<http://blog.sina.com.cn/s/blog_697b968901016s31.html>

Hibernate的注释该如何使用？每一个注释代表什么意思？

导语：在我学习Hibernate的过程中，注释这一个我刚开始比较难搞懂的地方，刚开始也比较无头绪，但是，自己花了一天的时间对比较常用的标签进行了归纳总结，就明白了很多。所以建议大家学到此处，能够好好的花点时间，一个一个搞懂哈。花点时间自己也归纳一下效果挺好的。

1、@Entity(name="对应的表的名字，比如说user。则数据库中的表名就是这个属性的名字，但是有前提，是你没有定义@Table这个属性")

2、@Table 是用来表示这个类(实体)对应的数据库表的信息

有三个属性：1、name 2、catalog 3、schema

这三个属性都是可选的

name表示数据库中的这个类对应的表的名称()

//大家是不是疑惑了？上面的Entity的name属性也同样的表示表的名字，通过代码发现，当我们在@Table中没有定义name的属性的时候，则表名时根据@Entity的name来指定的，如果@Entity中也没有，则默认按照这个类的类名。

所以：

if(@Table的name!=null){

该类对应的数据库的表名 = @Table(name=“表名");

}elseif(@Entity的name!=null){

该类对应的数据库的表名 = @Entity(name="表名");

}else{

该类对应的数据库的表名 = 该类的类名

}

catalog和schema分别为catalog和schema的名称，并没有对程序的运行有太大的影响，可以自行了解。这两个属性也是可选的，一般我们也不会去写，让系统自动去默认值即可

3、对于主键的修饰，我们有1、@Id(必须) 2、@GeneratedValue(可选) 3、@GeneratedValues(可选)

1)、@Id定义了映射到数据库表的主键的属性,一个实体只能有一个属性被映射为主键.置于getXxxx()前.

2)、对于GeneratedValue

@GeneratedValue(strategy=GenerationType.AUTO)  自动增长，根据底层数据库自动选择。数据库支持自动增长类型，则为自动增长。

@GeneratedValue(GenerationType.TABLE 使用指定的表来决定主键的取值，结合@TableGeneraors使用

@GeneratedValue(strategy=GenerationType.IDENTITY)  唯一

@GeneratedValue(strategy=GenerationType.SEQUENCE)

strategy:表示主键生成策略,有AUTO,INDENTITY,SEQUENCE 和 TABLE 4种,分别表示让ORM框架自动选择,根据数据库的Identity字段生成,根据数据库表的Sequence字段生成,以有根据一个额外的表生成主键,默认为AUTO

一般我们设置为默认，也就是自动增长，即GenerationType.AUTO即可,让其自动增长，所以我们也可以不写

generator:表示主键生成器的名称,这个属性通常和ORM框架相关,例如,Hibernate可以指定uuid等主键生成方式.一般也不写，取默认值

3)、对于@GeneratedValues我们一般也就取一次

示例:

@Id

@GeneratedValues(strategy=StrategyType.SEQUENCE)

public int getPk() {

return pk;

}

4、@Basic  可以看做是基础修饰，默认每一个属性都有。默认如果不写的话为：@Basic(fetch=FetchType.EAGER,optional=true)

fetch表示的是加载策略/读取策略,有EAGER和LAZY两种,分别表示主支抓取和延迟加载,默认为EAGER.

但是，如果想要使用懒加载，则必须使用load方法，使用get方式是不可以的

当使用Query的时候会不会懒加载？？？

optional表示的是该属性是否允许为null,默认为true，即允许为null

示例:

@Basic(optional=false)

public String getAddress() {

return address;

}

5、@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用

1)、name：表示数据库中该字段的名称，默认情形属性名称与类属性名一直

2)、nullable：表示该字段是否允许为null，默认为true  （注：同@Basic(optional=true/false) 的不同之处在于一个是类对象的属性，一个是数据库中的表的属性）

3)、unique：表示该字段是否唯一表示，默认为false

4)、length：表示该字段的大小，仅对String类型的字段有效

5)、insertable：表示在ORM框架执行插入操作的时候，该字段是否应出现Insert语句中，默认为true。

6)、updateable：表示在ORM框架执行更新操作的时候，该字段是否应该出现在UPDATE语句中,默认为true.对于一经创建就不可以更改的字段,该属性非常有用,如对于birthday字段.即 如果设置成false的时候，该字段就是不可更新的了

7)、columnDefinition:表示该字段在数据库中的实际类型.通常ORM框架可以根据属性类型自动判断数据库中字段的类型,但是对于Date类型仍无法确定数据库中字段类型究竟是DATE,TIME还是TIMESTAMP.此外,String的默认映射类型为VARCHAR,如果要将String类型映射到特定数据库的BLOB或TEXT字段类型,该属性非常有用.

示例:

@Column(name="BIRTH",nullable="false",columnDefinition="DATE")

public String getBithday() {

return birthday;

}

6、@Transient :表示该属性并非映射到数据库中的字段，即该字段数据库中的表是没有的。

@Transient与@Basic两者是不能同时存在的，如果没有定义@Transient，则默认为@Basic

我们可以应用到一些我们自己的定义的方法里面，比如说一个Person类，有birthday这个类型，然后我们为了得到它已经多少岁了，我们通常都是是在Person类中直接写一个 public int getAge()  这个时候age属性就不需要在数据库中的表定义了，因为可以通过计算得到，否则就有点儿数据冗余了。这个时候我们就通过@Transient属性让它不属于表的一个字段

示例:

//根据birth计算出age属性

@Transient

public int getAge() {

return getYear(new Date()) - getYear(birth);

}

出自：[java快快飞](http://www.keyprint.cn/)

8、@ManyToOne(fetch=FetchType,cascade = CascadeType)   //可选

表示的是一个 多对一 的映射。该注解标注的属性通常是数据库表的外键

1)、optional：是否允许该字段为null，即这个属性值是否可以为null，就是说是不是一定要有外键，比如说学生对班级，学生不可以没有班级，所以为false，即不能够为null。

2)、fetch，默认为FetchType.EAGER   还有FetchType.LAZY（懒加载）

3)、cascade：这边比较经常用。表示默认的级联操作方式。有ALL(all)、PERSIST(persist)、MERGE(merge)、REFRESH(refresh)和REMVOE(remove)中的若干组合，默认为无级联操作。

CascadeType各个值的含义：

CascadeType.REFRESH(refresh):表示的是级联刷新，当多个用户同时操作一个实体，为了用户取到的数据是实时的，在用实体中的数据之前就可以调用一下refresh()方法！

CsacadeType.REMOVE(remove):级联删除，当调用remove()方法删除Order实体时会先级联删除OrderItem的相关数据

CascadeType.MERGE(merge):级联更新，当调用了Merger()方法，如果Order中的数据改变了会相应的更新到OrderItem中的数据

CascadeType.ALL：包含以上所有级联属性。。上面的哈

(注：以上几种级联操作，只能实在满足数据库的约束时才能生效，比如上边的Order和OrderItem存在主外键关联，所以执行REMOVE()方法时是不能实现级联删除的)

CascadeType.PERSIST(persist)：级联保存，当调用了Persist()方法，会级联保存相应的数据。

4）、TagertEntity：表示该属性关联的实体类型，该属性通常不必指定，ORM框架根据属性类型自动判断targetEntity

示例:

//订单Order和用户User是一个ManyToOne的关系

//在Order类中定义

@ManyToOne()

@JoinColumn(name="USER")

public User getUser() {

return user;

}

9、@JoinColumn（可选）

这个量的作用就是用来定义外键在我们这个表中的属性名。只能够在有外键处才能够使用。@JoinColumn(name="USER")

如果我们没有定义这个属性，则默认的外键的属性名，如：

实体Order有一个user属性来关联实体User,则Order的user属性为一个外键,

其默认的名称为实体User的名称+下划线+实体User的主键名称

10、@OneToMany

描述的是一个一对多的关联，该属性应该为集体类型，在数据库中并没有实际字段。

1)fetch:表示的是抓取策略，默认为FetchType.LAZY,因为关联的多个对象通常不必从数据库中预先读取到内存

2)cascade:表示级联操作策略，对于OneToMany类型的关联非常重要，通常该实体更新或删除的时候，其关联的实体也应当被更新或删除

例如:实体User和Order是OneToMany的关系,则实体User被删除时,其关联的实体Order也应该被全部删除

示例:

@OneTyMany(cascade=ALL)

public List getOrders() {

return orders;

}

11、@OneToOne(fetch=FetchType,cascade=CascadeType)

这介描述的是一个一对一的关联

fetch：表示的是抓取策略，默认为LAZY

cascade：表示的是级联删除策略

示例:

@OneToOne(fetch=FetchType.LAZY)

public Blog getBlog() {

return blog;

}

12、@ManyToMany

@ManyToMany描述的是一个多对多关联。多对多关联实际上是两个一对多关联，但是在ManyToMany描述中，中间表是由ORM框架自动处理

targetEntity:表示多对多关联的另一个实体类的全名,例如:package.Book.class

 mappedBy：表示多对多关联的另一个实体类对应的属性集合名称

示例:

User实体表示用户,Book实体表示书籍,为了描述用户收藏的书籍,可以在User和Book之间建立ManyToMany关联

@Entity

public class User {

private List books;

@ManyToMany(targetEntity=package.Book.class)

public List getBooks() {

return books;

}

public void setBooks(List books) {

this.books=books;

}

}

@Entity

public class Book {

private List users;

@ManyToMany(targetEntity=package.Users.class, mappedBy="books")

public List getUsers() {

return users;

}

public void setUsers(List users) {

this.users=users;

}

}

两个实体间相互关联的属性必须标记为@ManyToMany,并相互指定targetEntity属性,

需要注意的是,有且只有一个实体的@ManyToMany注解需要指定mappedBy属性,指向targetEntity的集合属性名称

利用ORM工具自动生成的表除了User和Book表外,还自动生成了一个User\_Book表,用于实现多对多关联

13、@MappedSuperclass

可选

@MappedSuperclass可以将超类的JPA注解传递给子类,使子类能够继承超类的JPA注解

示例:

@MappedSuperclass

public class Employee() {

....

}

@Entity

public class Engineer extends Employee {

.....

}

@Entity

public class Manager extends Employee {

.....

}

14、@Embedded

可选

@Embedded将几个字段组合成一个类,并作为整个Entity的一个属性.

例如User包括id,name,city,street,zip属性.

我们希望city,street,zip属性映射为Address对象.这样,User对象将具有id,name和address这三个属性.

Address对象必须定义为@Embededable

示例:

@Embeddable //可以被嵌入的

public class Address {city,street,zip}

@Entity

public class User {

@Embedded   //将address的每一个属性嵌入到User的表中，Address类没有自己的表

public Address getAddress() {

..........

}

}