任务管理系统模型类的抽象与设计①

许南山，肖银涛，卢 罡

(北京化工大学 信息科学与技术学院，北京 100029)

摘 要：针对以往任务管理系统开发效率低、系统可维护性不强的情况，对任务管理系统模型类进行抽象与设计，分析了模型类的通用性和适用性，详细介绍了模型类抽象与设计的思路和方法。最后把抽象出的模型类应用到现有的任务管理系统中，验证设计思路，改进、优化了现有系统，降低了开发类似系统的成本，对任务管理系统的平台化研究和实现有一定的帮助。

关键词：任务管理系统；模型类；平台化研究；设计模式

Abstract and Design of Task Management System Model Classes

XU Nan-Shan, XIAO Yin-Tao, LU Gang

(School of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China)

Abstract: To solve the problems of low efficiency, weak maintenance in development of task management system, model classes of task management system are abstracted and designed in this paper. The design and implementation of the model classes with versatility and applicability are introduced in detail. The model classes are applied in the existing task management system, which has been verified, improved, and optimized. The method of model classes will reduce cost of developing similar systems. It is also helpful to the research and implementation of task management system platform.

Key words: task management system; model class; platform research; design pattern

1 引言

随着计算机技术的飞速发展，任务管理系统已经应用到各个领域。任务管理系统的普及，使任务管理者和任务执行者之间的责任更加明确、任务管理过程更加清晰有序，大大提高了管理效率[1]。但是，任务管理系统的开发，存在着代码可重用率不高、开发效率低、系统可维护性不强、对需求变更的适应性较弱等问题。这在很大程度上浪费了开发人员的精力，影响了任务管理系统的发展。

本文作者所在的研究室正在致力于任务管理系统平台化的研究和实现。任务管理系统平台化的实现，需要一个.NET 类库支持平台化系统的基本逻辑。鉴于常见的系统架构中都需要模型类，本文以任务管理系统项目为基础，对任务管理系统中的模型类进行抽象和设计，抽象出的模型类可以提高代码的重用性，再次开发任务管理系统、简单的办公自动化系统或 MIS 系统时，可以大大提高开发效率，降低开发成本，增加系统的可维护性和扩展性。对任务管理系统的平台化研究和实现也有所帮助。

2 模型类的抽象与设计

2.1 模型类的通用性分析

任务管理系统开发过程中，三层架构和 MVC 架构是开发中比较常用的两种系统架构。三层架构是把 Web 系统的整个业务划分成表示层（UI），业务逻辑层（BLL）和数据访问层（DAL）。这样的系统架构，层次分明，思路清晰，符合“高内聚，低耦合”的设计思想，系统有更强的扩展性和可维护性[2]。

① 收稿时间:2011-06-27;收到修改稿时间:2011-07-20

MVC（模型 Model-视图 View-控制器 Controller）设计模式，同样是架构级别的设计思想，二者的相同点都是把系统分层，使每个层次的功能尽量独立，降低系统的耦合度[3]。

这两种架构有很多优点，但对于某些需求而言，这两种架构也存在着一定的不足。比如系统需要增加或修改一个功能，有时会导致级联修改的问题，这种修改尤其体现在自上而下的方向，可能需要在整个系统级别上对进行修改和维护。软件开发过程中的需求变更是很常见的，我们应该尽量降低需求变更对整个系统带来的影响。

三层架构和 MVC 架构都需要模型类为系统提供基本的逻辑支持。模型类的抽象与设计主要针对的是三层结构中的业务逻辑层、数据访问层或 MVC 中的模型部分，抽象出的模型类具有更强的通用性，使系统更能适应需求的变化，易于维护和扩展。主要思想是对以往的开发方式进行改进，把类的属性和方法分离成两个部分，从属性中抽象出数据表的通用属性并实现基本的方法，但实现基本方法的代码由通用的数据表操作类完成。通用数据表操作类是对常用方法的抽象。最后我们得到一个通用的数据表类和一个通用的数据表操作类。通用数据表类是一个实体类，它对数据库中的数据表进行抽象。然后采用设计模式中的工厂模式（Factory Pattern），在工厂中生产具体的数据表对象。通用数据表操作类可以对数据表进行增加、删除、修改、查询等操作。

2.2 通用数据表类

具体实现中，通常数据库中的一个数据表对应着一个类，数据表中的字段是类的属性，该类具有增加、删除、修改、查询等一般方法和一些特有的方法。数据库中有多个表，就会对应着多个类。如果对数据库中某个字段进行了修改，那么就要相应的修改对应的类中的属性和涉及到的方法。这样的开发方式在通用性和可维护性上都存在不足，亟待改进。

观察分析多个具体的数据表类发现，它们都是对数据库中具体表到数据表类的映射。分析这些数据表的共性，对它们进行抽象，就可以抽象出通用数据表类。通用数据表类不考虑具体的字段，只是分析数据表应该具有的表名、列名等特征和一些常用的方法。实例化通用数据表类时再给它提供表名、列名等关键信息。通用数据表类具有更高层次的抽象性和更强的可重用性。通用数据表类的类图如图 1 所示：

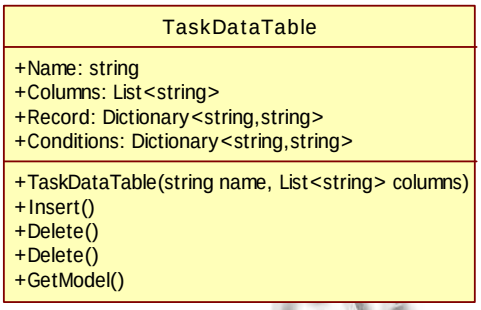

图 1 通用数据表类

通用数据表类是一个实体类，包括表名、列名、记录名、对表进行操作时的条件（主要用于 Where 子句部分）等属性和对数据表进行操作的常用方法。实例化该类时，需要提供表名和列名，构造函数对数据表进行初始化。通用数据表类的基本方法包括：增加、删除、修改、查询，这些方法是通过调用通用数据表操作类来实现的。通用数据表类在进行基本操作时，把自己（this）作为参数调用数据表操作类中的方法，比如 Insert 方法的实现方法如下所示：

```
Insert ()
{
TaskSqlDTOperator.Insert (this);
}
```

需要说明的是，Record 是一条记录，它和数据库中数据表的一行是有区别的。数据表的一行只是一行数据，只有值，而一个 Record 同时包含列名和值，是键值对。Record 的结构是字典（Dictionary），键是 string 型，对应着列名，在一个数据表中，列名是不可以重复的。Record 的值是 string 型。进行 SQL 操作时，任何的数据类型都可以转换成 string 型。实际上 Record 值的类型可以再用一个实体类来对应，这样就可以与其他数据类型对应。但为了控制代码的复杂度、适度的控制代码的扩展性，这样做的意义并不大[4]。

为了编程时使用方便，避免和其他命名空间类似的数据结构同名，本文抽象和设计的任务管理系统的模型类均以 Task 为前缀。

2.3 数据表工厂

数据表工厂（Table Factory 类）用于创建各种具体的数据表实例，只需要给通用数据表的构造函数提

供表名和列名，就可获取相应的数据表实例。实际编程中，程序员只需提供表名，由程序验证数据库中是否存在该表，如果存在，则获取该表的列名即可。如果不存在，则进行相应的异常处理。数据表工厂生成数据表实例的方法如下：

```
public static TaskDataTable GetTable(string tableName)
{
List<string> columns = new List<string>();
string mysql = "select top 1 * from " + tableName;
try
{
//微软封装的 SqlHelper 类，返回一个 DataTable
DataTable dt=SqlHelper.Execute Dataset (Database.ConnectionString, CommandType.Text, mysql).Tables [0];
//获取所有列名
for (int i = 0; i < dt.Columns.Count; i++)
{columns.Add(dt.Columns[i].ColumnName);}
//id 列为主键，自增 1。Insert 时不需要该字段，故移除。
columns.Remove("id");
//实例化 TaskDataTable
TaskDataTable table = new TaskDataTable (tableName, columns);
return table;
}
catch (Exception)
{
//捕获异常，返回 null。在调用本静态方法处需要对返回结果做判断。
return null;
}
}
```

2.4 数据表操作类

对数据表类的操作，主要是执行增加、删除、修改、查询等操作，对不同类型的数据库访问的方法有所区别。解决方法是，定义一个抽象类，针对不同类型的数据库再做相应的实现。类图如下所示：

数据表操作类的功能是一次性生成基本的增加、删除、修改、查询等命令，具体的执行由数据库访问类完成。每个操作都是以 TaskDataTable 为参数，执行操作时，循环遍历 TaskDataTable 的 Record 即可，不需要知道 Record 中条目的具体名字。而 TaskDataTable 的条目也是在实例化时才获取的。这就保证了该类的抽象性，针对不同的数据表，不需要对代码进行改变，由此降低了引入 bug 的风险。

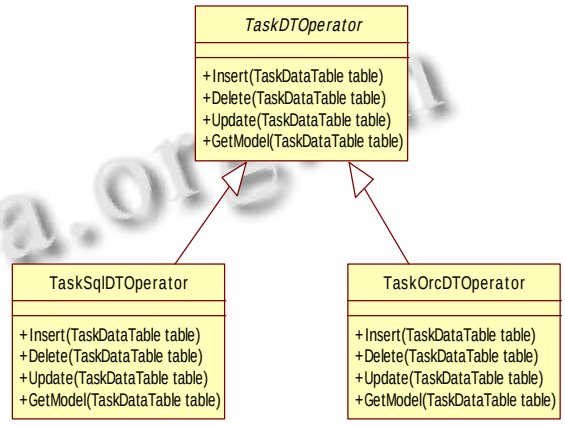

图 2 数据表操作类

这样设计数据表类和数据表操作类更符合“对扩展开放，对修改封闭”的类的设计原则。如果需要增加新的操作，只需要对数据表操作类进行继承或组合，并增加相应的方法即可，这样的改变不会影响系统的其他部分。找出系统中可能需要变化之处，把它们独立出来，不要和那些不需要变化的代码混在一起，可以达到让系统的某部分改变不会影响其他部分的目的，系统具有更强的稳定性。这种思想是设计模式中策略模式（Strategy Pattern）的核心思想[5]。

2.5 数据库访问类

数据库访问类是执行 SQL 命令的类。对于数据库的访问，不同类型的数据库的访问方法是有区别的。我们希望在编写应用系统的时候，能够尽量做到数据库无关，当后台数据库发生变更的时候，不需要对程序做修改或只需要少量修改。

实现上述目的方法是，设计一个数据库访问抽象类，针对具体的数据库，去做具体的实现。设计的方法与数据表操作类相似，如图 3 所示。创建一个 DBHelperFactory 类，这个类可以根据连接字符串来判断使用什么数据库，然后，返回适当的数据库操纵类的实例，从而实现数据库自动切换管理的目的。在实际中，随着数据库类的增加，判断的方法可能会有所

变化[6,7]。

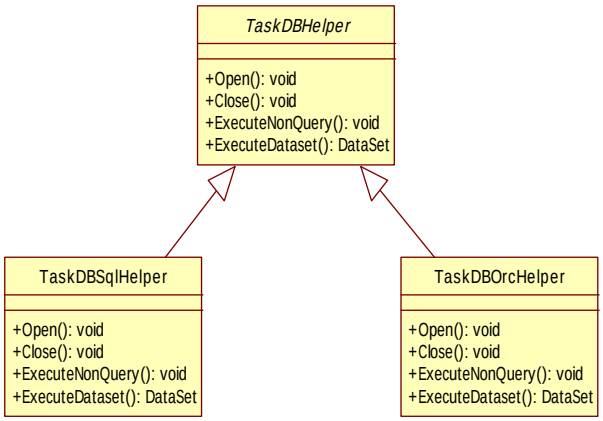

图 3　数据库访问类

3　模型类的应用

本节介绍模型类应用到改进研究室现有的任务管理系统的过程。任务管理系统的核心模型类是任务条目和任务包。其中，任务条目是最基本的任务，是任务管理系统的基本单元。任务包可以包括一个或多个任务条目，用于任务的分发和转发。以生成任务条目类为例，介绍模型类的使用方法。

首先在数据库中建好任务条目表，任务条目表结构如下：

表 1　任务管理系统核心表之一——任务条目表

列	数据类型	说明
VERSION	字符型	版本名
ITEMS	字符型	条目名
CONTENT	字符型	条目内容
BSF	整型	标准分
CPFF	字符型	查评方法
PFBZ	字符型	评价标准
NOTE	字符型	备注
CODE	字符型	编码
PARENT_ID	字符型	父结点
CPYJ	字符型	查评依据客户端显示文件名
CPYJ_FILENAME	字符型	查评依据服务器端文件名
MEMO	字符型	检查表模板客户端显示文件名
MEMO_FILENAME	字符型	检查表模板服务器端文件名
ORDERBY	整型	排序码，从小到大排序
HAS_CHILDREN	字符型	是否有子结点，Y 有 N 无

接下来，把任务条目表的表名作为参数，调用数据表工厂的 GetTable（string tableName）方法，就得到了任务条目表类，它是 TaskDataTable 的一个实例。该实例拥有增加、删除、修改、获得一条数据等方法。这些方法在通用数据表操作类中实现。

其他需要实现的特殊方法，可以继承通用数据表操作类或组合该类的方法进行实现。

用类似的方法可以生成任务管理系统的其它模型类。生成的模型类已经应用到研究室现有的任务管理系统中，对该系统进行了改进，提高了代码的重用性和系统的可维护性。

4　结语

本文对研究室已有任务管理系统的模型类进行了抽象和重新设计，把数据库操作封装、隔离，使业务逻辑层与底层数据结构分离，得到了通用的数据表类和数据表操作类。这在很大程度上提高了代码的可重用性，大大提高类似系统的开发效率。在系统中使用这种方法，可以大幅度减少代码量，提高系统的可维护性和扩展性，降低需求变更对系统的影响。任务管理系统模型类具有一定的通用性，在类似系统中也可以使用。本文介绍的方法已经应用到对现有任务管理系统的改进中，在很大程度上提高了开发效率和系统的可维护性。下一步的工作是对任务管理系统的其他类进行抽象和设计，构建一个类库，使类的可重用性提高到一个更高的层次。在类库的组织层次、接口、异常处理等方面还要进行深入的研究，加快实现任务管理系统的平台化。

参考文献

1 高芬.基于 STRUTS 框架的任务管理系统的设计与实现.北京:北京邮电大学,2009.
2 郝雯,艾玲梅,王映辉.三层结构软件框架扩展点实现方法.计算机应用,2009,29(9):2541−2545.
3 易可可,陈志刚.基于 MVC 模式的 Web OA 系统设计与研究.计算机工程与应用,2005,4:112−115.
4 邵维忠,杨芙清.面向对象的系统设计.北京:清华大学出版社,2007.
5 Freeman E.Head First 设计模式.北京:中国电力出版社,2007.
6 秦澎涛,王苏文.简单工厂模式在数据访问层中的应用.计算机工程与设计,2009,30(7):1799−1801.
7 陈一宝.基于简单工厂模式的一些研究.计算机工程应用技术,2009,5(18):4822−4823.