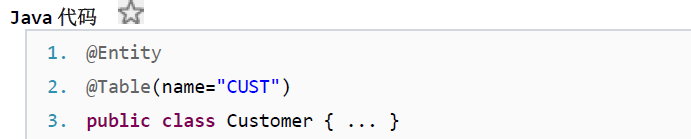
# 表

## @Table 和@Entity

Table用来定义entity主表的name，catalog，schema等属性。

属性说明：





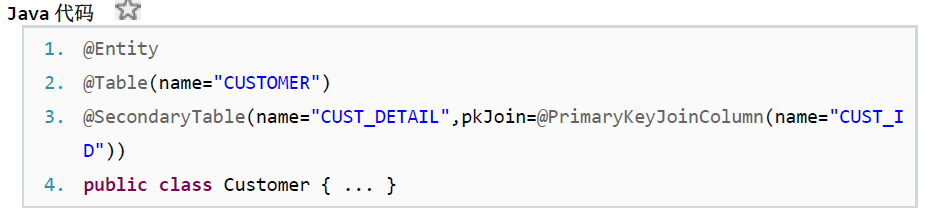
## @SecondaryTable

一个entity class可以映射到多表，SecondaryTable用来定义单个从表的名字，主键名字等属性。

属性说明：



下面的代码说明Customer类映射到两个表，主表名是CUSTOMER，从表名是CUST\_DETAIL，从表的主键列和主表的主键列类型相同，列名为CUST\_ID。

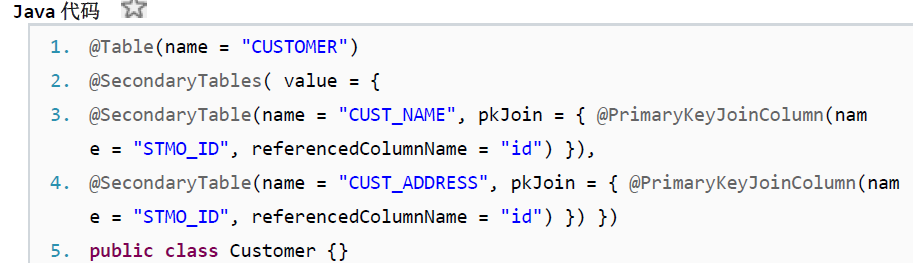


## @SecondaryTables

当一个entity class映射到一个主表和多个从表时，用SecondaryTables来定义各个从表的属性。

属性说明：

 value：定义一个SecondaryTable数组，指定每个从表的属性。



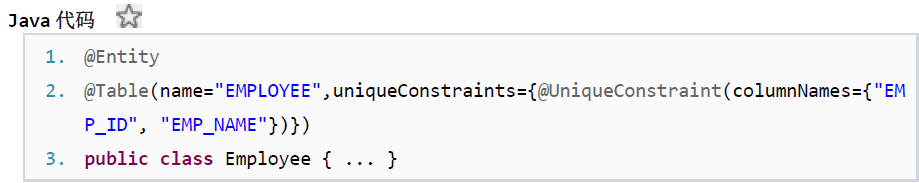
# 表的属性

## @UniqueConstraint

UniqueConstraint定义在Table或SecondaryTable元数据里，用来指定建表时需要建唯一约束的列。

属性说明：

 columnNames：定义一个字符串数组，指定要建唯一约束的列名。

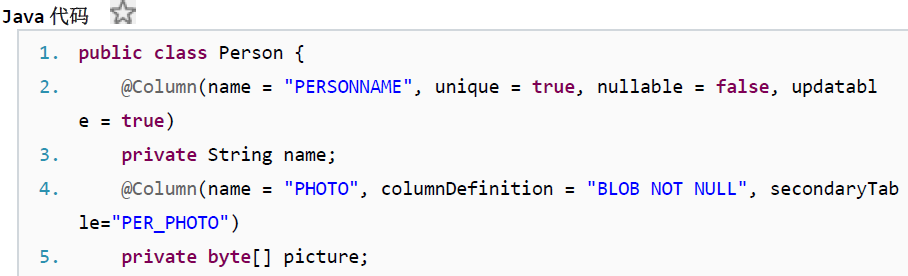


# 字段

## @Column

Column元数据定义了映射到数据库的列的所有属性：列名，是否唯一，是否允许为空，是否允许更新等。

属性说明：

## @JoinColumn

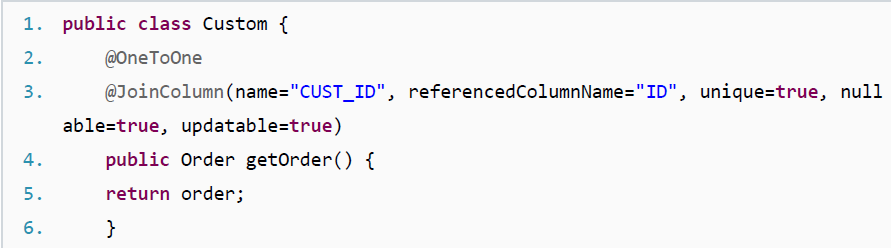
如果在entity class的field上定义了关系（one2one或one2many等），我们通过JoinColumn来定义关系的属性。JoinColumn的大部分属性和Column类似。

属性说明：



下面的代码说明Custom和Order是一对一关系。在Order对应的映射表建一个名为CUST\_ID的列，该列作为外键指向Custom对应表中名为ID的列。

**Java**代码



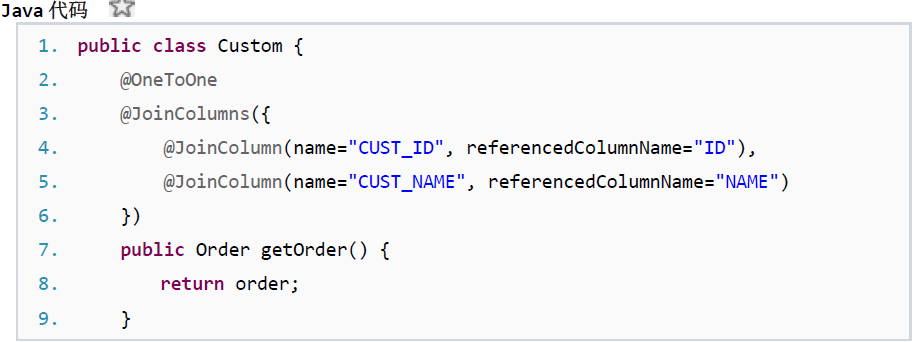
## @JoinColumns

如果在entity class的field上定义了关系（one2one或one2many等），并且关系存在多个JoinColumn，用JoinColumns定义多个JoinColumn的属性。

属性说明：

 value：定义JoinColumn数组，指定每个JoinColumn的属性。

下面的代码说明Custom和Order是一对一关系。在Order对应的映射表建两列，一列名为CUST\_ID，该列作为外键指向Custom对应表中名为ID的列,另一列名为CUST\_NAME，该列作为外键指向Custom对应表中名为NAME的列。



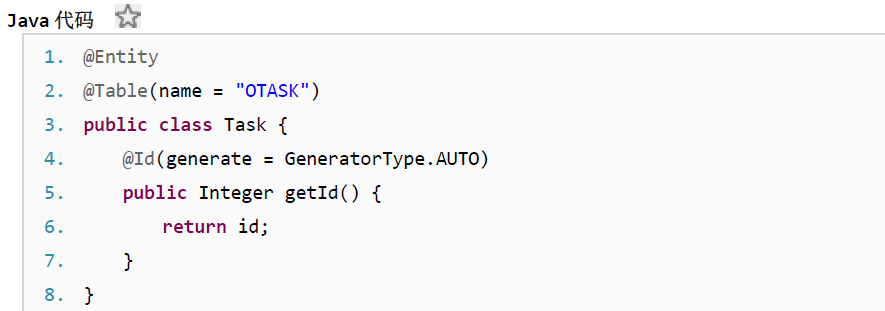
## @Id

声明当前field为映射表中的主键列。id值的获取方式有五种：TABLE, SEQUENCE, IDENTITY, AUTO, NONE。Oracle和DB2支持SEQUENCE，SQL Server和Sybase支持IDENTITY,mysql支持AUTO。所有的数据库都可以指定为AUTO，我们会根据不同数据库做转换。NONE(默认)需要用户自己指定Id的值。

属性说明：



下面的代码声明Task的主键列id是自动增长的。(Oracle和DB2从默认的SEQUENCE取值，SQL Server和Sybase该列建成IDENTITY，mysql该列建成auto increment。)



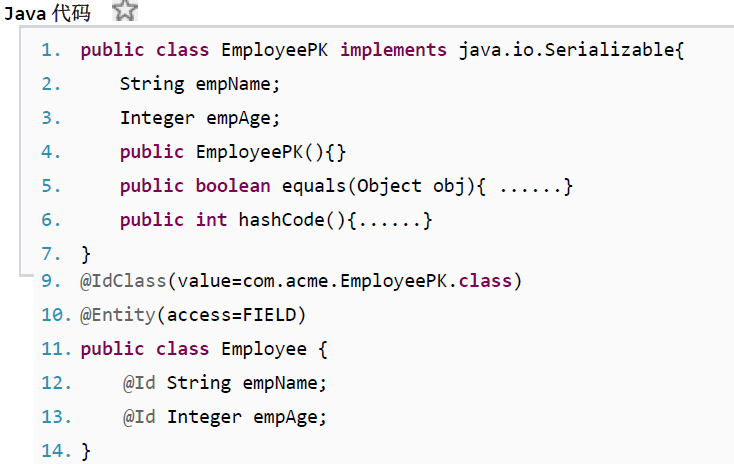
## @IdClass

当entity class使用复合主键时，需要定义一个类作为id class。id class必须符合以下要求:类必须声明为public，并提供一个声明为public的空构造函数。必须实现Serializable接，覆写equals()和hashCode（）方法。entity class的所有id field在id class都要定义，且类型一样。

属性说明：

 value：id class的类名

下面的代码声明Task的主键列id是自动增长的。(Oracle和DB2从默认的SEQUENCE取值，SQL Server和Sybase该列建成IDENTITY，mysql该列建成auto increment。)

****

# 对应关系

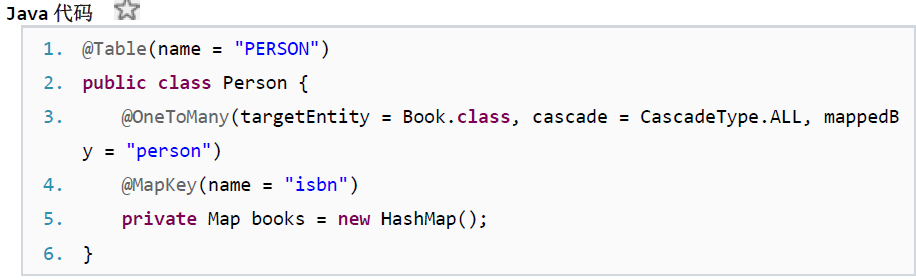
## @MapKey

在一对多，多对多关系中，我们可以用Map来保存集合对象。默认用主键值做key，如果使用复合主键，则用id class的实例做key，如果指定了name属性，就用指定的field的值做key。

属性说明：

 name：用来做key的field名字

下面的代码说明Person和Book之间是一对多关系。Person的books字段是Map类型，用Book的isbn字段的值作为Map的key。



## @MappedSuperclass

使用@MappedSuperclass指定一个实体类从中继承持久字段的超类。当多个实体类共享通用的持久字段或属性时，这将是一个方便的模式。

您可以像对实体那样使用任何直接和关系映射批注（如 @Basic 和 @ManyToMany）对该超类的字段和属性进行批注，但由于没有针对该超类本身的表存在，因此这些映射只适用于它的子类。继承的持久字段或属性属于子类的表。

可以在子类中使用@AttributeOverride或@AssociationOverride来覆盖超类的映射配置。

@MappedSuperclass没有属性。

**Java**代码

1. //如何将Employee指定为映射超类

2. @MappedSuperclass

3. **public class** Employee {

4. @Id

5. **protected** Integer empId;

6.

7. @Version

8. **protected** Integer version;

9.

10. @ManyToOne

11. @JoinColumn(name="ADDR")

12. **protected** Address address;

13. }

14.

15. //如何在实体类中使用@AttributeOverride以覆盖超类中设置的配置。

16. @Entity

17. @AttributeOverride(name="address", column=@Column(name="ADDR\_ID"))

18. **public class** PartTimeEmployee **extends** Employee {

19. @Column(name="WAGE")

20. **protected** Float hourlyWage;

21. }

## @PrimaryKeyJoinColumn

在三种情况下会用到@PrimaryKeyJoinColumn

 继承。

 entity class映射到一个或多个从表。从表根据主表的主键列（列名为referencedColumnName值的列），建立一个类型一样的主键列，列名由name属性定义。

 one2one关系，关系维护端的主键作为外键指向关系被维护端的主键，不再新建一个外键列。

属性说明：

 name：列名。

 referencedColumnName：该列引用列的列名

 columnDefinition：定义建表时创建此列的DDL

下面的代码说明Customer映射到两个表，主表CUSTOMER,从表CUST\_DETAIL，从表需要建立主键列CUST\_ID，该列和主表的主键列id除了列名不同，其他定义一样。

**Java**代码

1. @Entity

2. @Table(name="CUSTOMER")

3. @SecondaryTable(name="CUST\_DETAIL",pkJoin=@PrimaryKeyJoinColumn(name="CUST\_ID"，referencedColumnName="id"))

4. **public class** Customer {

5. @Id(generate = GeneratorType.AUTO)

6. **public** Integer getId() {

7. **return** id;

8. }

9. }

下面的代码说明Employee和EmployeeInfo是一对一关系，Employee的主键列id作为外键指向EmployeeInfo的主键列INFO\_ID。

**Java**代码

1. @Table(name = "Employee")

2. **public class** Employee {

3. @OneToOne

4. @PrimaryKeyJoinColumn(name = "id", referencedColumnName="INFO\_ID")

5. EmployeeInfo info;

6. }

## 13 @PrimaryKeyJoinColumns

如果entity class使用了复合主键，指定单个PrimaryKeyJoinColumn不能满足要求时，可以用PrimaryKeyJoinColumns来定义多个PrimaryKeyJoinColumn

属性说明：

 value： 一个PrimaryKeyJoinColumn数组，包含所有PrimaryKeyJoinColumn

下面的代码说明了Employee和EmployeeInfo是一对一关系。他们都使用复合主键，建表时需要在Employee表建立一个外键，从Employee的主键列id,name指向EmployeeInfo的主键列INFO\_ID和INFO\_NAME

**Java**代码

1. @Entity

2. @IdClass(EmpPK.**class**)

3. @Table(name = "EMPLOYEE")

4. **public class** Employee {

5. **private int** id;

6. **private** String name;

7. **private** String address;

8. @OneToOne(cascade = CascadeType.ALL)

9. @PrimaryKeyJoinColumns({

10. @PrimaryKeyJoinColumn(name="id", referencedColumnName="INFO\_ID"),

11. @PrimaryKeyJoinColumn(name="name" , referencedColumnName="INFO\_NAME")})

12. EmployeeInfo info;

13. }

14.

15. @Entity

16. @IdClass(EmpPK.**class**)

17. @Table(name = "EMPLOYEE\_INFO")

18. **public class** EmployeeInfo {

19. @Id

20. @Column(name = "INFO\_ID")

21. **private int** id;

22. @Id

23. @Column(name = "INFO\_NAME")

24. **private** String name;

25. }

## 15 @Version

Version指定实体类在乐观事务中的version属性。在实体类重新由EntityManager管理并且加入到乐观事务中时，保证完整性。每一个类只能有一个属性被指定为version，version属性应该映射到实体类的主表上。

属性说明：

value： 一个PrimaryKeyJoinColumn数组，包含所有PrimaryKeyJoinColumn

下面的代码说明versionNum属性作为这个类的version，映射到数据库中主表的列名是OPTLOCK

**Java**代码

1. @Version

2. @Column("OPTLOCK")

3. **protected int** getVersionNum() { **return** versionNum; }

## 16 @Lob

Lob指定一个属性作为数据库支持的大对象类型在数据库中存储。使用LobType这个枚举来定义Lob是二进制类型还是字符类型。

LobType枚举类型说明：

BLOB 二进制大对象，Byte[]或者Serializable的类型可以指定为BLOB。

CLOB 字符型大对象，char[]、Character[]或String类型可以指定为CLOB。

属性说明：

fetch：定义这个字段是lazy loaded还是eagerly fetched。数据类型是FetchType枚举，默认为LAZY，即lazy loaded.

type：定义这个字段在数据库中的JDBC数据类型。数据类型是LobType枚举，默认为BLOB。

下面的代码定义了一个BLOB类型的属性和一个CLOB类型的属性

**Java**代码

1. @Lob

2. @Column(name="PHOTO" columnDefinition="BLOB NOT NULL")

3. **protected** JPEGImage picture;

4.

5. @Lob(fetch=EAGER, type=CLOB)

6. @Column(name="REPORT")

7. **protected** String report;

## 17 @JoinTable

JoinTable在many-to-many关系的所有者一边定义。如果没有定义JoinTable，使用JoinTable的默认值。

属性说明：

 table：这个join table的Table定义。

 joinColumns：定义指向所有者主表的外键列，数据类型是JoinColumn数组。

 inverseJoinColumns：定义指向非所有者主表的外键列，数据类型是JoinColumn数组。

下面的代码定义了一个连接表CUST和PHONE的join table。join table的表名是CUST\_PHONE，包含两个外键，一个外键是CUST\_ID，指向表CUST的主键ID，另一个外键是PHONE\_ID，指向表PHONE的主键ID。

**Java**代码

1. @JoinTable(

2. table=@Table(name=CUST\_PHONE),

3. joinColumns=@JoinColumn(name="CUST\_ID", referencedColumnName="ID"),

4. inverseJoinColumns=@JoinColumn(name="PHONE\_ID", referencedColumnName="ID")

5. )

## 18 @TableGenerator

TableGenerator定义一个主键值生成器，在Id这个元数据的generate＝TABLE时，generator属性中可以使用生成器的名字。生成器可以在类、方法或者属性上定义。

生成器是为多个实体类提供连续的ID值的表，每一行为一个类提供ID值，ID值通常是整数。

属性说明：

 name：生成器的唯一名字，可以被Id元数据使用。

 table：生成器用来存储id值的Table定义。

 pkColumnName：生成器表的主键名称。

 valueColumnName：生成器表的ID值的列名称。

 pkColumnValue：生成器表中的一行数据的主键值。

 initialValue：id值的初始值。

 allocationSize：id值的增量。

下面的代码定义了两个生成器empGen和addressGen，生成器的表是ID\_GEN

**Java**代码

1. @Entity

2. **public class** Employee {

3. ...

4. @TableGenerator(name="empGen",table=@Table(name="ID\_GEN"),pkColumnName="GEN\_KEY", valueColumnName="GEN\_VALUE",pkColumnValue="EMP\_ID",allocationSize=1)

5. @Id(generate=TABLE, generator="empGen")

6. **public int** id;

7. ...

8. }

9.

10. @Entity

11. **public class** Address {

12. ...

13. @TableGenerator(name="addressGen",table=@Table(name="ID\_GEN"),pkColumnValue="ADDR\_ID")

14. @Id(generate=TABLE, generator="addressGen")

15. **public int** id;

16. ...

17. }

## 19 @SequenceGenerator

SequenceGenerator定义一个主键值生成器，在Id这个元数据的generator属性中可以使用生成器的名字。生成器可以在类、方法或者属性上定义。生成器是数据库支持的sequence对象。

属性说明：

 name：生成器的唯一名字，可以被Id元数据使用。

 sequenceName：数据库中，sequence对象的名称。如果不指定，会使用提供商指定的默认名称。

 initialValue：id值的初始值。

 allocationSize：id值的增量。

下面的代码定义了一个使用提供商默认名称的sequence生成器

**Java**代码

1. @SequenceGenerator(name="EMP\_SEQ", allocationSize=25)

## 20 @DiscriminatorColumn

DiscriminatorColumn定义在使用SINGLE\_TABLE或JOINED继承策略的表中区别不继承层次的列

属性说明：

 name：column的名字。默认值为TYPE。

 columnDefinition：生成DDL的sql片断。

 length：String类型的column的长度，其他类型使用默认值10。

下面的代码定义了一个列名为DISC，长度为20的String类型的区别列

**Java**代码

1. @Entity

2. @Table(name="CUST")

3. @Inheritance(strategy=SINGLE\_TABLE,discriminatorType=STRING,discriminatorValue="CUSTOMER")

4. @DiscriminatorColumn(name="DISC", length=20)

5. **public class** Customer { ... }

## 21 @NamedQuery

在使用JPA持久化规范的应用程序中，可以使用实体管理器动态创建和执行查询，也可以预定义查询并在运行时按名称执行。

使用@NamedQuery创建与@Entity或@MappedSuperclass关联的预定义查询，这些查询：

 使用JPA查询语言进行基于任何基础数据库的可移植执行

 经常被使用

 比较复杂并且难于创建

 可以在不同实体之间共享

 只返回实体（从不返回标量值），并只返回一个类型的实体

属性说明：

 query：(必须属性)要指定查询，请将 query 设置为 JPA 查询语言（作为 String）

 hints：默认值：空 QueryHint 数组。默认情况下，JPA 持续性提供程序假设 SQL 查询应完全按照 query 属性提供的方式执行。要微调查询的执行，可以选择将hints设置为一个QueryHint数组（请参阅 @QueryHint）。在执行时，EntityManager 将向基础数据库传递提示。

 name:(必须属性)要指定查询名称，请将name设置为所需的String名称

下面的代码使用@NamedQuery批注定义一个JPA查询语言查询，该查询使用名为firstname的参数

**Java**代码

1. //使用 @NamedQuery 实现一个带参数的查询

2. @Entity

3. @NamedQuery(name="findAllEmployeesByFirstName",

4. query="SELECT OBJECT(emp) FROM Employee emp WHERE emp.firstName = :firstname")

5. **public class** Employee **implements** Serializable {

6. …

7. }

8.

9. //执行命名查询

10. Query queryEmployeesByFirstName = em.createNamedQuery(“findAllEmployeesByFirstName”);

11. queryEmployeeByFirstName.setParameter(“firstName”, “John”);

12. Collection employees = queryEmployessByFirstName.getResultList();

## 22 @NamedQueries

如果需要指定多个@NamedQuery，则必须使用一个@NamedQueries指定所有命名查询

属性说明：

 value：要指定两个或更多属性覆盖，请将value设置为NamedQuery实例数组

**Java**代码

1. @Entity

2. @NamedQueries({@NamedQuery(name="findAllEmployeesByFirstName",

3. query="SELECT OBJECT(emp) FROM Employee emp WHERE emp.firstName = :firstname"),

4. @NamedQuery(name="findAllEmployeesByLasttName",

5. query="SELECT OBJECT(emp) FROM Employee emp WHERE emp.lasstName = :lastname")})

6. **public class** PartTimeEmployee **extends** Employee {

7.

8. }

## 23 @NamedNativeQuery

使用@NamedNativeQuery创建与@Entity或@MappedSuperclass关联的预定义查询，这些查询：

 使用基础数据库的原生SQL

 经常被使用

 比较复杂并且难于创建

 可以在不同实体之间共享

 返回实体、标量值或两者的组合（另请参阅 @ColumnResult、@EntityResult、@FieldResult 和@SqlResultSetMapping）

属性说明：

 query：(必须属性)要指定查询，请将query设置为SQL查询（作为String）

 hints：默认值：空 QueryHint 数组。默认情况下，JPA 持续性提供程序假设 SQL 查询应完全按照 query 属性提供的方式执行。要微调查询的执行，可以选择将hints设置为一个QueryHint数组（请参阅 @QueryHint）。在执行时，EntityManager 将向基础数据库传递提示。

 name：(必须属性)要指定查询名称，请将name设置为所需的String名称

 resultClass：默认值：JPA 持续性提供程序假设结果类是关联实体的Class.要指定结果类，请将resultClass设置为所需的 Class

 resultSetMapping：默认值：JPA持续性提供程序假设原生SQL查询中的SELECT语句：返回一个类型的实体；包括与返回的实体的所有字段或属性相对应的所有列；并使用与字段或属性名称（未使用AS语句）相对应的列名。要控制JPA持续性提供程序如何将JDBC结果集映射到实体字段或属性以及标量，请通过将resultSetMapping设置为所需的@SqlResultSetMapping的String名称来指定结果集映射

**Java**代码

1. //定义一个使用基础数据库的原生SQL的查询

2. @Entity

3. @NamedNativeQuery(name="findAllEmployees",query="SELECT \* FROM EMPLOYEE")

4. **public class** Employee **implements** Serializable {

5.

6. }

7.

8. //Hibernate如何使用EntityManager获取此查询以及如何通过Query方法getResultList执行该查询

9. Query queryEmployees = em.createNamedQuery("findAllEmployees");

10. Collection employees = queryEmployees.getResultList();

## 24 @NamedNativeQueries

如果需要指定多个@NamedNativeQuery，则必须使用一个@NamedNativeQueries指定所有命名查询

属性说明：

 value：要指定两个或更多属性覆盖，请将value设置为NamedNativeQuery实例数组

下面代码显示了如何使用此批注指定两个命名原生查询

**Java**代码

1. @Entity

2. @NamedNativeQueries({@NamedNativeQuery(name="findAllPartTimeEmployees",

3. query="SELECT \* FROM EMPLOYEE WHERE PRT\_TIME=1"),

4. @NamedNativeQuery(name="findAllSeasonalEmployees,

5. query="SELECT \* FROM EMPLOYEE WHERE SEASON=1")})

6.

7. **public class** PartTimeEmployee **extends** Employee {

8.

9. }

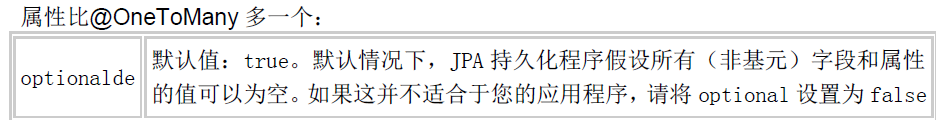
# 表关系

## @OneToMany和@ManyToOne

|  |  |
| --- | --- |
| 属性说明：  cascade | 默认值CascadeType的空数组。默认情况下，JPA不会将任何持久化操作层叠到关联的目标。如果希望某些或所有持久化操作层叠到关联的目标，应将cascade设置为一个或多个 CascadeType类型的枚举值，其中包括：● ALL – 针对拥有实体执行的任何持久化操作均层叠到关联的目标。● MERGE – 如果合并了拥有实体，则将merge层叠到关联的目标。● PERSIST – 如果持久保存拥有实体，则将persist层叠到关联的目标。● REFRESH – 如果刷新了拥有实体，则refresh为关联的层叠目标。● REMOVE – 如果删除了拥有实体，则还删除关联的目标。 |
| fetch | 在Hibernate里用时默认值：FetchType.LAZY，它要求程序运行时延迟加载所有的集合和实体。如果这不适合于应用程序或特定的持久字段，将fetch设置为FetchType.EAGER，它提示程序在首次访问数据时应马上加载所有的集合和实体 |
| mappedBy | 默认值：如果关系是单向的，则该关联提供程序确定拥有该关系的字段。如果关系是双向的，则将关联相反（非拥有）方上的mappedBy元素设置为拥有此关系的字段或属性的名称 |
| targetEntity | 默认值：使用一般参数定义的Collection的参数化类型。默认情况下，如果使用通过一般参数定义的Collection，则程序将从被引用的对象类型推断出关联的目标实体。如果Collection不使用一般参数，则必须指定作为关联目标的实体类：将关联拥有方上的 targetEntity元素设置为作为关系目标的实体的Class |



## @OneToOne



## @OrderBy

一般将@OrderBy与@OneToMany和@ManyToMany一起使用

在一对多，多对多关系中，有时我们希望从数据库加载出来的集合对象是按一定方式排序的，这可以通过OrderBy来实现，默认是按对象的主键升序排列。

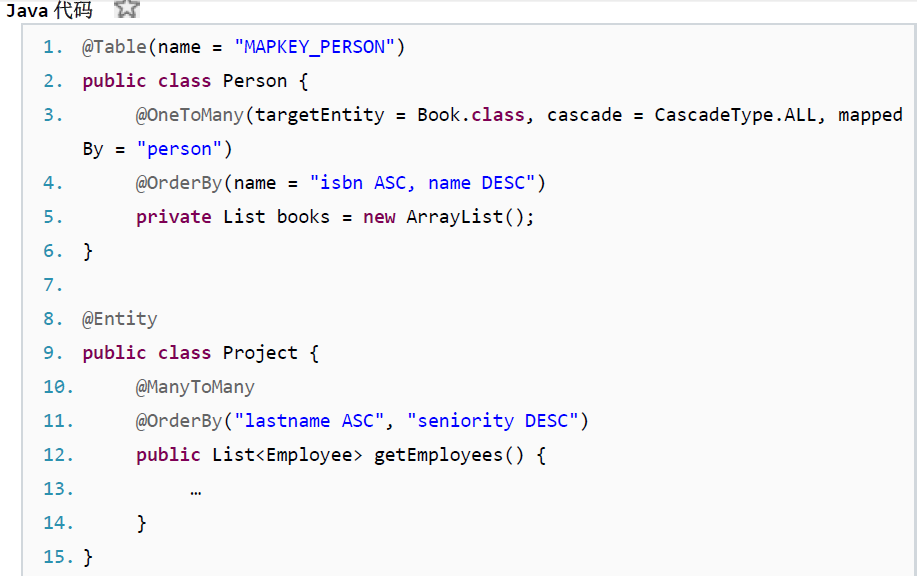
属性说明：

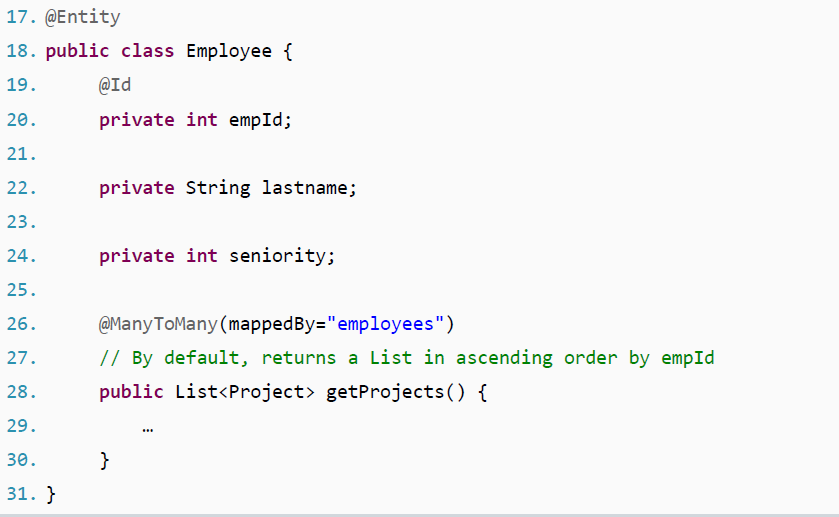
value：字符串类型，指定排序方式。

格式为"fieldName1 [ASC|DESC],fieldName2 [ASC|DESC],......"

排序类型可以不指定，默认是ASC升序。

下面的代码说明Person和Book之间是一对多关系。集合books按照Book的isbn升序，name降序排列。





# 表关系

## @Transient--持久化关系

Transient用来注释entity的属性，指定的这些属性不会被持久化，也不会为这些属性建表

**Java**代码

1. @Transient

2. **private** String name;