**第四部分：Hibernate**

**hibernate资源配置：**

1、hibernate-release-4.3.11.Final\lib\required\下jar都添加到项目类加载路径WEB-INF\lib\下，Connector.J 5.1\mysql-connector-java.jar驱动也添加到项目类加载路径，

2、src下编写javabean+持久化注解表示 javabean类和数据库表对应关系，然后需要添加src下hibernate.cfg.xml填写配置信息：JDBC连接的基本信息,持久化对象写进去。

在.xml中不推荐使用DriverManger（和以往一样编写java语言连接数据库），而使用C3P0数据源（需要连接数据库时取连接，不需要时归还，而不是关闭，提高数据库效率），需要把hibernate-release-4.3.11.Final\lib\optional\c3p0下所有jar文件放在WEB-INF\lib\下。

3、最后编写类 获取Session开启事务，然后操作javabean，再关闭Session即可。

Ps：在eclipse中可以下载hibernate的插件，这样创建cfg.xml等配置就很方便。

部署到tomcat或者打包时是在WEB-INF\lib下找包， 而eclipse中运行是在src同目录的library下找包，但jar放在WEB-INF\lib下时，eclipse自动添加到library下，如果想要自己的User library包能放在WEB-INF\lib下，这需要配置一下Deployment。

**持久化对象：**

要求：无参构造器、提供一个标识属性id、javabean规范、不适用final类

瞬态: News n=new News();还没被session关联

持久化: session.save(n) //立即执行sql语句 session.persist(n)//事务外部调用时并不立即转化，底层insert语句，返回Serializable即主键id值；News n=session.load(News.class,pk)||get()c//get立刻访问数据库，load加载一个代理，直到调用get方法访问数据库，都是底层select语句；n.setTitle()则会在session.flush()（将会话中的缓存都写入数据库）写入之前执行，底层是update语句。

托管: session.close();n.setTitle();session2.update(n)||. SaveOrupdate (n);

删除：session.delete(n);对应的数据库记录也被删除了。

**持久化注解：代替过时的xml配置**

1.类级别注解

@Entity     映射实体类

@Table    映射数句库表

2.属性级别注解

@Id     映射生成主键

@GeneratedValue(strategy=GenerationType,generator="") - 可选，用于定义主键生成策略

@Column      可将属性映射到列，使用该注解来覆盖默认值，@Column描述了数据库表中，该字段的详细定义，这对于根据 JPA 注解生成数据库表结构的工具非常有作用。

@Transient     不会被持久化

@Formula 用sql语句(select)的结果计算当前属性值,不会被保存到数据列

@Generated bean对象的属性由数据库生成

@Basic - 用于声明属性的存取策略：

@Basic(fetch=FetchType.EAGER)   即时获取（默认的存取策略）

@Basic(fetch=FetchType.LAZY)    延迟获取

@Temporal – bean中Date类和数据库中类型对应：

@Temporal(TemporalType=DATE)       日期

@Temporal(TemporalType=TIME)       时间

@Temporal(TemporalType=TIMESTAMP) 两者兼具

@Order(sql) 在查询此属性时总会按照order顺序出现，但在数据库中没有排序。

@JoinColumn (name=””,referenced=””)- name是外键列名称，到底在主表还是从表产生外键看情况。

3.无注解属性的默认值

如果属性为单一类型,则映射为@Basic，

否则,如果属性对应的类型定义了@Embeddable注解,则映射为@Embedded(两者任用一种即可，@Embedded配合@AttributeOverride可以映射对应列)，

否则,如果属性对应的类型实现了Serializable, 则属性被映射为@Basic并在一个列中保存该对象的serialized版本，

否则,如果该属性的类型为java.sql.Clob 或 java.sql.Blob,则作为@Lob并映射到适当的LobType.(如果显示用@Lob修饰属性，则数据库中会有相应的LobType列)

4.集合属性:都是保存在另一张表中，然后外键关联，删除对象时，集合属性也会自动删除（手动sql删除需要先删除集合属性表，再删除对象）。推荐采用代理模式即延迟加载。List和Map比Set效率高（数据库中set总是需要遍历才能找到）。

4.1 普通基本类型

List和数组一样操作：总以外键列和索引列作为主键

@ElementCollection(targetClass=String.class)

@CollectionTable(name="school\_inf",joinColumns=@JoinColumn(name="id")) //外键默认参照当前实体主键

@Column(name="school\_name")

@OrderColumn(name="school\_order")

private List<String> schools=new ArrayList<>();

Set只需要去掉OrderColumn注解即可：以外键列和非空列作为主键

Map增加一个@MapKeyColumn(name=””)和@MapKeyClass(String.class)表示key，@Column和@ElementCollection(targetClass=String.class)是value：以外键列和key列作为主键

若是使用有序集合SortedMap等,需要@SortNatural.

4.2 复合属性类型：

删除@Column()，因为复合类型无法对应到一列，然后使用@Embeddable修饰复合类。若是复合类作为Map中key，这需要重写equals（）和hashcode（），不是为了解决数据库主键问题（数据库总是按照基本类型值相等判断），而是Map问题，使得Map添加和数据库标准相同。

5.关联映射关系:单向和双向的不同在于bean中是否有对方的变量，而不在于数据库。建立外键时，依赖的主键必须先存在于数据库中或者写cascade=CascadeType.ALL让底层自动先建立依赖的表数据。

1、@ManyToOne - 表示多对一的映射，该注解标注的属性通常是数据库表的外键。

@ManyToOne(targetEntity=XXXX.class) //指定关联对象

@JoinColumn(name="") //指定产生的外键字段名，默认参照被修饰属性主键

2、@OneToOne – 表示一个一对一的映射，只需要写一端

@OneToOne(targetEntity=XXXX.class) //指定关联对象

@JoinColumn(name="" unique=true) //外键字段名，设置唯一，只需要在一端写，默认参照被修饰属性主键

3、@OneToMany - 描述一个一对多的关联，该属性应该为集合Set类型，在数据库中并没有实际字段。这是两个持久化对象关系（关联关系，都需要在.xml中声明），与之前直接写集合（组件关系）不同

@OneToMany 默认会使用连接表做一对多关联，添加@JoinColumn(name="") 后，就会使用外键关联，而不使用连接表了，会在N端产生外键，默认指向当前主键。通常需要set属性上设置mappedBy，这样在one一端无法设置set属性，只能在many一端修改one属性。

4、@ManyToMany- 必须要用到一个连接表，两端都要写@joinTable，建立关系只能在一遍建立，另一边自动拥有，否则主键重复错误。

@joinTable(name=”表名”,joinColumns=@joinColumn(name=””,reference=””)//外键关联当前实体主键,inverseJoinColumns=@joinColumn(name=””,reference=””)//外键关联被修饰变量主键

6、组件属性关联实体：person（持久化类）下有address（@Embeddable修饰）,address下有Set<school>（持久化类）,保存person时address会把其属性写到主表中，address与school建立@OneToMany属性时，注解@JoinColumn（reference=”person\_id”）因为address没有自己的表，但是仍可以查到其下school信息。反之，school不能建立和address的@ManyToOne关系。

7、组件和关联的传播性：组件默认联级，关联则默认不启用联级(需要在@OneToOne等中写添加cascade=(CascadeType.ALL)属性)

8、继承映射：ORM解决了继承上的逻辑sql语句问题，我们只要按照对象操作即可

整个类层次对应一个表：优点是效率高，缺点是数据冗余,系统默认

@DiscriminatorColumn(name=””,discriminatorType=)修饰根类，name是表中辨别者列名称，discriminatorType是value类型要求。

@DiscriminatorValue(“”)用来修饰各个类，value是指辨别者列的值，即这条数据是属于哪个类的。

@Inheritance(strategy=InteritanceType.SINGLE\_TABLE)不写也可以，是默认值。

连接子类的映射策略：子类继承的属性在父类表中，独有的属性在自己表中，所有表的主键都是继承根表的主键，也是靠这个主键进行多表连接的。缺点查询子类跨表太多。

@Inheritance(strategy=InteritanceType.JOINED)修饰根类即可。子类表主键已确定好。

每个具体类对应一个表：主键不能简单自增长或者默认值，需要所有表中id不一样

@Inheritance(strategy=InteritanceType.TABLE\_PER\_CLASS)修饰根类，根类表主键需要使用@GenericGenerator(strategy=“hilo”)生成策略(保证整个数据库唯一)。

**Hibernate查询：HQL语法借鉴jdbc中executeUpdate()，但其操作面向对象。**

查询语法：session.createQuery(“hql”)

News as n 别名一定要写，不然错误

用到集合属性（集合属性总是不在主表里面的）需要显性连接 inner join n.authors就可以表示连接到authors表了，不支持on连接语法，因为对象之间关系已经知道了，on的内容已经确定了。

select age,name,id 多个属性，返回的是List<Object[]>,需要从数据里面取值，或者也可以封装这些属性成一个Map，List，Class都行，若不写select，直接from Bean则封装成Bean,这些返回值都是List<Obejct>,但这些最外层总是list。queue.list()返回值不会是null，所以自己需要通过size()==0判断。

多态查询：from后面可以跟实例化类名或者任何java类，都会把继承的实例化对象也全部找出来，比如select o from java.lang.Object o，可以把所有继承Object的持久化对象找出来。

Where后面接着表达式: list[1] map[“xyh”]等符合OGNL表达式的特性

命名查询：@NameQuery(name=””,query=””) 让hql语句从java代码中分离出来（推荐使用），结合session.getNamedQuery(“name”)

条件查询：支持更加面向对象的操作方法，基本功能上替代了上面的hql语句

当前对象下面属性查询

Criteria criteria=session.createCriteria(News.class);

Criterion criterion=Restrictions.sizeEq("authors", 2); //查询类下面的属性条件很多

criteria.add(criterion); //可以增加很多条件

List list=criteria.list(); //返回对象集合

关联对象下面属性查询：在之前的criteria基础上联级操作

.createCriteria("关联对象:authors",INNER\_JOIN（连接方式）).add(Restrictions.gt("关联对象属性", 300));

list()和iterate()区别：list()返回查询到的结果集，数据已经全部取出来了，iterate()只返回数据库中的全部id值放到缓存中，然后遍历id再从数据库中根据id取值。

支持sql语句查询：session.createSQLQuery(“sql”),甚至可以让sql查询语句查出结果转换成对象属性。

**过滤器：针对整个类，或者类下集合属性，适用于频繁查询且条件不变的情况，只是暂时性的查询则使用一般语句**

@FilterDefs(

{@FilterDef(name="newsFilter",parameters={@ParamDef(name="newsID",type="int")})

})

@Filter(name="newsFilter",condition="news\_id > :newsID") //condition是一个接在where后面的sql语句，表示过滤条件。?1这种占位符只能在hql中且用setParameter(“1”,0)，:name则两者都行，但需要用setXxx()。在函数中传入参数通常使用Object… params。

session.enableFilter("newsFilter").setInteger("newsID", 0); //在manager中开启过滤器

**事务控制和缓存机制：**

1、推荐一次请求(一次逻辑操作)对应一次session,不要一个web应用或者一次http session对应一次session，Spring托管下的Hibernate的session生命周期交由spring管理，在Dao层代码中只需要sessionFactory.getCurrentSession()即可。ORM框架必须要手动开启session事务机制，要么手动，要么交给spring控制事务，否则无法操作数据库。

2、Session打开之后并不连接数据库，等到用到才会连接，保存持久化对象，修改对象等操作时，session并不会立即写入数据库，除非显示调用flush()或者关闭session，所以里面会缓存很多持久化对象，所以需要定时的flush()/clear()，通常会被保存在一级缓存中(Session级别，Session关闭之后缓存都会清空)，以减少与数据库交互。

3、二级缓存(SessionFactory级别，Session关闭之后依然起作用)是指开发者自己定义哪些domain类产生的对象需要被缓存，二级缓存策略虽然缓存了整个对象，但只能针对主键id查询请求(即根据id号才能知道缓存中是否有这个对象)，若是条件查询毫无用处(除非查询全部数据，第一次之后采用iterate()遍历查询，本质仍是根据id查询)。缓存实现机制是利用第三方缓存EnCache。

1. 在hibernate.cfg.xml中开启二级缓存，配置相关EhCacheRegionFactory类(若spring框架共用，必须使用SingletonEhCacheRegionFactory类)和ehcache.xml文件(这里文件名只能用ehcache.xml，不能自定义，系统默认根据ehcache名字找到对应配置，即不需要在hibernate.cfg.xml中配置其路径)
2. 配置ehcache.xml文件，设置基本条件
3. domain层类中使用注解修饰@Cache(usage=CacheConcurrencyStrategy.READ\_ONLY)

这样该类无法update，但可以增加，所以一般针对不需要修改的数据使用此策略，若是经常会修改的数据，缓存机制也就失去了意义。

<!-- 配置二级缓存和查询缓存 -->

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

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

<!-- 二级缓存提供类SingletonEhCacheRegionFactory，表示和spring公用一个ehcache--> <property name="hibernate.cache.region.factory\_class">org.hibernate.cache.ehcache.SingletonEhCacheRegionFactory </property>

<!-- 二级缓存配置文件的位置 -->

<property name="hibernate.cache.provider\_configuration\_file\_resource\_path">ehcache.xml</property>

4、查询缓存(SessionFactory级别，主要就是针对条件查询)，二级缓存默认不会开启hql语句缓存。这里需要注意查询缓存 缓存的是对象id(和iterate()机制一样)，即当hql语句一样时(.setCacheable(true)都需要写)，会先到查询缓存中取id，如果有，则根据id到缓存(一级/二级)中取实体，如果缓存中取不到实体，再查询数据库，此时查就会根据id查，所以产生很多hql语句。这样导致若是要使用查询缓存，则必须开启二级缓存。

5、hibernate缓存根据数据库表最后更新的时间，判断缓存是否有效。若是数据库表更改了，其能够自动识别同步缓存。

**Domain层校验机制：**

1. 通过注解方式在domain层编写校验机制@NotNull @Length(min=3,max=6)等
2. 在dao层的save()中进行判断，如下：

ValidatorFactory factory = Validation.buildDefaultValidatorFactory();

Validator validator = factory.getValidator();

Set<ConstraintViolation<T>> constraintViolations =validator.validate(entity);

if (constraintViolations.size() != 0) {

return false;

}else {

getSessionFactory().getCurrentSession().save(entity);

return true;

}

1. 在action层也需要通过返回success和error告诉前端到底是否通过验证

**补充：**

1、Model层最好实现Serializable接口，然后添加static final UID(这个属性在修改field后，再次序列化时，若此属性相同则修改原来的，否则会报错)。

数据库中主键id类型就是Serializable，如session.get(Class k,Serializable pk)函数需要这样定义。

2、若是已有数据库，编写映射类，不需要所有列都映射，需要的列写在类中即可，主键列必须映射，且要是类中出现数据库中没有的字段，则数据库自动生成此列。

3、现在web开发，基本不使用外键(可以级联操作，一个sql语句就能查两张表，否则需要两条sql才能完成查询)，外键使得操作数据逻辑麻烦，可以用service层逻辑来保证数据完整性。