

# 基于 Annotation 数据结构自动生成的研究与实现

王雪梅<sup>1</sup>, 李彩虹<sup>1</sup>, 慕富强<sup>2</sup>, 李 廉<sup>1</sup>

(1 兰州大学 信息学院 甘肃 兰州 730000; 2 上海易可智联公司, 上海 201315)

**摘 要:** 为了使整个数据库设计更加工程化、规范化和便捷化, 使用 Annotation 注解技术实现数据库物理结构的自动生成, 使之从数据库设计文档自动生成关系型数据库结构代码的方法, 从而大大缩减了相关数据库开发的周期, 提高了数据库开发的效率。

**关键词:** 数据库结构; 自动生成; Annotation; java 反射

**中图分类号:** TP311.13

**文献标识码:** A

**文章编号:** 1000-7180(2017)01-0114-05

## Annotation-Based Database Structure Automatically Generated Development

WANG Xue-mei<sup>1</sup>, LI Cai-hong<sup>1</sup>, MU Fu-qiang<sup>2</sup>, LI Lian<sup>1</sup>

(1 College of Information, Lanzhou University, Lanzhou 730000, China;

2 Shanghai E-connections Company, Shanghai 201315, China)

**Abstract:** In order to make the entire database design more engineering, standardization and facilitation, as used herein, annotation technology to automatically generate the physical structure of the database, so that the method relational database structure of the code generated automatically from a database design document, thus greatly reducing the related database development the cycle, improve the efficiency of database development.

**Key words:** database structure; automatically generated; Annotation; Java reflection

### 1 引言

数据库设计自动化能有计算机参与中间大部分开发活动,减少软件人员的大量、复杂的重复劳动,从而有效地提高数据库系统开发速度,缩短开发周期<sup>[1]</sup>。但在开发大型信息管理系统软件的过程中,存在大批数据库表的建立和设计,这就使得工作变得非常繁琐,复杂,而且很耗时,如何快速高效地建立大批数据库表并提高开发效率已经成为数据库技术发展所关注的问题。目前,已经有很多关于数据库设计的工具,包括有 PowerDesigner、ER Studio、CA 的 ER-Win、Microsoft Visio 等计算机辅助数据库设计的工具,这些工具都在不同程度上简化方便了数据库设计各阶段<sup>[2]</sup>的开发,但由于其大多数的设计工具都是收费的,且在由数据库逻辑阶段转化到物理阶段的过程

中没有更为简化的操作。因此相关数据库设计中数据结构自动生成问题有待解决和研究。

数据库逻辑设计旨在根据数据库的逻辑结构来选定关系数据库(RDBMS),并设计和实施数据库的存储结构、存取方式。在实际应用中,数据库的物理结构就是 SQL 语句。陈金水等学者<sup>[3]</sup>提出使用数据表结构来定义数据表逻辑结构和物理结构之间的关系,完成数据表物理结构的自动生成;文献<sup>[4]</sup>提出从设计文档中读取信息,从而实现数据库物理结构自动创建的方法。这些方法虽然都有效地解决了物理结构自动创建的问题,但却不是最佳最好的方法。

本文利用 Java5 的新特性 Annotation 注释技术、java 反射技术和 JavaBean 实体封装技术,及数据库物理结构的 SQL 表现语言,实现数据库表的表结构代码生成器,从而使数据库的物理结构自动快

速创建物理数据库管理系统,大大节约开发时间和精力,提高开发效率。

## 2 相关技术

### 2.1 java 反射技术

JAVA 反射机制<sup>[5]</sup>是在运行状态中,对于任意一个类,都能够知道这个类的所有属性和方法;对于任意一个对象,都能够调用它的任意一个方法和属性;这种动态获取的信息以及动态调用对象的方法的功能称为 java 语言的反射机制。Java 提供了一套 API,用来对 JAVA 虚拟机中的类、接口、对象进行反射,对系统框架的设计来说,它提供了十分有用的功能。具体功能如下:

- (1)获取一个类的信息(类名、属性、方法、构造函数等);
- (2)创建一个类的实例,而类的名字在运行时才决定;
- (3)调用一个对象的方法,此方法的名字运行时才决定;
- (4)获取或设置一个对象域的值,该域可以在运行时才决定。

### 2.2 Annotation 注释技术

从 JDK 5 开始,Java 增加了对元数据(MetaData)的支持,也就是 Annotation 注释。Annotation<sup>[6-7]</sup>提供了一种为程序元素设置元数据的方法,程序元素包括修饰包、类、构造器、方法、成员变量、参数、局部变量,从某些方面来看,Annotation 就像修饰符一样,可用于程序元素的声明,这些信息被存储在 Annotation 的“name = value”对中。同时,Annotation 是一个接口,程序可以通过反射来获取指定程序元素的 Annotation 对象,然后通过 Annotation 对象来取得注释里的元数据,且不会影响程序代码的执行,无论增加、删除 Annotation,代码都始终如一地执行。如果希望让程序中的 Annotation 在运行时起一定的作用,只有通过某种配套的工具对 Annotation 中的信息进行访问和处理。

元 Annotation 指的是修饰注解的注解。常见的元 Annotation 有 @Retention, @Target, @Documented, @Inherited。通常我们在自定义一个新的注解时,要使用的元 Annotation 是 @Retention 和 @Target,分别用来标注定义该注解的作用域和修饰程序单元的类型。由于篇幅限制,在这里仅介绍定义新注解时要用到的两个元注解。

@Retention 包含一个 RetentionPolicy 类型的

成员变量,其值可以是如下 3 个:

- (1)RetentionPolicy. Class: 默认值,编译器将 Annotation 存储于 class 文件中,但运行 Java 程序时,JVM 不会保留 Annotation,即只存储但不参与程序运行;
- (2)RetentionPolicy. RUNTIME: 编译器将把 Annotation 存储于 class 文件中,运行 Java 程序时,JVM 也会保留 Annotation,即存储且参与程序运行;
- (3)RetentionPolicy. SOURCE: 编译器不会存储 Annotation 到 class 文件中,运行 Java 程序时 JVM 也不会保留 Annotation,即不存储不保留。

需要通过反射获取注释信息,所以就需要使用 value 属性值为 RetentionPolicy. RUNTIME 的 @Retention。

- @Target 的 value 值有如下几个:
- (1)ElementType. ANNOTATION\_TYPE: 指定该策略的 Annotation 只能修饰 Annotation;
  - (2)ElementType. CONSTRUCTOR: 只能修饰构造器;
  - (3)ElementType. FIELD: 只能修饰成员变量;
  - (4)ElementType. LOCAL\_VARIABLE: 只能修饰局部变量;
  - (5)ElementType. METHOD: 只能修饰方法定义;
  - (6)ElementType. PACKAGE: 只能修饰包定义;
  - (7)ElementType. PARAMETER: 可以修饰参数;
  - (8)ElementType. TYPE: 可以修饰类、接口(包括注释类型)或枚举定义。

该语言特性的 Java 实现使用一种类似于接口的结构和 Java 语言语法的一些特殊扩展,但不同于常规接口,注释必须在定义中使用关键字 @interface。同样不同于常规接口的是,注释只能定义不带参数且只返回简单值(基本类型、String、Class、enum 类型、注释,以及任意这些类型的数组)的“方法”。这些“方法”是与注释关联的值的名称。如下列所示:

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface Table{
    String name() default AnnotationUtil.AUTO; // 表名
    String comment() default""; // 表注释
}
```

### 3 生成方法

由于数据库中的数据定义、关系创建及约束定义等是由专门的语言——数据库标准语言 SQL 构成的,所以建立数据库逻辑结构和物理结构之间的对应关系的实质就是依据标准 SQL 语言,创建与 SQL 语言相对应的 Annotation 注解.本文采用 Mysql 数据库作为数据存储的软件工具,分四步实现.

第一步:首先分析数据库表是如何和 SQL 语言之间一一对应的,找到这种连接的关系,即数据库逻辑结构到数据库物理结构的映射关系,并创建与之对应的 Annotation 注解.

在数据库中数据表是用来存储具体的数据,便于供开发的应用系统操作具体相应的数据.其以创建一条数据表的 SQL 语句为例(该语句包含数据库 SQL 语句的所有条件):

create table + 表名(列的名称 列的数据类型 列的约束);

从该语句中可以看出要在具体的数据库工具软件中创建一个数据表,就要明确知道数据表的表名、列的名称、列的数据类型及列的约束,而列的约束就是给一定的条件限制,使它只能为某种状态.以下是数据库建表对列的所有约束条件(可根据实际需要选择):

(1)非空约束<not null>(就是在创建名称时不能为空,否则得报错);

(2)指定默认约束<default>(就是在创建名称时给定一个默认的);

(3)检查性约束<check>(就是当满足某个条件时才能够被插入数据);

(4)主键约束<primary key>(每一张表需要一列来让这张表的每条记录不重复,即标示列,主键);

(5)唯一约束<unique>(重复的数据将被数据库所拒绝).

以上是要操作一个数据表的所有必须条件.要实现用 Annotation 注释技术实现数据库表的自动创建,即编写一次 SQL 操作语句,创建具体应用系统的所有数据库表,就要使 Annotation 注释与 SQL 语句的条件对应起来.以下是自定义的注释用来标注其数据表的表名、列名、列类型及列的约束:

```
@Target(ElementType. TYPE)
```

```
@Retention(RetentionPolicy. RUNTIME)
```

```
public @interface GeneratedValue{} //自动递增方式的 annotation 类型
```

本文每个自定义的注释都有 @Target 和 @Retention 元

注释来修饰,以下省略.

```
public @interface Id{} 数据表主键声明的 annotation 类型
```

```
public @interface Index{} 数据表字段是否创建索引声明的 annotation 类型
```

```
public @interface Unique{} 数据表字段是否创建 unique 索引字段的 annotation 类型
```

```
@Target(ElementType. CLASS)
```

```
public @interface Table{} //数据表声明的 annotation 类型
```

```
String name() default AnnotationUtil. AUTO; //表名
```

```
String comment() default ""; //表的注释
```

```
}
```

```
public @interface Column{} //数据表字段声明的 annotation 类型
```

```
String name() default AnnotationUtil. AUTO; //字段名称
```

```
DataType type(); //字段类型
```

```
int length() default 0; //字段长度
```

```
boolean notNull() default true; //是否必填
```

```
String defaultValue() default ""; //默认值
```

```
String comment() default ""; //字段描述
```

```
boolean log() default false; //日志记录是否包含该字段
```

```
}
```

第二步:依据数据库逻辑结构和自定义的 Annotation 注解,使用 Java 代码编程实现对已定义注释的解析类.所谓对 Annotation 注释的解析,也就是对 Annotation 注释的访问.

当在程序元素中使用 Annotation 修饰了类、方法、Field 等成员之后,这些 Annotation 不会自己生效,必须由开发者提供相应的代码来提取并处理 Annotation 信息.而这一过程也就是 Java 的反射机制.利用 java 反射机制获取到相对应的程序元素对象,就可以通过该程序元素的对象调用相关注解的处理方法,获取到具体的注解信息.本文以 AnnotationUtil 为类名,作为对所有注解的访问处理.其主要包含和使用的方法如下:

(1) isAnnotationPresent (Class <? extends Annotation> annotationClass):判断该程序元素上是否存在指定类型的注解,如果存在则返回 true,否则返回 false;

(2) getAnnotation (Class < T > annotationClass):返回该程序元素上存在的指定类型的注解,如果该类型的注解不存在,则返回 null;

(3) Annotation[] getAnnotations():返回该程序元素上存在的所有注解;

若定义的注解有成员变量,且要获取注解中的成员变量值时,直接调用注解对象的"成员变量名()"形式的方法就可以获取到.如上面定义的 @Col-

umn,要获得其成员变量 name 的属性值,可直接使用 column.name()方法获取。

第三步:创建 SQL 语句,在 SQL 语句中充分体现利用 Annotation 注释来访问其相关被注释的信息,使得创建的一条 SQL 语句可用来重复建立大批的数据表,从而节约开发时间,提高效率。其主要代码如下所示(其中 AnnotationUtil 是对相关注解的一个解析工具类,entityClass 是相应实体类的对象)。

```
StringBuffer sql = new StringBuffer("create table if not
exists ").append(tableName).append(" ( ");
for (Field field : AnnotationUtil. GetAnnotationDe-
claredFields(entityClass, Column.class, true)){
    Column column = field. getAnnotation(Column.
class);
    sql.append(" ").append(AnnotationUtil. Get-
ColumnName(field));
    sql.append(" ").append(null == column.type()?"
varchar(50)":AnnotationUtil. GetDbTypeDataString(col-
umn));
    if(column.notNull()){
        sql.append(" not null "); }
    if(field.isAnnotationPresent(GeneratedValue.class)){
        if(AnnotationUtil. GeneratedValues. auto. equals(field.
getAnnotation(GeneratedValue.class). strategy())){
            sql.append(" auto_increment "); } }
    if(! column.defaultValue(). isEmpty()){
        if(String.class == field.getType()){ sql.append("
default ").append(column.defaultValue()).append(""); }
        else{
            sql.append(" default ").append(column.defaultValue
()); } }
    if(! column.comment(). isEmpty()){
        sql.append(" comment ").append(column.comment
()).append(""); }
    sql.append(" , "); }
    if(null != primaryKey){
        sql.append(" primary key ( ").append(primaryKey.
getName()).append(" )"); }
    sql.append(" ) ENGINE=MyISAM ");
    if(! table.comment(). isEmpty()){
        sql.append(" comment ").append(table.comment()).
append(""); }
    sql.append(";");
```

在编程过程中,使用线程安全的 StringBuffer 字符串类来声明和处理字符串类型,用 if 语句判断处理所有列属性约束的条件,最终完成完整规范的 Annotation 注释的 SQL 语句。在实际创建数据库物理结构时,根据数据库具体定义表的属性定义其对应的实体类,并在实体类中充分体现对 Annotation 注释技术的使用,最后依据具体的实体类,调用 SQL 生成数据库的物理结构。

第四步:编写数据实体类。实体类的各个属性与

数据库表中的属性一一对应,并且实体类中的各个属性都有相应的 annotation 注解注释。完成实体类的编写后,就可以执行创建数据表的 Sql 语句,完成逻辑数据表向物理数据表的转换。

## 4 测试

为了验证本文提出的自动创建数据库物理结构生成方法的正确性和有效性,在此以应用系统中的用户信息表为实例,实现其自动创建的数据库物理结构。

开发工具:使用 Myeclipse。MyEclipse,是在 eclipse 基础上加了自己的插件开发而成的功能强大的企业级集成开发环境,且自带 JRE、JDK 和已配置好的 TOMCAT,在使用时只需安全安装 MyEclipse,主要用于 Java、JavaEE 以及移动应用的开发。Myeclipse 的功能非常强大,支持也十分广泛,尤其是对各种开源产品的支持相当不错。

数据库:数据库使用 MySQL。MySQL 是最受欢迎的开源 SQL 数据库管理系统,它由 MySQL AB 开发、发布和支持。MySQL 是一个小巧玲珑的数据库服务器软件,对于中、小型应用系统是非常理想的。除了支持标准的 ANSI SQL 语句外,最重要的是,它还支持多种平台,而在 Unix/Linux 系统上,MySQL 支持多线程运行方式,从而能获得相当好的性能,并属于开发源代码软件。

数据库管理工具:使用 MySQL Front 管理工具。MySQL Front 是一款 MySQL 客户端管理软件,可对 MySQL 数据库实现图形界面管理,实现对数据库的可视化操作,方便进行各种数据库的管理工作。

用户信息表是所有应用管理软件所必须涉及到的基本表。该表反映了管理和操作应用软件系统的所有人员的基本信息,包括登录名称、登录账号、登录密码、用户性别、联系方式等等。其对应的实体类类名定为 JSand\_User,在该实体类中充分体现对本文自定义 Annotation 使用,最后调用本文设计实现的 SQL 创建数据库物理结构语句,执行该 SQL 语句,在 MySQL Front 管理工具中就可看到有图 1 所示的物理结构图。

## 5 结束语

注解是众多引入到 java SE 中的重要语言变化之一,它们可以提供用来完整描述程序所需要的信息,而这些信息是无法用 java 来表达的。本文使用

| 名称                | 类型                           | 空 | 默认值              | 属性     | 备注 |
|-------------------|------------------------------|---|------------------|--------|----|
| 索引 (3)            |                              |   |                  |        |    |
| 主索引               | id                           |   |                  | unique |    |
| Index_JSand_User  | groupid,code,name,pyName,sex |   |                  |        |    |
| Unique_JSand_User | uid,account                  |   |                  | unique |    |
| 字段 (20)           |                              |   |                  |        |    |
| id                | bigint(20)                   | 否 | <auto_increment> | 编号     |    |
| uuid              | varchar(32)                  | 否 |                  | 全局编号   |    |
| ctime             | bigint(20)                   | 否 |                  | 创建时间   |    |
| groupid           | bigint(20)                   | 否 |                  | 所属分组   |    |
| code              | varchar(200)                 | 否 |                  | 分组编码   |    |
| name              | varchar(50)                  | 否 |                  | 用户名称   |    |
| pyName            | varchar(50)                  | 否 |                  | 拼音索引   |    |
| account           | varchar(50)                  | 否 |                  | 登录账户   |    |
| password          | varchar(50)                  | 否 |                  | 登录密码   |    |
| sex               | tinyint(4)                   | 否 |                  | 用户性别   |    |
| tel               | varchar(20)                  | 是 | <空>              | 联系电话   |    |
| imei              | varchar(15)                  | 是 | <空>              | 手机号码   |    |
| qq                | varchar(15)                  | 是 | <空>              | 用户QQ   |    |
| comment           | varchar(512)                 | 是 | <空>              | 用户描述   |    |
| rights            | text                         | 是 |                  | 权限编码   |    |
| attributes        | text                         | 是 |                  | 扩展属性   |    |
| loginIP           | varchar(20)                  | 否 | 0.0.0.0          | 登录IP   |    |
| loginTime         | bigint(20)                   | 否 |                  | 登录日期   |    |
| mtime             | bigint(20)                   | 否 |                  | 修改日期   |    |
| editor            | varchar(50)                  | 否 |                  | 修改用户   |    |

图 1 测试图

Annotation 技术实现对数据库物理结构的自动生成,避免了人工对大批的数据表依次重复进行创建的无用工作,使相关数据库管理系统开发人员可以非常便捷、有效、快速地建立数据库的物理结构。

参考文献:

[1] 刘念祖. 数据库设计中数据库结构的自动生成[J]. 上海第二工业大学学报, 1992,9(2): 50-54.

[2] 陈艳, 张冬姣. 数据库设计[J]. 福建电脑, 2012, 28(2): 109-110.

[3] 陈金水, 钱昆. 数据库物理结构自动生成技术的研究[J]. 电脑开发与应用, 2005,18(4):42-43.

[4] 刘杰. 数据库结构自动生成工具的开发[J]. 计算机时代, 2007,25(1): 52-54.

[5] 檀明, 许强, 李丛. 基于标注和反射机制的 JavaBean 持久化工具类[J]. 微电子学与计算机, 2010, 27(4): 29-32.

[6] 张文胜. 基于 Annotation 的 Spring AOP 日志处理的设计与实现[J]. 价值工程, 2012, 31(32): 207-209.

[7] 杨宇仙, 杨志勇. 用 Annotation 简化 Java 程序的开发[J]. 计算机与现代化, 2008,24(6): 121-122.

作者简介:

王雪梅 女,(1992-),硕士研究生. 研究方向为计算机技术. E-mail:18794842794@163. com.

李彩虹 女,(1973-),博士,高级实验师. 研究方向为计算机教学模式研究、软件设计.

慕富强 男,(1974-),硕士,教授. 研究方向为云计算、无线传感器、人工智能、智能家居.

李 廉 男,(1951-),博士,教授. 研究方向为网格计算、高性能计算及其在 e-Science 系统中的应用、无线传感器网络,以及行为与情感分析、健康关怀等.

(上接第 113 页)

表 1 FPGA 芯片资源使用报告表

| 逻辑器件            | 资源利用率               |
|-----------------|---------------------|
| Slice Registers | 25 700/2443 200(1%) |
| Slice LUTs      | 42 341/122 1600(3%) |
| LUT-FF pairs    | 11 185/56 856(19%)  |

6 结束语

本文通过对时钟共享多线程处理器的研究,采用了一种与其相适应的存储结构,通过对整个处理器的运行分析发现,将数据存储分为 8 个 bank,指令存储分为 4 个 bank,能够有效地解决 PE 内部四路线程取指和八路线程回写以及不同 PE 间数据交互问题,最终的综合结果表明交叉存储的最高频率是 518. 309 MHz. 所以该结构能够保证时钟共享多线程处理器对指令和数据存储访存速率的需求,并有效提高了处理器的速率。

参考文献:

[1] 屈文新, 樊晓桢, 张盛兵. 多核多线程处理器存储技术研究进展[J]. 计算机科学, 2007, 34(4):13-16.

[2] Spracklen L, Abraham S G. Chip multithreading: op-

portunities and challenges[C]// International Symposium on High-performance Computer Architecture, Austin, 2005:248-252.

[3] Hammond L, Hubbert B A, Siu M, et al. The Stanford Hydra CMP[J]. Micro IEEE, 2000, 20(2): 71-84.

[4] Li T, Xiao L, Huang H, et al. PAAG: A polymorphic aArray architecture forgraphics and image processing[C]// International Symposium on Parallel Architectures, IEEE, 2012:242-249.

[5] 刘应天. 时钟共享多线程处理单元的设计与实现[D]. 西安:西安邮电大学, 2015.

[6] 雷晓锋, 李涛. 时钟共享多线程处理器通信机制的设计与实现[J]. 电子技术应用, 2016, 42(3):42-46.

作者简介:

李卯良 男,(1991-),硕士研究生. 研究方向为电路与系统. E-mail:limaoliang\_lml@163. com.

李 涛 男,(1954-),教授. 研究方向为计算机体系结构、计算机图形学.

刘 欢 男,(1991-),硕士研究生. 研究方向为电路与系统.

杨 铮 男,(1991-),硕士研究生. 研究方向为电路与系统.

郭佳乐 女,(1992-),硕士研究生. 研究方向为电路与系统.