Java私塾《Hibernate4注解》
——系列精品教程

准备工作

- 在Hibernate中使用Annotation，跟以前xml配置的方式相比：

  1：仍然需要cfg.xml

  2：在cfg.xml中需要配置要通过注解来配置的类，例如：

```
<mapping package="test.animals"/>
<mapping class="test.Flight"/>
```

  3：程序里面，原来的new Configuration()的地方，可以变成：new AnnotationConfiguration()，也可以不用改。

  4：可以通过编程的方式来添加要映射的类，例如：

```
new AnnotationConfiguration().addPackage("test.animals")
                .addAnnotatedClass(Flight.class)
```

Hibernate注解简介

传统上，Hibernate的配置依赖于外部 *.hbm.xml文件：数据库映射被定义为一组 XML 映射文件，并且在启动时进行加载。

借助新的 Hibernate Annotation 库，即可一次性地分配所有旧映射文件——一切都会按照您的想法来定义——注解直接嵌入到您的 Java 类中，并提供一种强大及灵活的方法来声明持久性映射。籍由自动代码完成和语法突出显示功能，最近发布的Java IDE也为其提供了有力的支持。

Hibernate Annotation还支持新的 EJB 3 持久性规范。这些规范旨在提供一种标准化的 Java 持久性机制。由于 Hibernate 3 还提供了一些扩展，因此您可以十分轻松地遵从这些标准，并使用 EJB 3 编程模型来对 Hibernate 持久层进行编码。

映射实体

- @Entity，注册在类头上，将一个类声明为一个实体bean(即一个持久化POJO类)。
- @Table，注册在类头上，注解声明了该实体bean映射指定的表（table）。

映射属性—概述

@Id用来注册主属性，@GeneratedValue用来注册主属性的生成策略，@Column用来注册属性，@Version用来注册乐观锁，@Transient用来注册不是属性。

以上的@Id、@GeneratedValue、 @Column 、 @Version，可以用来注册属性，既可以写在Java类的属性上，也可以注册在属性对应的getter上。

@Transient注册在多余的属性或多余的getter上，但是必须与以上的@Column等对应。

映射属性—非主属性--1

- @Column

标识属性对应的字段，示例：@Column(name=“userName")

```
@Column(
    name="columnName";                                  (1)
    boolean unique() default false;                     (2)
    boolean nullable() default true;                    (3)
    boolean insertable() default true;                  (4)
    boolean updatable() default true;                   (5)
    String columnDefinition() default "";               (6)
    String table() default "";                          (7)
    int length() default 255;                           (8)
    int precision() default 0; // decimal precision     (9)
    int scale() default 0; // decimal scale             (10)
```

(1) name 可选,列名(默认值是属性名)
(2) unique 可选,是否在该列上设置唯一约束(默认值false)
(3) nullable 可选,是否设置该列的值可以为空(默认值false)

映射属性—非主属性--2

(4) insertable 可选,该列是否作为生成的insert语句中的一个列(默认值true)
(5) updatable 可选,该列是否作为生成的update语句中的一个列(默认值true)
(6) columnDefinition 可选: 为这个特定列覆盖SQL DDL片段 (这可能导致无法在不同数据库间移植)
(7) table 可选,定义对应的表(默认为主表)
(8) length 可选,列长度(默认值255)
(8) precision 可选,列十进制精度(decimal precision)(默认值0)
(10) scale 可选,如果列十进制数值范围(decimal scale)可用,在此设置(默认值0)

映射属性—主属性--1

- @Id，标识这个属性是实体类的唯一识别的值。

  注意：这个注解只能标注单一列构成的主键，如tbl_grade那种有两个字段组成的联合主键由其他注解标识。

  回忆*.hbm.xml：

```
<id name="uuid">
    <generator class="assigned"/>
    </id>
```

  @Id，只是标识这个属性是主键，但是并没有指出其生成策略，如上例中的assigned就是由程序员指定的生成策略。

  如果仅仅写出@Id，即是使用assigned生成略，如：

```
@Id
@Column
private int uuid;
```

  如果想使用Oracle支持的sequence取主键，必须通过@GeneratedValue来指定生成策略，而由@SequenceGenerator指定如何使用sequence。

映射属性—主属性--2

```
@Id
@Column
@GeneratedValue(
      strategy = GenerationType.SEQUENCE,//使用sequence生成主键
      generator =“generator“//引用下面名为gernator的生成策略
)
@SequenceGenerator(
      name = “generator”,//定义名为generator的生成策略
      allocationSize = 1, //每次sequence加1
      name=“seq_a”//引用名为seq_a的sequence
)
private int uuid;
```

映射属性—乐观锁和不用持久化

- @Version
  标识这个属性用来映射乐观锁的version
- @Transient
  标识这个属性不用持久化

映射属性—复合属性—组件映射

- @Embeddable【小对象的头上】

  标识实体中可以定义一个嵌入式组件(embedded component)。组件类必须在类一级定义@Embeddable注解。

- @Embedded【大对象的属性头上】

  引用定义的小对象。

映射属性—复合属性—复合主键

- @Embeddable【小对象的头上】

  标识实体中可以定义一个嵌入式组件(embedded component)。组件类必须在类一级定义@Embeddable注解。

  注意：如果这个小对象作为复合主键，一定要实现Serializable接口。这并不是注解决定的，而是Hibernate的主键都需要实现Serializable接口。

- @EmbeddedId 【大对象的属性头上】

  引用定义的小对象作为主键。

  注意：不需要再使用@Id注解。

关联关系映射—简介

在hibernate中，支持对象之间的关联关系映射，这样可以减少我们的dao操作，操作一个对象的时候，就可以顺带操作它的关联对象。

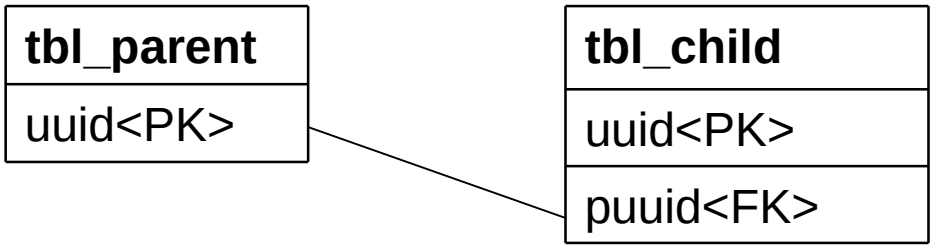
我们知道，hibernate支持三种关联关系，1:1，1:M，M:N。但，这只是对象之间的关系。数据库的设计当然也支持1:1，1:M，M:N三种关系。比如，我们经常说的1:M，就是把1这张表的主键拿到多那边做外键。

但是，很多同学经常迷惑，为什么网上介绍的1:M，比我们讲的还要复杂的多？我们只需要<set>和<many-to-one>，但是网上介绍的还有一种使用<set>和<join>的？

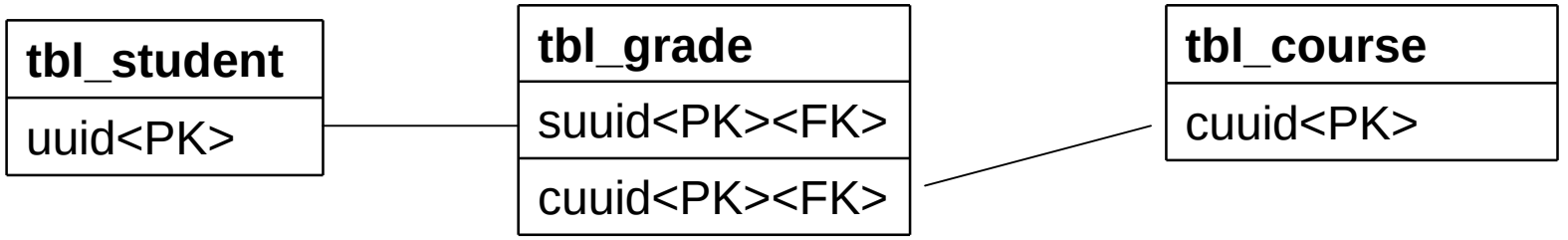
这里我们就不得不提出“数据库设计的降级使用”这个概念了。

关联关系映射—数据库降级使用

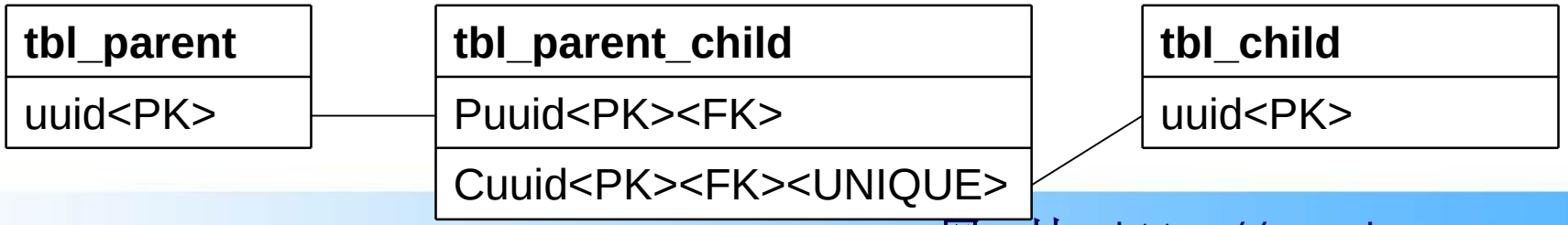
标准的1:M



标准的M:N



将M:N的数据库设计降级为1:M使用



关联关系映射—1:1—共享主键

标准的1:1

XML的配置

主1【tbl_product】：

```
<one-to-one name="info" cascade="all"/>
```

从1【tbl_product_info】：

```
<id name="uuid">
    <generator class="foreign【写死，使用外来生成策略】">
        <param name="property">product【引用自己的Java属性名】
        </param>
    </generator>
</id>
<one-to-one name="product"/>
```

关联关系映射—1:1—共享主键

注解的配置

主1【tbl_product】：

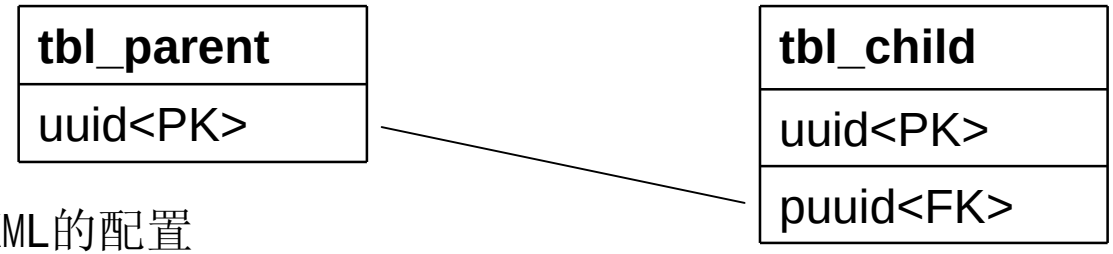
```
@OneToOne(cascade=CascadeType.ALL)
@PrimaryKeyJoinColumn
private ProductInfoModel info;
```

从1【tbl_product_info】：

```
@Id
@Column
@GeneratedValue(generator="copy【引用生成策略】")
@GenericGenerator(name="copy【定义生成策略】",strategy="foreign【写死，使用外来策略】",parameters=@Parameter(name="property",value="product【引用自己的Java属性】"))
private int uuid;
@OneToOne(mappedBy="info【引用对方的Java属性】")
private ProductModel product;
```

关联关系映射—1:M—外键

标准的1:M



XML的配置

1【tbl_parent】：

```
<set name="children">
    <key column="puuid【对方的数据库外键列名】"/>
    <one-to-many class="cn.javass.model.c.ChildModel【对方的Java类名】"/>
</set>
```

多【tbl_child】：

```
<many-to-one name="parent" column="puuid【自己的数据库外键列名】"/>
```

关联关系映射—1:M—外键

注解的配置

1【tbl_parent】：

@OneToMany

@JoinColumn(name="puuid【*对方的数据库外键列名*】")

private Set<ChildModel> children = new HashSet<ChildModel>();

多【tbl_child】：

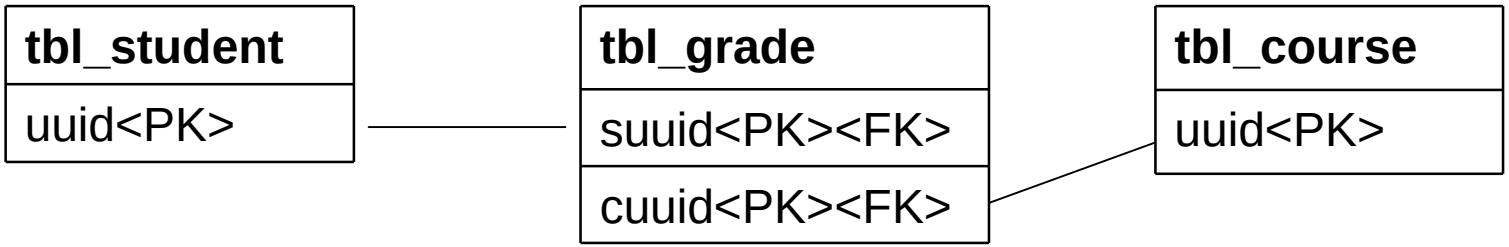
@ManyToOne

@JoinColumn(name="puuid【*自己的数据库外键列名*】")

private ParentModel parent;

关联关系映射—M:N—联接表

标准的1:M



XML的配置

```
<set name="courses" table="tbl_grade【联接表】">
        <key column="suuid【联接表里代表自己的数据库字段名】"/>
        <many-to-many column="cuuid【联接表里代表对方的数据库字段名】"
            class="cn.javass.model.e.CourseMode【对方的类名】l"/>
</set>
```

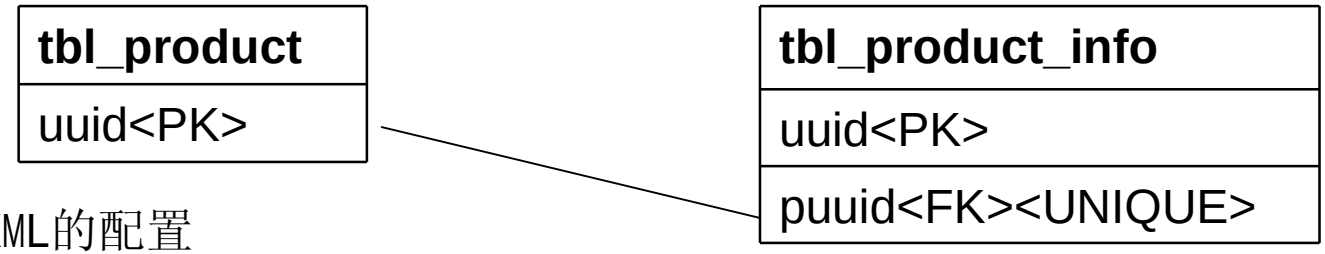
关联关系映射—M:N—联接表

注解的配置

```
@ManyToMany
@JoinTable(
     name="tbl_grade【联接表】",
    joinColumns=@JoinColumn(name="suuid【联接表里代表自己的数据库字段名】
     "),
     inverseJoinColumns=@JoinColumn(name="cuuid 【联接表里代表对方的数据
     库字段名】")
)
private Set<CourseModel> courses = new HashSet<CourseModel>();
```

关联关系映射—1:1—引用外键

标准的1:M



XML的配置

主1【tbl_product】：

<one-to-one name="info" foreign-key="puuid【对方的数据库外键列名】"
cascade="all"/>

从1【tbl_product_info】：

<many-to-one name="product" column="puuid【自己的数据库外键列名】"
unique="true【写死】"/>

关联关系映射—1:1—引用外键

注解的配置

主1【tbl_product】：

@OneToOne(cascade=CascadeType.*ALL,mappedBy="product【对方的Java类属性名】"*)

private ProductInfoModel info;

从1【tbl_product_info】：

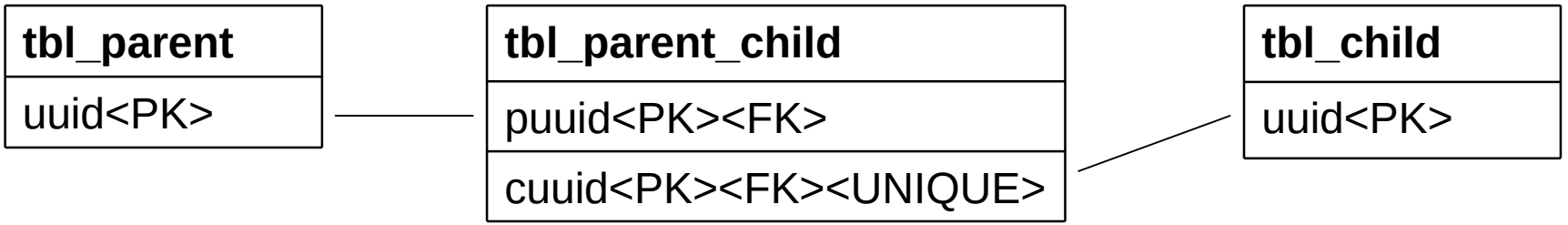
@OneToOne

@JoinColumn(name="puuid【自己的数据库外键列名】")

private ProductModel product;

关联关系映射—1:M—联接表

标准的1:M



XML的配置

1【tbl_parent】：

```
<set name="children" table="tbl_parent_child【联接表】">
    <key column="puuid【联接表里代表自己的数据库列名】"/>
    <many-to-many column="cuuid【联接表里代表对方的数据库列名】"
        unique="true【写死】"
  class="cn.javass.model.d.ChildModel【对方的Java类名】"/>
    </set>
```

关联关系映射—1:M—联接表

```
<join table="tbl_parent_child【联接表】">
        <key column="cuuid【联接表里代表自己的数据库列名】"/>
        <many-to-one name="parent" column="puuid【联接表里代表对方的数据库列名】" unique="true【写死】"/>
</join>
```

关联关系映射—1:M—联接表

注解的配置

1【tbl_parent】：

@OneToMany(mappedBy="parent【对方的Java类属性名 】")

private Set<ChildModel> children = new HashSet<ChildModel>();

多【tbl_child】：

@ManyToOne

@JoinTable(

name="tbl_parent_child【联接表】",

joinColumns=@JoinColumn(name="cuuid【联接表里代表自己的数据库字段名】"),

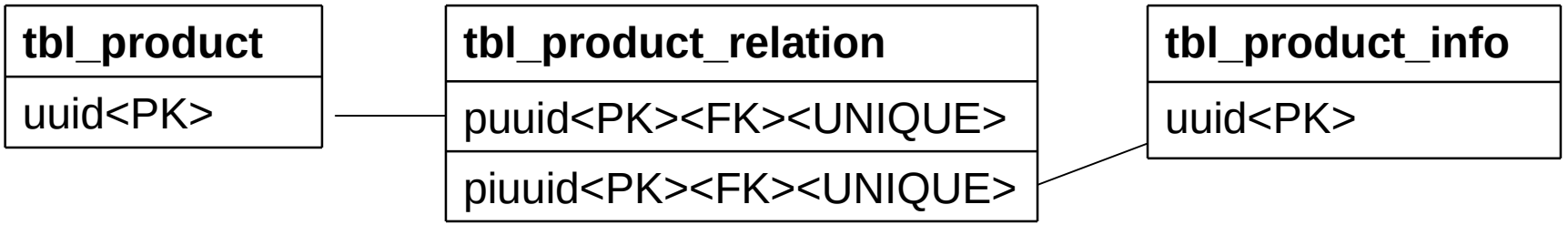
inverseJoinColumns=@JoinColumn(name="puuid【联接表里代表对方的数据库字段名】")

)

private ParentModel parent;

关联关系映射—1:1—联接表

标准的1:M



XML的配置

1【tbl_product】：

```
<join table="tbl_product_relation【联接表】">
        <key column="puuid【联接表里代表自己的列名】"/>
        <many-to-one name="course【自己的Java属性名】" column="cuuid【联接表里代表对方的列名】" unique="true【写死】"/>
 </join>
```

关联关系映射—1:1—联接表

注解的配置

1【tbl_product】：

```
@ManyToOne
@JoinTable(
    name=" tbl_product_relation 【联接表】",
    joinColumns=@JoinColumn(name="suuid【联接表里代表自己的列名】"),
    inverseJoinColumns=@JoinColumn(name="cuuid【联接表里代表对方的列名】
    ",unique=true【写死】)
)
private CourseModel course;
```

二级缓存

- @Cache示例

  定义在实体类上，示例如下：

```
@Entity
@Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
public class Forest { ... }
```