## Hibernate的入门搭建

1. 创建项目导入依赖包

required文件中的包+mysql的驱动包

1. 加入配置文件

全局文件：数据源，映射文件，方言

|  |
| --- |
| <?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>  <session-factory>  <!-- 数据源 -->  <property name=*"connection.driver\_class"*>com.mysql.jdbc.Driver</property>  <property name=*"connection.url"*>jdbc:mysql://localhost:3306/hibdata</property>  <property name=*"connection.username"*>root</property>  <property name=*"connection.password"*>root</property>  <!-- SQL方言 -->  <property name=*"dialect"*>  org.hibernate.dialect.MySQLDialect  </property>    <!-- 控制台显示sql -->  <property name=*"show\_sql"*>true</property>  <!-- 格式化sql -->  <property name=*"format\_sql"*>true</property>    <!-- 管理映射文件 -->  <mapping resource=*"com/offcn/bean/News.hbm.xml"*/>  </session-factory>  </hibernate-configuration> |

映射文件：数据库表和实体以及表中列和实体属性的对应关系

|  |
| --- |
| <?xml version=*"1.0"*?>  <!DOCTYPE hibernate-mapping  SYSTEM  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd" >  <hibernate-mapping package=*"com.offcn.bean"*>  <!-- name:实体名，table:表名 -->  <class name=*"News"* table=*"news"*>  <!-- id:完成实体中和表中主键的映射  name:实体中主键的属性名  column:表中主键的列名 -->  <id name=*"nid"* column=*"nid"*>  <!-- 主键的增长方式 -->  <generator class=*"native"*/>  </id>  <!--property:完成实体中和表中非主键的映射  name:实体中的属性名  column:表中的列名  type:属性的类型-->  <property name=*"title"* column=*"title"* type=*"string"*/>  <property name=*"content"* column=*"content"* type=*"string"*/>  <property name=*"photo"* column=*"photo"* type=*"string"*/>  </class>  </hibernate-mapping> |

1. 解析配置文件

解析配置----SessionFactory-----Session--------curd

|  |
| --- |
| **package** com.offcn.bean;  **import** org.hibernate.Session;  **import** org.hibernate.SessionFactory;  **import** org.hibernate.cfg.Configuration;  **public** **class** App {  **public** **static** **void** main(String[] args) {  //解析全局配置文件  Configuration configure = **new** Configuration().configure("com/offcn/bean/hibernate.cfg.xml");  //获取SessionFactory会话工厂  SessionFactory sf = configure.buildSessionFactory();  //获取Session会话  Session session = sf.openSession();  //调用根据主键查询的方法  News news = session.get(News.**class**, 3);  System.***out***.println(news);  session.close();  }  } |

## Hibernate完成基本的CURD 操作

CURD:get,load,save,update,delete

|  |
| --- |
| **public** **class** App2 {    Session session;    @Before  **public** **void** setup() {  //解析全局配置文件  Configuration configure = **new** Configuration().configure("com/offcn/bean/hibernate.cfg.xml");  //获取SessionFactory会话工厂  SessionFactory sf = configure.buildSessionFactory();  //获取Session会话  session = sf.openSession();  }    /\*  \* get:当查询的数据不存在返回null，在调用get方法时会立即从数据库查询  \* load:当查询的数据不存会抛出异常，在调用load方法时不会立即从数据库查询，在使用非之间属性时才去从数据查询（延时）  \*/  @Test  **public** **void** getByIdget() {  News news = session.get(News.**class**, 3);  System.***out***.println(news.getNid());  System.***out***.println(news.getTitle());  }  @Test  **public** **void** getByIdload() {  News news = session.load(News.**class**, 3);  System.***out***.println(news.getNid());  System.***out***.println(news.getTitle());  }  //save方法返回值是最新插入数据的主键  @Test  **public** **void** save() {  News n=**new** News();  n.setTitle("标题");  n.setContent("北京垃圾分类");  n.setPhoto("no.gif");  session.save(n);  session.beginTransaction().commit();  }    //根据主键更新  @Test  **public** **void** update() {  News n=**new** News();  n.setNid(13);  n.setTitle("垃圾");  n.setContent("北京垃圾分类开始了吗");  n.setPhoto("no.gif");  session.update(n);  session.beginTransaction().commit();  }    //delete方法是根据逐渐删除  @Test  **public** **void** del() {  News n=**new** News();  n.setNid(13);  session.delete(n);  session.beginTransaction().commit();  }      } |

## Hibernate常用的两种查询方式

### Query查询

（query对象的api方法：list,uniqueResult,executeUpdate）

hql语句的语法：from 类名 where 属性名=值

|  |
| --- |
| @Test  **public** **void** getAll() {  String hql=" from News";  Query q = session.createQuery(hql);  List<News> nlist = q.list();  System.***out***.println(nlist);  }    //默认将查询的列放到一个一个数组里，而不是一个对象里  @Test  **public** **void** getAll1() {  String hql="select nid,title from News";  Query q = session.createQuery(hql);  List<Object[]> nlist = q.list();  System.***out***.println(nlist.get(0)[0]+"----"+nlist.get(0)[1]);  }    //如果要查询某些列，同时将这些列的值赋值到一个对象里，需要类的有参构造方法  @Test  **public** **void** getAll2() {  String hql="select new News(nid,title) from News";  Query q = session.createQuery(hql);  List<News> nlist = q.list();  System.***out***.println(nlist.get(0).getNid()+"----"+nlist.get(0).getTitle());  }    @Test  **public** **void** getById() {  String hql="select new News(nid,title) from News where nid=3";  Query q = session.createQuery(hql);  List<News> nlist = q.list();  System.***out***.println(nlist.get(0).getNid()+"----"+nlist.get(0).getTitle());  }    /\*  \* hql语句中占位符的使用  \* ？------q.setParameter(索引, 值)  \* :变量名---- q.setParameter(变量名, 值);  \*/  @Test  **public** **void** getById1() {  String hql="select new News(nid,title) from News where nid=? and title like ?";  Query q = session.createQuery(hql);  q.setParameter(0, 3);  q.setParameter(1, "%五%");  List<News> nlist = q.list();  System.***out***.println(nlist.get(0).getNid()+"----"+nlist.get(0).getTitle());  }    @Test  **public** **void** getById2() {  String hql="select new News(nid,title) from News where nid=:nnid and title like :ti";  Query q = session.createQuery(hql);  q.setParameter("nnid", 3);  q.setParameter("ti", "%五%");  List<News> nlist = q.list();  System.***out***.println(nlist.get(0).getNid()+"----"+nlist.get(0).getTitle());  }  //通过hql语句完成删除  @Test  **public** **void** geByTitle() {  String hql="delete from News where title=?";  Query q = session.createQuery(hql);  q.setParameter(0, "gffh");  **int** row=q.executeUpdate();  System.***out***.println(row);  session.beginTransaction().commit();  }  //通过hql语句完成更新  @Test  **public** **void** updByTitle() {  String hql="update News set content=? where title=?";  Query q = session.createQuery(hql);  q.setParameter(0, "的客户看的好附件是");  q.setParameter(1, "jintian");  **int** row=q.executeUpdate();  System.***out***.println(row);  session.beginTransaction().commit();  } |

### Criteria查询（list,uniqueResult）

|  |
| --- |
| @Test  **public** **void** getAllBycriteria() {  Criteria cc = session.createCriteria(News.**class**);  cc.add(Restrictions.*eq*("title", "标题"));  List<News> nlist = cc.list();  System.***out***.println(nlist);  }    @Test  **public** **void** getByIds() {  Criteria cc = session.createCriteria(News.**class**);  cc.add(Restrictions.*in*("nid", 1,3,4,5,6));  List<News> nlist = cc.list();  System.***out***.println(nlist);  }    @Test  **public** **void** getByTitleLike() {  Criteria cc = session.createCriteria(News.**class**);  cc.add(Restrictions.*like*("title", "%周五%"));  List<News> nlist = cc.list();  System.***out***.println(nlist);  }    @Test  **public** **void** getByTileAndNid() {  Criteria cc = session.createCriteria(News.**class**);  cc.add(Restrictions.*like*("title", "%jint%"));  cc.add(Restrictions.*idEq*(4));  News news = (News) cc.uniqueResult();  System.***out***.println(news);  } |

主键的增长方式：

identity：数据库有主键自增功能的数据库可用这种方式，由数据库完成主键的自增，主要用于mysql或者sqlServer.

sequence:主要用于支持序列功能的数据库，例如DB2，oracle.

native:若连接的是含有主键自增功能的数据库，那么由数据库完成主键的自增；如链接的是序列功能的数据库，由序列完成主键自增。

increment：主键的自增由Hibernate 框架完成。在内存中获取主键的最大值，加1（mysql,oracle）.

assigned:主键是由程序生产的。(mysql,oracle)

foreign：使用另外一个相关联的对象的主键作为该对象主键，主要用于一对一关系中。

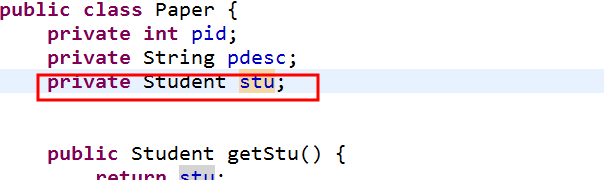
## 关系映射

### 一对一

学生和学生证（双向的一对一）

1. 将表与表之间的关系转换为实体与实体之间的关系

在一方加入另一方的实体类型属性以及属性的set,get方法。



2、在映射文件中完成自定义类型属性的映射

语法：

<one-to-one name="自定义类型的属性名" class="属性的类型"></one-to-one>

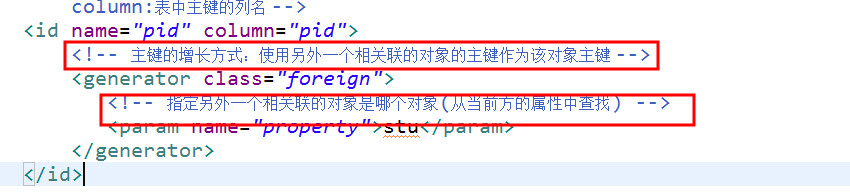
案例：

在Paper的映射文件中完成自定义类型属性stu的映射



注意：

1.在Paper的映射文件中指定主键是由哪一方的主键生产



1. 可以通过cascade="all" constrained="true" 设置级联操作

测试案例：

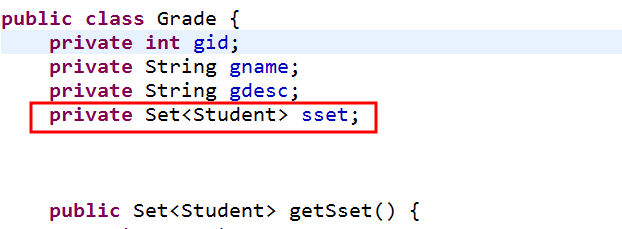
|  |
| --- |
| /\*  查询60001学生证以及对应学生的基本信息  \*/  @Test  **public** **void** getByIdget() {  Paper paper = session.get(Paper.**class**, 60001);  Student stu = paper.getStu();  System.***out***.println(paper.getPdesc()+"----"+stu.getSname());  }  //保存学生以及学生证的信息  @Test  **public** **void** saveStuAndPaper() {  Student stu=**new** Student();  stu.setSid(2019909002);  stu.setSname("张三丰");  stu.setSex("女");    Paper p=**new** Paper();  p.setPdesc("哈弗");  p.setStu(stu);//建立Student 和Paper的主外键关系,即将学生的主键作为Papepr的主键    //session.save(stu);  session.save(p);  session.beginTransaction().commit();    } |

### 一对多

1.将表与表之间的关系转换为实体与实体之间的关系

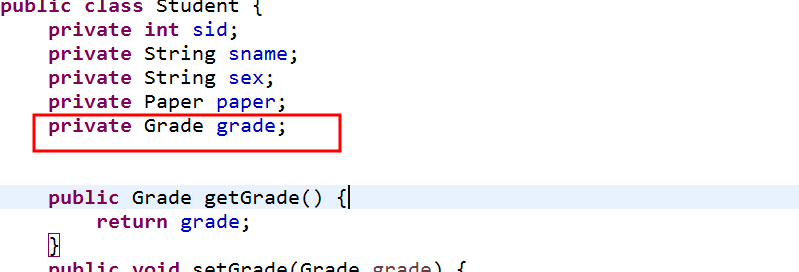
**在一的一方：**

加入多的一方的实体属性，例如在年级的实体类里加学生的集合Set；



**在多的一方**

加入一的一方的实体类型属性，例如在学生的实体类中加入年级的类型属性



2.在年级的映射文件完成set的映射

语法：

**一的一方**：

<set name="集合的属性名">

<key column="当前方和另一方关联的外键字段"></key>

<one-to-many class="关联的另一方的类型，即set集合的泛型"/>

</set>

**多的一方**：

<many-to-one name=”自定义类型的属性名” class=”属性的类型” column=”关联的外键”>

</many-to-one>

3.

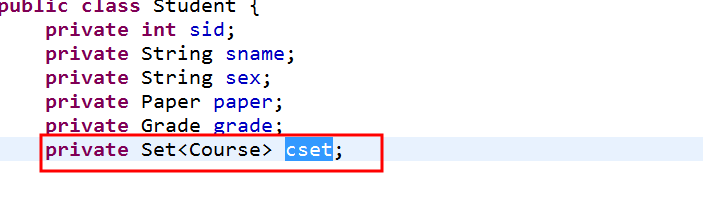
测试：查询某个年级的基本信息以及年级下的所有学生

|  |
| --- |
| ///////////////////////1-n////////////////////////////  //根据年级查询学生  @Test  **public** **void** getGraAndStusByGid() {  Grade grade = session.get(Grade.**class**, 4);  Set<Student> sset = grade.getSset();  System.***out***.println(grade.getGname()+"----"+grade.getGdesc());  **for**(Student s:sset) {  System.***out***.println(s.getSname()+"----"+s.getSex());  }  }    //根据学生查询年级  @Test  **public** **void** getGraAndStusBySid() {  Student student = session.get(Student.**class**, 1115);  Grade grade = student.getGrade();  System.***out***.println(student.getSname()+"----"+grade.getGname());  } |

### 多对多

1.将表与表之间的关系转换为实体与实体之间的关系

在其中一方加入另一方的set集合属性



2.在映射文件中完成集合的映射

语法：

<set name="集合属性名" table="中间表">

<key column="当前方和中间表关联的外键字段"></key>

<many-to-many column="另一方和中间表关联的外键字段" class="另一方的类型"></many-to-many>

</set>

案例：学生的映射文件

<set name="cset" table="sc">

<!--column：当前方和中间表关联的外键字段 -->

<key column="sid"></key>

<!--column：另一方和中间表关联的外键字段，class：另一方的类型 -->

<many-to-many column="cid" class="Course"></many-to-many>

</set>

测试：

|  |
| --- |
| //查询某个学生的基本信息以及学生选修的所有课程  @Test  **public** **void** getstuAndCouBySid() {  Student student = session.get(Student.**class**, 10008);  System.***out***.println(student.getSname());  Set<Course> cset = student.getCset();  **for**(Course c:cset) {  System.***out***.println(c.getCid()+"----"+c.getCname());  }  }    //查询某门课程的基本信息以及选修该课程的所有学生  @Test  **public** **void** getstuAndCouByid() {  Course cou = session.get(Course.**class**, 1022);  System.***out***.println(cou.getCname());  Set<Student> sset = cou.getSset();  **for**(Student s:sset) {  System.***out***.println(s.getSid()+"----"+s.getSname());  }  } |

## Hibernate中对象的三状态

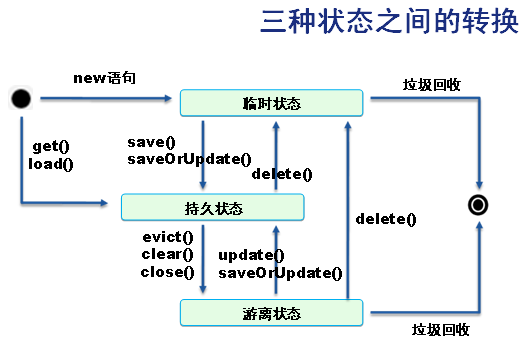
**临时状态**：采用new关键字创建的对象，该对象未与Session发生关联（未调用

Session的API）。也叫临时对象。临时状态的对象会被Java的垃圾回收机制回收。

**持久状态**：实体对象与Session发生关联（调用了Session的get、load、save、update等API）。也叫持久对象,持久状态的对象在被修改后是不需要调用更新方法，直接提交即可。

**游离状态**：原来是持久状态，后来脱离了Session的管理。如：Session被关闭，对象将从持久状态变为游离状态，同时垃圾回收机制可以回收掉，不再占用缓存空间了。

12.4.1.2 三种状态下对象的转换



## 缓存机制

一级缓存：不需要做任何设置，Session级别的。只能在使用自己的一级缓存，不同的Session对象不可以共享一级缓存。

|  |
| --- |
| @Test  **public** **void** getByIdget() {  News news = session.get(News.**class**, 4);  News news14 = session.get(News.**class**, 14);  System.***out***.println("第一次查询"+news);  System.***out***.println("第一次查询"+news14);    //session.clear();//清除session中的所有缓存  session.evict(news);//清除session中的某一个缓存数据    News news2 = session.get(News.**class**, 4);  News news214 = session.get(News.**class**, 14);  System.***out***.println("第二次查询"+news2);  System.***out***.println("第二次查询"+news214);  } |

二级缓存：Sessionfactory级别的。由同一个SessionFactory创建的多个不同的Session对象，可以共享二级缓存区域的数据。

配置：

1. 缓存的依赖包

D:\4\day03\hibernate-release-5.1.6.Final\lib\optional\ehcache



1. 在全局配置文件设置二级缓存的类型

|  |
| --- |
| <!-- 设置二级缓存 -->  <property name=*"hibernate.cache.region.factory\_class"*>  org.hibernate.cache.ehcache.EhCacheRegionFactory  </property> |

1. 在映射文件设置二级缓存的只读性

