JPA规范

## 概述

这份文档是关于在Java EE和Java SE中管理持久化和对象/关系映射的Java API的规范。这

项工作的技术目标是提供一个对象/关系映射机制，通过它，Java应用开发者可以使用域模

型来管理关系数据库。

这个持久化API——连同在本文中定义的查询语言和对象/关系映射元数据——是Enterprise

JavaBeans 3.0必须要支持的。它也被定位于在Java SE中独立地使用。

## 实体

### 概述

实体（entity）是一个轻量的持久化的域对象（domain object）。

实体类是编程工作的主要制品。实体类可以使用辅助类，它们起助手类的作用或者被用于表示实体的状态。

### 实体类的需求

实体类必须用Entity注解进行标注或者在XML描述符中将它声明为实体。

实体类必须有一个无参数的构造方法，而且必须为public或protected。在这之外，实体类

可以有其它的构造方法。

实体类必须是一个顶层类（top-level class）。枚举和接口不能作为实体。

实体类不能用final修饰。实体类的所有方法和持久实例变量都不能是final的。

如果一个实体的实例作为脱管对象（detached object）被传值（passed by value，如，通

过远程接口），那么这个实体类必须实现Serializable接口。

实体支持继承，多态关联和多态查询。

抽象类和具体类都可以作为实体。实体类可以扩展非实体类和实体类，非实体类也可以扩展

实体类。

### 主键和实体标识

每个实体都必须要有一个主键。

主键必须定义在实体层次（entity hierarchy）的根（root）实体，或实体层次的映射超类

（mapped superclass）上。一个实体层次只能定义一个主键。

简单（即，非组合的）主键必须对应实体的一个持久化字段或属性。Id注解被用来指示简

单主键。见9.1.8节。

复合主键要么对应单个持久化字段/属性，要么对应一组这样的字段或属性（见下文说明）。

要表示一个复合主键，必须定义一个主键类。复合主键通常出现在从遗留数据库进行的映射

中，这些遗留数据库的主键由多个列构成。EmbeddedId和IdClass注解用于指示复合主键。

见9.1.14和9.1.15节。

主键（或者复合主键中的字段或属性）应为下列类型之一：Java基本类型；基本类型的包

装类；java.lang.String；java.util.Date；java.sql.Date。然而，一般说来，不应该为

主键使用近似数类型（例如，浮点数）。使用其它类型作为实体主键会失去可移植性。如果

使用（自动）生成的主键，则只有整形是可移植的。如果用java.util.Date作为主键字段

或属性，则时态类型（temporal type）应该指定为DATE。

主键类的访问类型（基于字段还是属性）由包含该主键类的实体的访问类型决定。

### 实体关系

实体之间的关系有:一对一,一对多，多对一，多对多，关系是多态的。

如果两个实体之间存在关联，参照实体(referencing entity)上对应的持久化属性或实例变

量上必须应用下列关系建模注解之一：OneToOne, OneToMany, ManyToOne, ManyToMany。

对于未指定目标类型的关联（例如，对集合没有使用Java泛型），则有必要指定作为关系

目标的实体

（译注：如果实体A参照/引用实体B，则实体A为“参照实体”，实体B为“被参照实

体”）

这些Java注解“镜像”了关系数据库建模中的通常做法，用这些注解进

行对象/关系映射完全可以采用缺省的方式，以便快速开发。

关系可以是双向或单向的，双向关系同时具有持有端（owning side）和反向端（inverse side），单向关系只有一个持有端，关系的持有端觉得对数据库的更新。

以下规则应用于双向关系：

双向关系中的反向端必须通过oneToOne，OneToMany或ManyToMany注解的mappedBy元素指向他的持有端，mappedBy元素指出实体中作为关系的主控端的属性/字段

一对多/多对一双向关系中的‘多’端必须是持有端,因此不能再ManyToOne注解中使用mappedBy注解

对于一对一双向关系，包含对应的外间（foreign key）的那一端是持有端。

对于多对多双向关系，任一端都可以作为持有端

关系模型注解限制对cascade=REMOVE的使用。cascade=REMOVE只应该用在OneToOne或

OneToMany关系上，对其它类型的关系应用cascade=REMOVE是不可移植的。

其它映射注解（如，数据库表和列的注解）可以用来覆盖或进一步明确缺省的映射（2.1.8

节中描述缺省映射）。例如，可以对单向一对多映射使用外键映射。这种模式级别（

schema-level）的映射注解必须在关系的持有端指定。任何此类覆盖都必须和已指定的关系

模型注解保持一致。例如，如果指定了多对一的关系，就不允许在关系的外键上指定唯一键

约束。

持久化提供者负责处理关系的对象——关系映射，包括从数据库中加载和写回到数据库（根

据实体类的元数据），以及数据库中指定的关系的参照完整性（如，外键约束）。

（TODO: 原文是斜体、段落缩进）维护运行期关系的完整性是应用程序的责任，例如，当应

用程序在运行期间更新关系时，要确保双向关系的“一”和“多”端相互保持一致。

对于实体的多值关系，如果在数据库中没有获取到相关联的实体。持久化提供者应返回一个

空集合作为关系的值。

### 关系映射

#### 双向OneToOne关系

假设:

实体A引用实体B的单个实例

实体B引用实体A的单个实例

实体A被指定为关系的持有者。

应用以下映射缺省:

实体a被映射到表A

实体B被映射到表B

表A含有一个参照表B的外键，外键的列名是按如下顺序拼接的字符串：实体A的关系属性或字段的名称，‘\_’，表b的主键名，外键的类型和表b的主键类型相同，并且在外键上有个唯一键约束。

例如:

@Entity

public class Employee {

private Cubicle assignedCubicle;

@OneToOne

public Cubicle getAssignedCubicle() {

return assignedCubicle;

}

public void setAssignedCubicle(Cubicle cubicle) {

this.assignedCubicle = cubicle;

}

...

} @

Entity

public class Cubicle {

private Employee residentEmployee;

@OneToOne(mappedBy="assignedCubicle")

public Employee getResidentEmployee() {

return residentEmployee;

}

public void setResidentEmployee(Employee employee) {

this.residentEmployee = employee;

}

...

}

在这个例子中：

实体Employee引用实体Cubicle的单个实例。

实体Cubicle引用实体Employee的单个实例。

实体Employee是关系的持有端。

将应用以下映射缺省：

实体Employee被映射到表EMPLOYEE。

实体Cubicle被映射到表CUBICLE。

表EMPLOYEE中有一个参照CUBICLE表的外键。外键的列名为：ASSIGNEDCUBICLE\_<PK

of CUBICLE>，这里<PK of CUBICLE>是CUBICLE表的主键的名称。外键的类型与CUBICLE表

的主键的类型相同，并且外键上具有唯一约束。