

0≡(-∞,+∞)


放空自己

目录视图

摘要视图

RSS 订阅

个人资料



piwwwiq

访问: 25724次

积分: 1145

等级:

BLOG > 4

排名: 第16424名

原创: 86篇 转载: 35篇

译文: 0篇 评论: 13条

文章搜索

文章分类

Java (18)

ACM (54)

IDE&工具 (2)

数据库 (8)

C++ (7)

算法&数据结构 (11)

设计 (2)

SSH (14)

JSH (11)

文章存档

2014年10月 (1)

2014年07月 (2)

2014年06月 (2)

2014年05月 (17)

2014年04月 (10)

展开

推荐文章

CSDN学院精品课程推荐

《京东技术解密》有奖试读获奖名单公布

"我的2014"年度征文活动火爆开启

11.1-12.31推荐文章汇总

JPA注解的使用,用于实体类的注解

分类: 数据库 JSH SSH

2014-05-21 10:02

699人阅读

评论(0)

收藏

举报

1、@Entity(name="EntityName")

表示该类是一个可持化的实体。当在容器中时，服务器将会首先加载所有标注了@Entity注释的实体类，其中@Entity中的" name "属性表示实体名称，若不作设置，默认为标注实体类的名称（注意大小写与系统的关系。）。@Entity标注是必需的，name属性为可选。

Java代码

☆

```
01. @Entity(name="person_1")
02. public class Person implements Serializable {
03.
04.
05.     public P
06.         id=System.currentTimeMillis();
07.     }
08.
09.     public Person(Long id){
10.         this.id = id;
11.     }
12. }
```

@Entity标注的实体类至少需要有一个无参的构造方法。这是因为，在使用类反射机制 Class.newInstance()方法创建实例时，需要有一个默认的无参数构造方法，否则会抛出实例化异常（InstantiationException）。

如果指定name属性后，在执行JPQL时应该使用name属性所指的名称。像上面的标注后，在执行JPQL时要像下面这样：

Sql代码

☆

```
01. SELECT p FROM person_1 AS p
```

name属性为默认值时就使用类名称：

Sql代码

☆

```
01. SELECT p FROM Person AS p;
```

有两点须要注意：

- (1) 实体可继承，非实体类可以继承自实体类，实体类也要中以继承自非实体类。
- (2) 抽象类（abstract）也可以标注为实体类。

2、@Table

在使用@Table标记时，需要注意以下几个问题。

- (1) 此标记需要标注在类名前，不能标注在方法或属性前。
- (2) name 属性表示实体所对应表的名称，默认表名为实体名称。
- (3) catalog 和schema 属性表示实体指定的目录名或数据库名，这个根据不同的数据类型有所不同。
- (4) uniqueConstraints 属性表示该实体所关联的唯一约束条件，一个实体可以有多个唯一的约束，默认没有约束条件。
- (5) 若使用uniqueConstraints 属性时，需要配合标记UniqueConstraint标记来使用。

| |
|--------------------------------------|
| * Nginx 模块开发 |
| * 我的2014碎碎念 |
| * 2015年10大web预测 |
| * Android Fragment 你应该知道的一切 |
| * C++, Java、JavaScript中的正则表达式 |
| * Spark技术内幕: Shuffle Map Task运算结果的处理 |

| |
|--------------------------------|
| 阅读排行 |
| JPA注解的使用,用于实体 (697) |
| Factory'javax.faces.rend (654) |
| Java实现排序和类型选择 (431) |
| zsj2451-Minimizing max (422) |
| zsj2048-Highways (380) |
| zsj1788-Quad Trees (368) |
| STL数据结构读书笔记 (367) |
| zsj1805-Squadtrees (288) |
| UML建模图解教程知识点 (282) |
| zsj1203-Swordfish(prim (279) |

| |
|-----------------------------|
| 评论排行 |
| zsj2048-Highways (10) |
| zsj2511-Design T-Shirt((2) |
| 词法分析(Java实现)不用: (1) |
| zsj1854-Election(wa) (0) |
| Java基础笔记杂糅&Java (0) |
| 对象锁&类锁 (0) |
| IdentityHashMap&Hashl (0) |
| Eclipse&MyEclipse使用, (0) |
| ClassLoader&Class使F (0) |
| Oracle学习笔记 (0) |

| |
|--|
| 最新评论 |
| 词法分析(Java实现)不用状态机 hulan_baby: 老师是天才, 你是人才。哈哈 |
| zsj2048-Highways piwwwiq: @JOUeu:361170868@qq.com |
| zsj2048-Highways HelKYLE: @piwwwiq:能否留个邮箱, 我发代码给你, 帮我看看? |
| zsj2048-Highways piwwwiq: @JOUeu:额,你有用压缩吗?还有你直接这样parent=b是达不到折叠效果的,因为要判断长度,你... |
| zsj2048-Highways HelKYLE: @piwwwiq:感觉我跟你做法是一样的, 区别就是我的merge函数里面是直接parent=b |
| zsj2048-Highways HelKYLE: @piwwwiq:能否帮我看看代码, 不知道为什么一直超时 |
| zsj2048-Highways piwwwiq: @JOUeu:没有折叠?不好意思,不知道你表达的意思 |
| zsj2048-Highways HelKYLE: 不知道为什么我的代码没有折叠就一直TLE。。。. |
| zsj2048-Highways HelKYLE: @l631068264:你用的是vc6.0吧 |
| zsj2048-Highways piwwwiq: @l631068264:额...我用的vs2010,没有这方面的问题,可能是vs2010自动吧int... |

Java代码 ☆

```
01. package model;
02.
03. import java.io.Serializable;
04.
05. import javax.persistence.Entity;
06. import javax.persistence.Table;
07. import javax.persistence.UniqueConstraint;
08.
09. @Entity
10. @Table(name = "tb_contact", schema = "test", uniqueConstraints = {
11.     @UniqueConstraint(columnNames = {"name", "email" }),
12.     @UniqueConstraint(columnNames = {"col1", "col2" })
13. })
14. public class ContactEO implements Serializable {
15.
16.     private Long id;
17.
18.     private String name;
19.
20.     private String email;
21.
22.     private String col1;
23.
24.     private String col2;
25.
26. }
```

以上的@Table注释表示指定数据库名“test”，表名为“tb_contact”，并创建了两组唯一索引。

3、@Column

@Column标记表示持久化属性映射表中的字段。此标记可以标注在Getter方法或属性前。如标注在属性前

Java代码 ☆

```
01. public class ContactEO implements Serializable {
02.
03.     @Column(name = "name")
04.     private String name;
05.
06.     public String getName() {
07.         return name;
08.     }
09.
10.     public void setName(String name) {
11.         this.name = name;
12.     }
13. }
```

或者标注在Getter方法前。

Java代码 ☆

```
01. public class ContactEO implements Serializable {
02.
03.
04.     private String name;
05.
06.     @Column(name = "name")
07.     public String getName() {
08.         return name;
09.     }
10.
11.     public void setName(String name) {
12.         this.name = name;
13.     }
14. }
```

- (1) unique 属性表示该字段是否为唯一标识，默认为false。如果表中有一个字段需要唯一标识，则既可以使用@Column标记也可以使用@Table标记中的@UniqueConstraint。
- (2) nullable 属性表示该字段是否可以null值，默认为true（允许为null值）。
- (3) insertable 属性表示在使用“INSERT” SQL脚本插入数据时，是否需要插入该字段的值。
- (4) updatable 属性表示在使用“UPDATE”脚本插入数据时，是否需要更新该字段的值。insertable和updatable属性一般多用于

只读属性，例如主键和外键等。这些字段值通常是自动生成的。

(5) **columnDefinition** 属性表示创建表时，该字段创建的SQL语句，一般用于通过Entity生成表定义时使用。

(6) **table** 属性表示当映射多个表时，指定表中的字段。默认值为主表的表名。

(7) **length** 属性表示该字段的长度，当字段的类型为varchar时，该属性才有效，默认为255个字符。

(8) **precision** 属性和**scale** 属性表示精度，当字段类型为double时，**precision**表示数值的总长度，**scale**表示小数点所占的位数。

示例一、

Java代码 ☆

```
01. private String name;
02.
03. @Column(name = "name", nullable=false, length=512)
04. public String getName() {
05.     return name;
06. }
```

生成的SQL脚本为

Java代码 ☆

```
01. CREATE TABLE contact(
02.     id integer not null,
03.     name varchar(512) not null,
04.     primary key(id)
05. );
```

示例二、为double型指定精度为12位，小数点位数为2位。

Java代码 ☆

```
01. private BigDecimal monthlyIncome;
02.
03. @Column(name="monthly_income", precision=12, scale=2)
04. public BigDecimal getMonthlyIncome() {
05.     return monthlyIncome;
06. }
07.
08. public void setMonthlyIncome(BigDecimal monthlyIncome) {
09.     this.monthlyIncome = monthlyIncome;
10. }
```

生成的SQL脚本为

Java代码 ☆

```
01. CREATE TABLE contact(
02.     id integer not null,
03.     monthly_income double(12,2),
04.     primary key(id)
05. );
```

示例三、自定义生成CLOB类型字段的SQL语句

Java代码 ☆

```
01. @Column(name = "contact_name", columnDefinition=" clob not null" )
02. private String name;
03.
04. public String getName() {
05.     return name;
06. }
07.
08. public void setName(String name) {
09.     this.name = name;
10. }
```

生成的SQL脚本为

Sql代码 ☆

```
01. CREATE TABLE contact(
02.     id integer not null,
03.     contact_name <strong>clob(200) not null,</strong>
04.
05. )
```

```
06.
07.
08.
09.
10.
11.
12.
13.
14.
15.
16.     primary key(id)
17. );
```

其中加粗的部份为columnDefinition属性设置的值。若不指定该属箭，通常使用默认的类型建表，若此时需要自定义建表的类型时，可以在该属性设置。

可持久化的数据类型

| 分类 | 类型 |
|--------------------------|--|
| Java的基本数据类型 | byte、int、short、long、boolean、char、float、double |
| Java的基本数据类型对应的封装类 | Byte、Int、Short、Long、Boolean、Character、Float、Double |
| 字节和字符型数组 | byte[]、Byte[]、char[]、Character[] |
| 大数值类型 | java.math.BigInteger java.math.BigDecimal |
| 字符串类型 | java.lang.String |
| 日期时间类型 | java.util.Date java.util.Calendar java.sql.Date java.sql.Time java.sql.Timestamp |
| 枚举型 | 用户自定义的枚举型 |
| Entity类型 | 标注为@Entity的类 |
| 包含Entity类型的集合Collection类 | java.util.Collection java.util.Set java.util.List java.util.Map |
| 嵌入式（embeddable）类 | |

Java数据类型与数据库中的类型转换是由JPA实现框架自动转换的，所以不同的JPA实现框架转换的规则也不太一样。

例如MySQL中，varchar和char类型都转化为String类型。Blob和Clob类型可以转化成Byte[]型。由于类型转化是JPA底层来实现的，这就遇到一个问题，很有可能在将表中的数据转换成Java的数据类型时出现异常。

我们知道对于可以持久化的Java类型中，即可以映射基本的数据类型，如byte、int、short、long、boolean、char、float、double等，也可以映射成 Byte、Int、Short、Long、Boolean、Character、Float、Double类型。那么选择哪种类型比较合适呢？

举下面的例子进行说明。

Sql代码 ☆

```
01. CREATE TABLE contact(
02.     id integer not null,
03.     monthly_income double(12,2),
04.     primary key(id)
05. );
```

对于表字段id，它的值不能为null，所以映射成int型和Integer型都是可以的。

但对于表字段monthly_income来说，它的值可能为null。当为null时，若此时java的Entity类的对应属性的类型int，则将一个null值转化成int型必定产生转换异。但此时java的Entity类对应的属性为Integer，它是一个对象，对象可以为null，所以不会产生问题。

4、@Basic

在默认情况下，Entity中属箭加载方式都是即时加载（EAGER）的，当Entity对象实例化时，就加载了实体中相应的属性值。

但对于一些特殊属箭，比如大文本型text、字节流型blob型的数据，在加载Entity时，这些属性对应的数据量比较大，有时创建实体时如果也加载的话，可能造成资源严重占用。那么就可以为这些特殊的实体属性设置加载方式为惰性加载（LAZY）

- (1) `fetch`属性表示获取值的方式，它的值定义的枚举型，可选值为`LAZY`、`EAGER`。其中`EAGER`表示即时加载、`LAZY`表示惰性加载。默认为即时加载。
- (2) `optional`表示属性是否可以`null`，不能用于`java`基本数据类型（`byte`、`int`、`short`、`long`、`boolean`、`char`、`float`、`double`）。
- 如：

Java代码 ☆

```
01. @Basic(fetch=FetchType.LAZY)
02. @Column(name = "contact_name",columnDefinition=" clob not null" )
03. private String name;
04.
05. public String getName() {
06.     return name;
07. }
08.
09. public void setName(String name) {
10.     this.name = name;
11. }
```

5、@Id

主键是实体的唯一标识，调用`EntityManager`的`find`方法，可以获得相应的实体对象。每一个实体类至少要有一个主键（`Primary key`）。

一旦使用`@Id` 标记属性为主键，该实体属性的值可以指定，也可以根据一些特定的规则自动生成。这就涉及另一个标记`@GeneratedValue` 的使用。

- `@GeneratedValue` 标注有以两个属性：
- (1) `strategy` 属性表示生成主键的策略，有4种类型，分别定义在枚举型`GenerationType`中，其中有`GenerationType.TABLE`、`GenerationType.SEQUENCE`、`GenerationType.IDENTITY`、`GenerationType.AUTO` ,其中，默认为`AUTO`，表示自动生成。
- (2) `generator` 为不同策略类型所对应的生成规则名，它的值根据不同的策略有不同的设置。
- (3) 能够标识为主键的属性类型，有如表5-2所列举的几种。

| 分类 | 类型 |
|-------------------|-----------------------------------|
| Java的基本数据类型 | byte、int、short、long、char |
| Java的基本数据类型对应的封装类 | Byte、Integer、Short、Long、Character |
| 大数值类型 | java.math.BigInteger |
| 字符串类型 | java.lang.String |
| 时间日期型 | java.util.Date java.sql.Date |

`double`和`float`浮点类型和它们对应的封装类不能作为主键，这是因为判断是否唯一是通过`equals`方法来判断的，浮点型的精度太大，不能够准确地匹配。

例一，自增主键。在不同的数据库，自增主键的生成策略可能有所不同。例如MySQL的自增主键可以通过`IDENTITY`来实现，而Oracle可能需要创建`Sequence`来实现自增。JPA的实现将会根据不同的数据库类型来实现自增的策略。

Java代码 ☆

```
01. @Id
```

```

02. @GeneratedValue(strategy = GenerationType.AUTO)
03. public Long getId() {
04.     return id;
05. }
06.
07. public void setId(Long id) {
08.     this.id = id;
09. }

```

例二，表生成器。将当前主键的值单独保存到一个数据库的表中，主键的值每次都是从指定的表中查询来获得，这种生成主键的方式也是很常用的。这种方法生成主键的策略可以适用于任何的数据库，不必担心与同数据库不兼容造成问题。

配置的Customer类

Java代码 ☆

```

01. package model;
02.
03. import java.io.Serializable;
04.
05. import javax.persistence.Basic;
06. import javax.persistence.Column;
07. import javax.persistence.Entity;
08. import javax.persistence.GeneratedValue;
09. import javax.persistence.GenerationType;
10. import javax.persistence.Id;
11. import javax.persistence.Table;
12. import javax.persistence.TableGenerator;
13.
14. @Entity
15. @Table(schema = "open_jpa", name = "customer")
16. public class Customer implements Serializable
17. {
18.     private static final long serialVersionUID = -8480590552153589674L;
19.
20.     private Integer id;
21.
22.     private String name;
23.
24.     @Id
25.     @GeneratedValue(strategy = GenerationType.TABLE, generator = "customer_gen")
26.     @TableGenerator(schema = "open_jpa",
27.         name = "customer_gen",
28.         table = "tbl_generator",
29.         pkColumnName = "gen_name",
30.         pkColumnValue = "CUSTOMER_PK",
31.         valueColumnName = "gen_value",
32.         allocationSize = 1,
33.         initialValue = 0)
34.     public Integer getId()
35.     {
36.         return id;
37.     }
38.
39.     public void setId(Integer id)
40.     {
41.         this.id = id;
42.     }
43.
44.     @Basic
45.     @Column(name = "name")
46.     public String getName()
47.     {
48.         return name;
49.     }
50.
51.     public void setName(String name)
52.     {
53.         this.name = name;
54.     }
55. }

```

- (1) 在Entity标记的主键的位置，指定主键生成策略为“GenerationType.TABLE”。
- (2) 指定生成主键策略的名称，例如这里命名为“generator = “customer_gen””。
- (3) 使用@TableGenerator标定义表生成策略的具体设置：

Java代码 ☆

```
01. @TableGenerator(schema = "open_jpa",
02.                 name = "customer_gen",
03.                 table = "tbl_generator",
04.                 pkColumnName = "gen_name",
05.                 pkColumnValue = "CUSTOMER_PK",
06.                 valueColumnName = "gen_value",
07.                 allocationSize = 1,
08.                 initialValue = 0)
```

可以看到数据生成tbl_generator 的结构及添加的数据:

Sql代码 ☆

```
01. CREATE TABLE `tbl_generator` (
02.   `GEN_NAME` varchar(255) NOT NULL,
03.   `GEN_VALUE` bigint(20) default NULL,
04.   PRIMARY KEY (`GEN_NAME`)
05. ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
06. INSERT INTO tbl_generator VALUES ('CUSTOMER_PK', 1);
```

使用Oracle的sequence方式生成ID值。

Java代码 ☆

```
01. @Id
02. @GeneratedValue(strategy = GenerationType.SEQUENCE,generator="payablemoney_seq")
03. @SequenceGenerator(name="payablemoney_seq", sequenceName="seq_payment")
```

@SequenceGenerator定义

Java代码 ☆

```
01. @Target({TYPE, METHOD, FIELD})
02. @Retention(RUNTIME)
03. public @interface SequenceGenerator {
04.     String name();
05.     String sequenceName() default "";
06.     int initialValue() default 0;
07.     int allocationSize() default 50;
08. }
```

name属性表示该表主键生成策略的名称，它被引用在@GeneratedValue中设置的“generator”值中。
sequenceName属性表示生成策略用到的数据库序列名称。
initialValue表示主键初识值，默认为0。
allocationSize表示每次主键值增加的大小，例如设置成1，则表示每次创建新记录后自动加1，默认为50。

上一篇 JSF消息FacesMessage的使用
下一篇 重定向与转发

主题推荐 jpa 类 interface 服务器 嵌入式

猜你在找

- | | |
|----------------------------------|------------------------------|
| Web开发之新闻发布系统详解 | Spring渲染Velocity模版实例 |
| 开放2013 | ThreadLocal 与线程池 |
| gsoap 访问c# webservice 返回数据集的示例小结 | 关于jdk7中的新语法带String的switch |
| 通过JAVA反射实现简单的ORM将查询结果封装为对象 | JS快速查找法 |
| 判断字符串是否为Guid格式C# | jQuerycookiejs中cookie设置遇到的问题 |

查看评论

暂无评论

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题

Hadoop

AWS

移动游戏

Java

Android

iOS

Swift

智能硬件

Docker

OpenStack

VPN

Spark

ERP

IE10

Eclipse

CRM

JavaScript

数据库

Ubuntu

NFC

WAP

jQuery

BI

HTML5

Spring

Apache

.NET

API

HTML

SDK

IIS

Fedora

XML

LBS

Unity

Splashtop

UML

components

Windows Mobile

Rails

QEMU

KDE

Cassandra

CloudStack

FTC

coremail

OPhone

CouchBase

云计算

iOS6

Rackspace

Web App

SpringSide

Maemo

Compuware

大数据

apttech

Perl

Tornado

Ruby

Hibernate

ThinkPHP

HBase

Pure

Solr

Angular

Cloud Foundry

Redis

Scala

Django

Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司 提供商务支持
京 ICP 证 070598 号 | Copyright © 1999-2014, CSDN.NET, All Rights Reserved 