# Hibernate

## Hibernate状态

New Objecr()瞬时状态 -----save--🡪 持久状态-------close session--🡪游离状态(离线状态)

## Hibernate 主键策略

Native 交由数据库自动增长 assigned 程序员来指定 uuid 自动生成字符串

## Hibernate 关系映射

### 多对一 单向

<many to one> name ：对象中的属性名称 column :表示外键 先添加(一)， 再添加(多)

cascade :级联 一般不使用 ，用的话一般交由(一)方

### 一对多 单向 效率低 不使用

<set name =”” lazy =”extra” > 用lazy=extra,会根据取值的不同 判断是调用count /投影

<key column =””> 指定在对方外键的名称

<one to many calss=”” > 设置所持有集合的对象类型

</set>

### 一对多 双向

<many to one>

<set name =”” lazy =”extra” inverse=”true”> 将关系维护交给对方 多的一方

### 一对一 单向

<many to one name=”类的属性名”, column=”持有对方的外键名” unique=”true”>

unique=”true” ： 只能对应一个对象

### 一对一 双向

取出的是 没有维护关系的一方， 会自动将关联对象取出 1条

One 2 One 最好不使用双向关联，如果使用，尽可能在没有维护关系的一方取数据，hibernate会自动join

如果使用维护关系端取数据，在延迟加载取关联对象时，会再次取关联对象的 对端，会发出3条

<many to one >

<one to one name =”属性名” property-ref =”表示由对方来维护关系,对方的属性” >

### 多对多 双向

使用many to many 不论在哪边维护关系都比较麻烦 经常使用2个一对多,代替 多对多

<set name =”持有对方的属性” table =”关联表” lazy =”extra”>

<key column=”我在对方的外键” ></key>

<many to many calss=”对方的类型” column =“用我的外键来去对方的类型”></many to many>

</set>

### 多对多转换为 2个 一对多

往往需要中间表显示其他信息 学生 -- 课程 -----每门课程的成绩

一般不使用双向关联，特别不建议使用一的一方管理，因为从一的一端取数据有可能涉及分页，所以基本不使用双向关联。

## Annotation配置

### 实体类

@Entity @Table 属性get方法设置@ID / @column

### 多对一

@Many to one(fetch = FetchType.EAGER) @ joinColumn (name = “指定外键字段名称”)

@LazyCollection(LazyCollectionOption.FALSE) fetch =eager+ lazy =false ,取消延迟加载

@One to Many (mappedBy = "指定由对方维护关系,对方的字段" fetch = FetchType.EAGER)

@LazyCollection(LazyCollectionOption.EXTRA)

### 一对一

@One to One @ joinColumn(维护字段的名称)

@One to One (mappedBy = "指定由对方维护关系,对方的字段")

### 多对多

@many to many （mappedBy =“”）

@many to many

@JoinTable(name = "role\_admin", joinColumns = { @JoinColumn(name = "rid") }, inverseJoinColumns = { @JoinColumn(name = "aid") })

## HQL查询

### 对象(条件)查询

不能查询select \* , 可以使用 select XX from TABLE XX ;

条件查询 ：where xx = ? , query.setParameter(0, “yy”); 问号下标 从0开始

Where xx = :name query.setParameter(“name”, “yy”); 别名

### 函数聚合查询

支持 select count( \*) , 唯一结果 uniqueResult();

### 连接查询 having 取group函数条件

对象导航查询 使用对象的属性 / 不推荐使用 join

连接对象查询 直接使用left join / right join 不跟on

### DTO对象

Select new com.XX.entity.YY (stu.id as sid , stu.name as name )

使用完整类名 ，且必须有相应的构造函数

## 抓取策略 Fetch

### 多对一

XML 默认为lazy 添加 <many to one fetch =”join ”> join一条查询

此配置仅针对 LOAD方法 对HQL查询无用

解决方法

1. hbm.xml文件设置被抓取对象的 batch-size,每次抓取单个对象的X条关联对象
2. 在HQL语句中指定 使用fetch 指定抓取 from X join fetch , 如果使用join fetch 就无法使用count(\* ) JOIN FETCH 变为 JOIN 即可删除

Annotation 默认为JOIN查询

1.Mony to one 默认fetch =EAGER ==XML的JOIN /fetch=lazy = = XML的select

类名@

2.二方法 同 XML 2方法

### 一对多

XML <set fetch = “join ”> load 发送一条 HQL无用

1. XML 文件设置被抓取对象的set的batch-size完成批量抓取
2. fetch =”subselect” 子查询

Annotation

[1.@Fetch(FetchMode.JOIN)](mailto:1.@Fetch(FetchMode.JOIN)) JOIN查询

1. @Fetch(FetchMode.*SUBSELECT*)查询集合对象集合属性
2. @BatchSize(size = 20)

## 缓存

### N+1 问题

使用session.createQuery("from Student").iterate(); 返回迭代器，只是查询了到了所有的ID，在使用某个对象时，才发送SQL从数据库根据ID进行查询

存在的原因：有可能会在一个session中查询多次数据，list()方法，会一次把所有对象都查询出来，iterate()只查询ID，此时所有对象都存储在session缓存中，可以直接获取。

### 一级缓存 Session级别的缓存

### 二级缓存 SessionFactory级别的缓存

#### 对象缓存 仅缓存对象

开启二级缓存 XML

1. 导入二级缓存Jar， 一般常用ehcache
2. 开启二级缓存

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

1. 指定二级缓存类

<property name=*"hibernate.cache.provider\_class"*>net.sf.ehcache.hibernate.EhCacheProvider</property>

1. 4.0之后需要设置 Factory Class

<property name=*"hibernate.cache.region.factory\_class"*>org.hibernate.cache.ehcache.EhCacheRegionFactory</property>

1. 设置相应的ehcache.xml 文件

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

1. 在hbm..xml 使用cache <cache usage=*"read-only"* />
2. 可以在ehcache中 .xml 配置对象使用某个ehcache

<cache name=*"com.rzp.hibernate.entity.Student"*

1. 二级缓存 缓存的仅仅是对象，当去非对象时，不会将Object[] 进行缓存
2. 取出List对象，进行二级缓存，再使用iterate(); 会发送查询ID语句到二级缓存中进行查找，去掉N+1 ,若 再次使用List(),会发送查询对象语句到二级缓存中

Annotation 开启二级缓存

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

#### 查询缓存 仅缓存HQL的ID， 不缓存对象

1. 开启查询缓存

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

session.createQuery("from Student").setCacheable(**true**)

1. 只要HQL的语句不一样就不会开启缓存，只有2个HQL语句完全一致，且参数都要一致
2. 查询缓存 缓存的是ID， 如果对象关闭二级缓存，此时缓存的仅仅是对象的ID，再次使用对象时，只有ID，没有对象，还会发送SQL,必须开启 对象的二级缓存

## 数据库并发

#### 悲观锁

基于数据库机制，基于同步机制，只要读取这个对象，这个被对象就会被枷锁，只有在第一个线程读取完毕后，第二个线程再能读取

不使用

session.load(Student.**class**, 1,LockOptions.*UPGRADE*);

#### 乐观锁

在数据库中增加version 字段实现，每一次修改都会让version 数字 +1，在读取的时候，根据version的版本读取，如果并发修改就会抛异常

Hbm.xml 文件加入

<version name=*"version"*></version> 必须在ID 后面

实体类加入version int

Annotate

@version

## 一种提高效率的方式

#### 关系尽可能使用单向关联，不使用双向关联

#### 在大项目中，使用Hibernate可以酌情考虑以下原则

##### 不要使用对象关联，使用冗余字段代替外键，

需要手动更新冗余字段，比跨表查询速度快

##### 不使用HQL，全部查询SQL查询，自己实现cache

## SQL 查询

session.createSQLQuery("select \* from student").addEntity(Student.class).setFirstResult(2).setMaxResults(5).list();

session.createSQLQuery("select {stu.\*} ,{cla.\*} ,{g.\*} from xml\_student stu left join xml\_classroom cla on (stu.cid=cla.id) left join xml\_grade g on (cla.gid =g.id)").addEntity("stu",Student.class).addEntity("cla", ClassRoom.class).addEntity("g", Grade.class).list();

session.createSQLQuery("select stu.name as sname , cla.name as cname , g.name as gname from xml\_student stu left join xml\_classroom cla on (stu.cid=cla.id) left join xml\_grade g on (cla.gid =g.id)")

.setResultTransformer(Transformers.aliasToBean(StudentDTO.class)).list();

转换类必须设置 2个构造方法

stu.name as XX XX为结果转换对象的属性名