = session.createCriteria(Topic.class)

/\*.add(Restrictions.gt("id", 2))

.add(Restrictions.lt("id", 8))\*/

.add(e);

;

for (Object o: c.list()) {

Topic t = (Topic)o;

System.out.println(t.getTitle()+"-"+t.getCategory().getPlate());

}

session.getTransaction().commit();

}

代码练习部分28

1 知识点：

Hibernate 1+N 详解

2代码工程ES2800：

hibernate\_2800\_Hibernate1+N

3 详解：

1．课程实体类

/\*\*

\* 1+N 详解 注 Criteria 自动解决详解

\* ManyToOne 中 FetchType 默认 为 EAGER

\* 取Topic 是 默认 关联查询 Category 对应的对象

\*

\* 第一种解决方法： FetchType设置为LAZY --test\_Nquery1

\* 第二种解决方法： join fetch --test\_Nquery2

\* 第三种解决方法： 在Category使用@BatchSize(size=5) --test\_Nquery3 执行效率高

\*\*/

代码练习部分29

1 知识点：

Hibernate List 和 Iterator 区别

2代码工程ES2900：

hibernate\_2900\_list\_iterate

3 详解：

/\*\* list 遍历两次时 第一次从数据库查询 第二次也从数据库查询 \*/

String hql = "from Topic t";

Query q = session.createQuery(hql);

List<Topic> topics = q.list();

for (Topic t: topics) { System.out.println(t.getTitle());}

List<Topic> topics2 = q.list();

for (Topic t: topics2) {System.out.println(t.getTitle());

}

/\*\*Iterator 遍历两次时 第一次从数据库查询 第二次从缓存里查询 \*/

String hql = "from Topic t";

Query q = session.createQuery(hql);

Iterator<Topic> topics = q.iterate();

while(topics.hasNext()) {

Topic t = topics.next();

System.out.println(t.getTitle());

}

Iterator<Topic> topics2 = q.iterate();

while(topics2.hasNext()) {

Topic t = topics2.next();

System.out.println(t.getTitle());

}

代码练习部分30

1 知识点：

二级缓存适合：1\_\_）不是同一个session 2\_)经常被访问 3\_）改动不平凡 4\_) 数量有限

2代码工程ES3000：

hibernate\_3000\_3KingOf\_Cache

3 详解：

1\_) 配置 hibernate.hbm.xml 二级缓存

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

<property name="cache.provider\_class">org.hibernate.cache.EhCacheProvider</property>

2\_) 需要ehcache 的配置文件ehcache.xml

文件模版路径：hibernate-distribution-3.3.2.GA\hibernate-distribution-3.3.2.GA\project\etc

文件配置详解:

<defaultCache

maxElementsInMemory="10000" //最大缓存多少个对象

eternal="false" //永远生存

timeToIdleSeconds="120" //缓存中老对象无访问失效时间120秒

timeToLiveSeconds="120" //缓存中老对象生存的时间120秒

overflowToDisk="true" //是否在内存溢出时，将对象放入硬盘

/>

<diskStore path="java.io.tmpdir"/> //设置硬盘存储临时目录

3\_) ehcache jar文件 和 commons-logging-1.1.1-bin.zip

ehcache jar文件路径：

hibernate-distribution-3.3.2.GA\hibernate-distribution-3.3.2.GA\lib\optional\ehcache

commons-logging-1.1.1-bin.zip (ehcache 的日志系统)需要下载解压

commons-logging-1.1.1.jar

4\_) 在需要存入二级缓存中的实体类上 标注

@Cache(usage=CacheConcurrencyStrategy.READ\_WRITE

// ,region="sampleCache1") //region 属性指定二级缓存使用自定义配置

注： load,iterator默认使用二级缓存，list默认往二级缓存里加数据，查询的时，除非语句一样否则不会使用二级缓存

代码练习部分31

1 知识点：

查询缓存

2代码工程ES3100：

hibernate\_3000\_3KingOf\_Cache

3 详解：

1\_) 打开 hibernate.hbm.xml 二级缓存 和查询缓存配置

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

<property name="cache.provider\_class">org.hibernate.cache.EhCacheProvider</property>

<property name="cache.use\_query\_cache">true</property> //查询缓存

2\_) 程序中指明使用二级缓存

query.setCacheable(true); //设置使用 二级缓存 否则发出两条查询语句

3\_) 缓存算法

A. LRU ------- least recently used （替换最少被使用的对象）

B LFU--------- least frequently used（替换最近不常被使用的对象）

C FIFO---------first ln frist out （替换最早被使用的对象）

注意设置配置文件ehcache.xml ：

memoryStoreEvictionPolicy=”LRU”

代码练习部分32

1 知识点：

事务的并发处理（hibernate l 悲观锁）

解决 hibernate.connection.isolation=2时出现repeatable read 详解

2代码工程ES3200：

hibernate\_3100\_Hibernate\_Concurrency\_Pessimistic\_Lock

3 详解：

事务处理依赖数据库的 事务处理级别

1 read-uncommitted --1-- 可以读取未提交的

2. read-committed --2--读取提交之后的

3.repeatable-read --4--重复读取

4.serializable --8--禁止多线程

@Test

/\*\*

\* 加悲观锁

\*/

public void testPessimisticLock(){

Session session = sessionFactory.getCurrentSession();

session.beginTransaction();

Accmount acc =(Accmount) session.load(Accmount.class, 1 ,LockMode.UPGRADE);//加锁

//do some caculation

acc.setBalance(acc.getBalance()+10);

session.getTransaction().commit();

}

代码练习部分33

1 知识点：

事务的并发处理（hibernate l 乐观锁）

机制：有个标识字段更新过改变一下 标识的值，在执行update的同时就检索一下当前的值是否一致

2代码工程ES3300：

hibernate\_3100\_Hibernate\_Concurrency\_Pessimistic\_Lock

3 详解：

1 在实体类 添加 version 标注@version