uml.org.cn

分享到

深入浅出UML类图

作者：刘伟 ，发布于：2012-11-23，来源：CSDN

在UML 2.0的13种图形中，类图是使用频率最高的UML图之一。Martin Fowler在其著作《UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition》（《UML精粹：标准对象建模语言简明指南（第3版）》）中有这么一段：“If someone were to come up to you in a dark alley and say, 'Psst, wanna see a UML diagram?' that diagram would probably be a class diagram. The majority of UML diagrams I see are class diagrams.”（“如果有人在黑暗的小巷中向你走来并对你说：‘嘿，想不想看一张UML图？’那么这张图很有可能就是一张类图，我所见过的大部分的UML图都是类图”），由此可见类图的重要性。

类图用于描述系统中所包含的类以及它们之间的相互关系，帮助人们简化对系统的理解，它是系统分析和设计阶段的重要产物，也是系统编码和测试的重要模型依据。

1. 类

类(Class)封装了数据和行为，是面向对象的重要组成部分，它是具有相同属性、操作、关系的对象集合的总称。在系统中，每个类都具有一定的职责，职责指的是类要完成什么样的功能，要承担什么样的义务。一个类可以有多种职责，设计得好的类一般只有一种职责。在定义类的时候，将类的职责分解成为类的属性和操作（即方法）。类的属性即类的数据职责，类的操作即类的行为职责。设计类是面向对象设计中最重要的组成部分，也是最复杂和最耗时的部分。

在软件系统运行时，类将被实例化成对象(Object)，对象对应于某个具体的事物，是类的实例(Instance)。

类图(Class Diagram)使用出现在系统中的不同类来描述系统的静态结构，它用来描述不同的类以及它们之间的关系。

在系统分析与设计阶段，类通常可以分为三种，分别是实体类(Entity Class)、控制类(Control Class)和边界类(Boundary Class)，下面对这三种类加以简要说明：

(1) 实体类：实体类对应系统需求中的每个实体，它们通常需要保存在永久存储体中，一般使用数据库表或文件来记录，实体类既包括存储和传递数据的类，还包括操作数据的类。实体类来源于需求说明中的名词，如学生、商品等。

(2) 控制类：控制类用于体现应用程序的执行逻辑，提供相应的业务操作，将控制类抽象出来可以降低界面和数据库之间的耦合度。控制类一般是由动宾结构的短语（动词+名词）转化来的名词，如增加商品对应有一个商品增加类，注册对应有一个用户注册类等

(3) 边界类：边界类用于对外部用户与系统之间的交互对象进行抽象，主要包括界面类，如对话框、窗口、菜单等。

在面向对象分析和设计的初级阶段，通常首先识别出实体类，绘制初始类图，此时的类图也可称为领域模型，包括实体类及其它们之间的相互关系。

2. 类的UML图示

在UML中，类使用包含类名、属性和操作且带有分隔线的长方形来表示，如定义一个Employee类，它包含属性name、age和email，以及操作modifyInfo()，在UML类图中该类如图1所示：

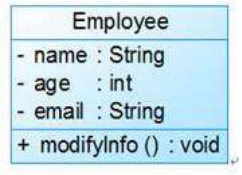


图1 类的UML图示

图1对应的Java代码片段如下：

```
public class Employee {
        private String name;
        private int age;
        private String email;

        public void modifyInfo() {
                ......
        }
}
```

在UML类图中，类一般由三部分组成：

(1) 第一部分是类名：每个类都必须有一个名字，类名是一个字符串。

(2) 第二部分是类的属性(Attributes)：属性是指类的性质，即类的成员变量。一个类可以有任意多个属性，也可以没有属性

相关文章

如何向妻子解释OOD
OOAD与UML笔记
UML类图与类的关系详解
UML统一建模语言初学
总结一下领域模型的验证
基于 UML 的业务建模

相关培训课程

面向对象的分析设计
基于UML的面向对象分析设计
UML + 嵌入式系统分析设计
关系数据库面向OOAD设计
业务建模与业务架构
使用用例进行需求管理
更多课程...

成功案例

某航空IT部门 业务分析与业务建模
联想 业务需求分析与建模
北京航管科技 EA工具与架构设计
使用EA和UML进行嵌入式系统分析
全球最大的茶业集团 UML系统分析
华为 基于EA的嵌入式系统建模
水资源服务商 基于EA进行UML建模
更多...

UML规定属性的表示方式为：

可见性 名称:类型 [= 缺省值]

其中：

- “可见性”表示该属性对于类外的元素而言是否可见，包括公有(public)、私有(private)和受保护(protected)三种，在类图中分别用符号+、-和#表示。
- “名称”表示属性名，用一个字符串表示。
- “类型”表示属性的数据类型，可以是基本数据类型，也可以是用户自定义类型。
- “缺省值”是一个可选项，即属性的初始值。

(3) 第三部分是类的操作(Operations)：操作是类的任意一个实例对象都可以使用的行为，是类的成员方法。

UML规定操作的表示方式为：

可见性 名称(参数列表) [: 返回类型]

其中：

- “可见性”的定义与属性的可见性定义相同。
- “名称”即方法名，用一个字符串表示。
- “参数列表”表示方法的参数，其语法与属性的定义相似，参数个数是任意的，多个参数之间用逗号“，”隔开。
- “返回类型”是一个可选项，表示方法的返回值类型，依赖于具体的编程语言，可以是基本数据类型，也可以是用户自定义类型，还可以是空类型(void)，如果是构造方法，则无返回类型。

在类图2中，操作method1的可见性为public(+)，带入了一个Object类型的参数par，返回值为空(void)；操作method2的可见性为protected(#)，无参数，返回值为String类型；操作method3的可见性为private(-)，包含两个参数，其中一个参数为int类型，另一个为int[]类型，返回值为int类型。

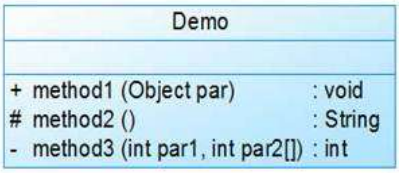


图2 类图操作说明示意图

由于在Java语言中允许出现内部类，因此可能会出现包含四个部分的类图，如图3所示：

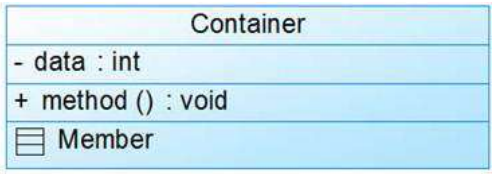


图3 包含内部类的类图

类与类之间的关系（1）

在软件系统中，类并不是孤立存在的，类与类之间存在各种关系，对于不同类型的关系，UML提供了不同的表示方式。

1. 关联关系

关联(Association)关系是类与类之间最常用的一种关系，它是一种结构化关系，用于表示一类对象与另一类对象之间有联系，如汽车和轮胎、师傅和徒弟、班级和学生等等。在UML类图中，用实线连接有关联关系的对象所对应的类，在使用Java、C#和C++等编程语言实现关联关系时，通常将一个类的对象作为另一个类的成员变量。在使用类图表示关联关系时可以在关联线上标注角色名，一般使用一个表示两者之间关系的动词或者名词表示角色名（有时该名词为实例对象名），关系的两端代表两种不同的角色，因此在一个关联关系中可以包含两个角色名，角色名不是必须的，可以根据需要增加，其目的是使类之间的关系更加明确。

如在一个登录界面类LoginForm中包含一个JButton类型的注册按钮loginButton，它们之间可以表示为关联关系，代码实现时可以在LoginForm中定义一个名为loginButton的属性对象，其类型为JButton。如图1所示：

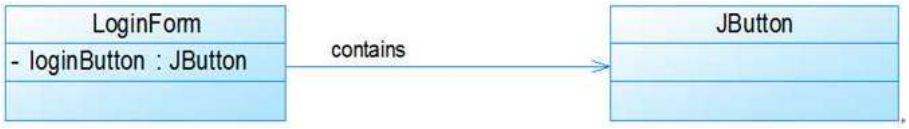


图1 关联关系实例

图1对应的Java代码片段如下：

```
public class LoginForm {
private JButton loginButton; //定义为成员变量
……
}

public class JButton {
    ……
}
```

在UML中，关联关系通常又包含如下几种形式：

（1） 双向关联

默认情况下，关联是双向的。例如：顾客(Customer)购买商品(Product)并拥有商品，反之，卖出的商品总有某个顾客与之相关联。因此，Customer类和Product类之间具有双向关联关系，如图2所示：

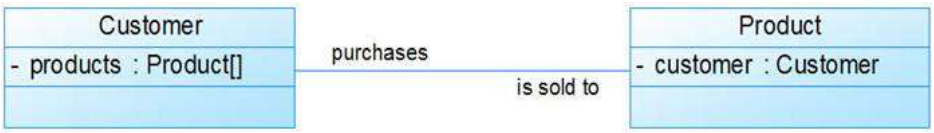


图2 双向关联实例

图2对应的Java代码片段如下：

```
public class Customer {
private Product[] products;
……
}

public class Product {
private Customer customer;
……
}
```

（2） 单向关联

类的关联关系也可以是单向的，单向关联用带箭头的实线表示。例如：顾客(Customer)拥有地址(Address)，则Customer类与Address类具有单向关联关系，如图3所示：

图3 单向关联实例

图3对应的Java代码片段如下：

```
public class Customer {
private Address address;
……
}

public class Address {
……
}
```

（3） 自关联

在系统中可能会存在一些类的属性对象类型为该类本身，这种特殊的关联关系称为自关联。例如：一个节点类(Node)的成员又是节点Node类型的对象，如图4所示：

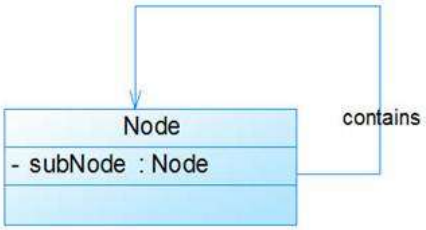


图4 自关联实例

图4对应的Java代码片段如下：

```
public class Node {
private Node subNode;
……
}
```

（4） 多重性关联

多重性关联关系又称为重数性(Multiplicity)关联关系，表示两个关联对象在数量上的对应关系。在UML中，对象之间的多重性可以直接在关联直线上用一个数字或一个数字范围表示。

对象之间可以存在多种多重性关联关系，常见的多重性表示方式如表1所示：

表1 多重性表示方式列表

表示方式	多重性说明
1..1	表示另一个类的一个对象只与该类的一个对象有关系

0..*　　表示另一个类的一个对象与该类的零个或多个对象有关系

1..*　　表示另一个类的一个对象与该类的一个或多个对象有关系

0..1　　表示另一个类的一个对象没有或只与该类的一个对象有关系

m..n　　表示另一个类的一个对象与该类最少m，最多n个对象有关系（m≤n）

例如：一个界面(Form)可以拥有零个或多个按钮(Button)，但是一个按钮只能属于一个界面，因此，一个Form类的对象可以与零个或多个Button类的对象相关联，但一个Button类的对象只能与一个Form类的对象关联，如图5所示：

图5 多重性关联实例

图5对应的Java代码片段如下：

```
public class Form {
private Button[] buttons; //定义一个集合对象
……
}

public class Button {
……
}
```

（5） 聚合关系

聚合(Aggregation)关系表示整体与部分的关系。在聚合关系中，成员对象是整体对象的一部分，但是成员对象可以脱离整体对象独立存在。在UML中，聚合关系用带空心菱形的直线表示。例如：汽车发动机(Engine)是汽车(Car)的组成部分，但是汽车发动机可以独立存在，因此，汽车和发动机是聚合关系，如图6所示：

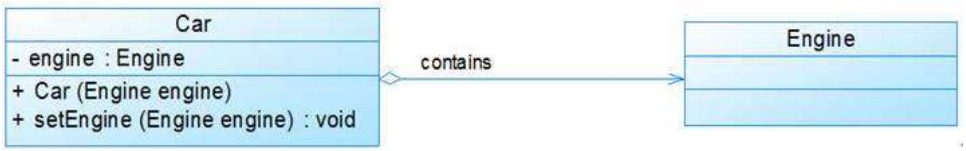


图6 聚合关系实例

在代码实现聚合关系时，成员对象通常作为构造方法、Setter方法或业务方法的参数注入到整体对象中，图6对应的Java代码片段如下：

```
public class Car {
        private Engine engine;

    //构造注入
        public Car(Engine engine) {
                this.engine = engine;
        }

    //设值注入
public void setEngine(Engine engine) {
    this.engine = engine;
}
……
}

public class Engine {
        ……
}
```

（6） 组合关系

组合(Composition)关系也表示类之间整体和部分的关系，但是在组合关系中整体对象可以控制成员对象的生命周期，一旦整体对象不存在，成员对象也将不存在，成员对象与整体对象之间具有同生共死的关系。在UML中，组合关系用带实心菱形的直线表示。例如：人的头(Head)与嘴巴(Mouth)，嘴巴是头的组成部分之一，而且如果头没了，嘴巴也就没了，因此头和嘴巴是组合关系，如图7所示：

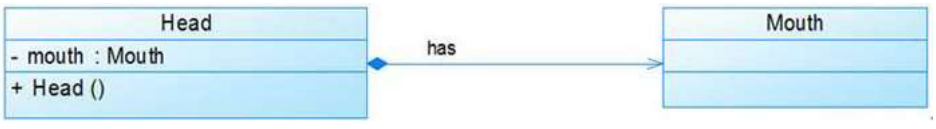


图7 组合关系实例

在代码实现组合关系时，通常在整体类的构造方法中直接实例化成员类，图7对应的Java代码片段如下：

```
public class Head {
        private Mouth mouth;
```

```
        public Head() {
                mouth = new Mouth(); //实例化成员类
        }
    ……
    }

    public class Mouth {
        ……
    }
```

类与类之间的关系（2）

2. 依赖关系

依赖(Dependency)关系是一种使用关系，特定事物的改变有可能会影响到使用该事物的其他事物，在需要表示一个事物使用另一个事物时使用依赖关系。大多数情况下，依赖关系体现在某个类的方法使用另一个类的对象作为参数。在UML中，依赖关系用带箭头的虚线表示，由依赖的一方指向被依赖的一方。例如：驾驶员开车，在Driver类的drive()方法中将Car类型的对象car作为一个参数传递，以便在drive()方法中能够调用car的move()方法，且驾驶员的drive()方法依赖车的move()方法，因此类Driver依赖类Car，如图1所示：

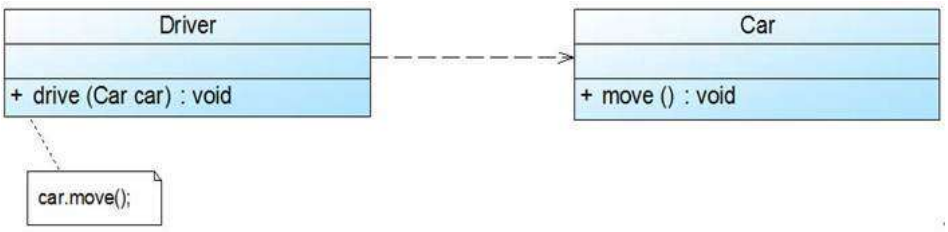


图1 依赖关系实例

在系统实施阶段，依赖关系通常通过三种方式来实现，第一种也是最常用的一种方式是如图1所示的将一个类的对象作为另一个类中方法的参数，第二种方式是在一个类的方法中将另一个类的对象作为其局部变量，第三种方式是在一个类的方法中调用另一个类的静态方法。图1对应的Java代码片段如下：

```
    public class Driver {
        public void drive(Car car) {
                car.move();
        }
      ……
    }

    public class Car {
        public void move() {
                ......
        }
      ……
    }
```

3. 泛化关系

泛化(Generalization)关系也就是继承关系，用于描述父类与子类之间的关系，父类又称作基类或超类，子类又称作派生类。在UML中，泛化关系用带空心三角形的直线来表示。在代码实现时，我们使用面向对象的继承机制来实现泛化关系，如在Java语言中使用extends关键字、在C++/C#中使用冒号“：”来实现。例如：Student类和Teacher类都是Person类的子类，Student类和Teacher类继承了Person类的属性和方法，Person类的属性包含姓名(name)和年龄(age)，每一个Student和Teacher也都具有这两个属性，另外Student类增加了属性学号(studentNo)，Teacher类增加了属性教师编号(teacherNo)，Person类的方法包括行走move()和说话say()，Student类和Teacher类继承了这两个方法，而且Student类还新增方法study()，Teacher类还新增方法teach()。如图2所示：

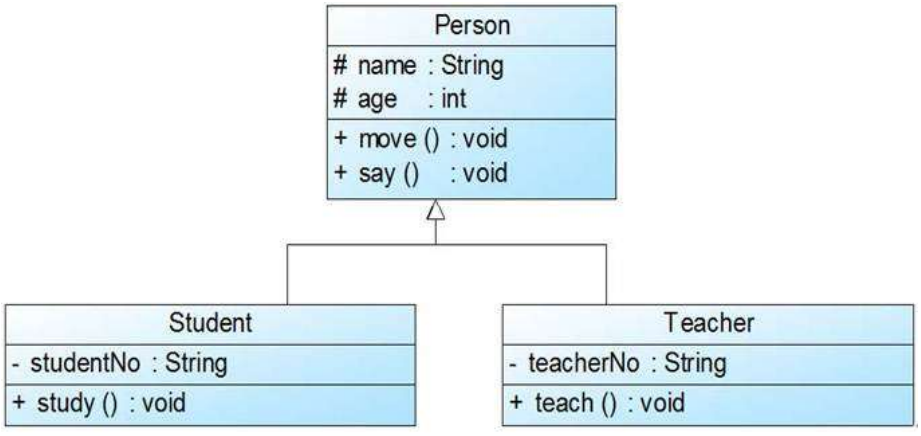


图2 泛化关系实例

图2对应的Java代码片段如下：

```
//父类
public class Person {
protected String name;
protected int age;

public void move() {
        ……
}

    public void say() {
    ……
    }
}

//子类
public class Student extends Person {
private String studentNo;

public void study() {
    ……
    }
}

//子类
public class Teacher extends Person {
private String teacherNo;

public void teach() {
    ……
    }
}
```

4. 接口与实现关系

在很多面向对象语言中都引入了接口的概念，如Java、C#等，在接口中，通常没有属性，而且所有的操作都是抽象的，只有操作的声明，没有操作的实现。UML中用与类的表示法类似的方式表示接口，如图3所示：

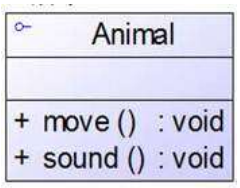


图3 接口的UML图示

接口之间也可以有与类之间关系类似的继承关系和依赖关系，但是接口和类之间还存在一种实现(Realization)关系，在这种关系中，类实现了接口，类中的操作实现了接口中所声明的操作。在UML中，类与接口之间的实现关系用带空心三角形的虚线来表示。例如：定义了一个交通工具接口Vehicle，包含一个抽象操作move()，在类Ship和类Car中都实现了该move()操作，不过具体的实现细节将会不一样，如图4所示：

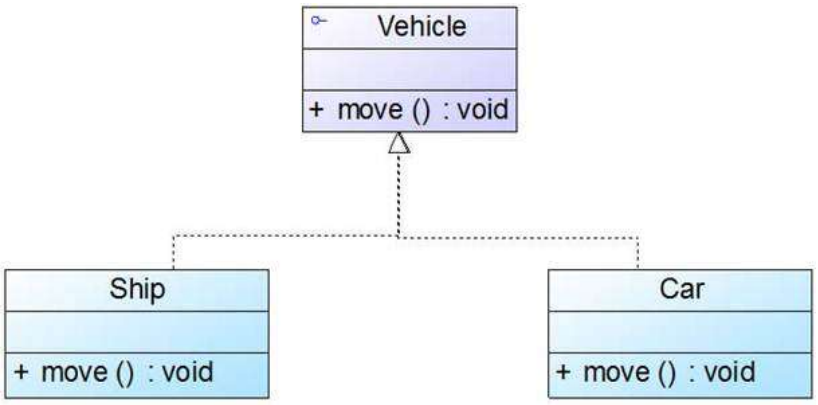


图4 实现关系实例

实现关系在编程实现时，不同的面向对象语言也提供了不同的语法，如在Java语言中使用implements关键字，而在C++/C#中使用冒号“：”来实现。图4对应的Java代码片段如下：

```
public interface Vehicle {
public void move();
}

public class Ship implements Vehicle {
public void move() {
    ……
    }
```

```
}

public class Car implements Vehicle {
public void move() {
    ……
    }
}
```

实例分析1——登录模块

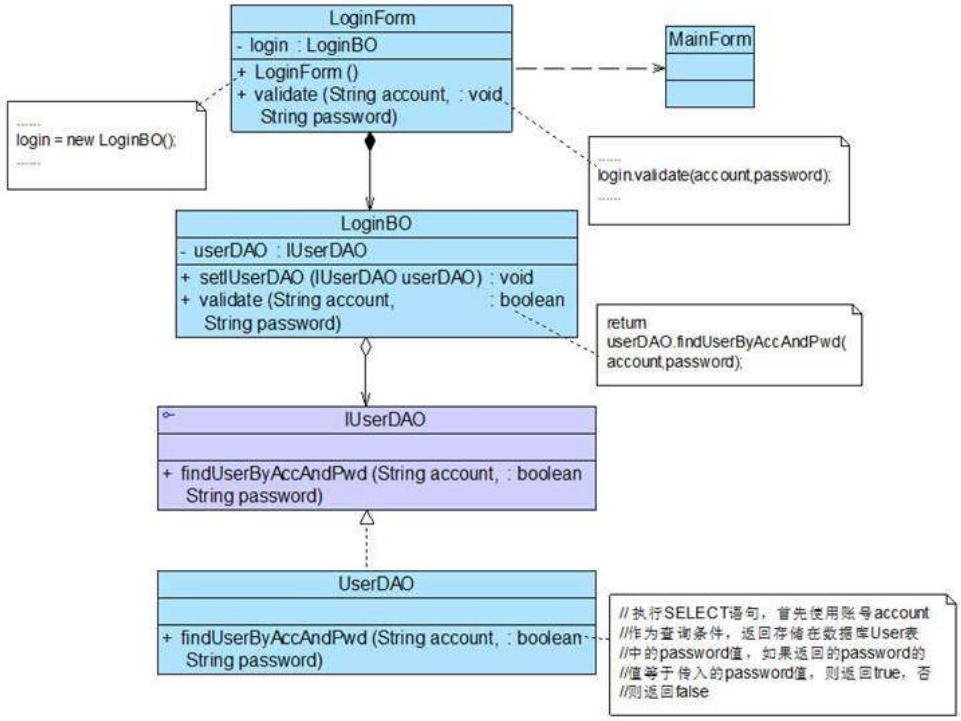
某基于C/S的即时聊天系统登录模块功能描述如下：

用户通过登录界面(LoginForm)输入账号和密码，系统将输入的账号和密码与存储在数据库(User)表中的用户信息进行比较，验证用户输入是否正确，如果输入正确则进入主界面(MainForm)，否则提示“输入错误”。

根据以上描述绘制初始类图。

参考解决方案：

参考类图如下：



考虑到系统扩展性，在本实例中引入了抽象数据访问接口IUserDAO，再将具体数据访问对象注入到业务逻辑对象中，可通过配置文件（如XML文件）等方式来实现，将具体的数据访问类类名存储在配置文件中，如果需要更换新的具体数据访问对象，只需修改配置文件即可，原有程序代码无须做任何修改。

类说明：

类 名	说 明
LoginForm	登录窗口，省略界面组件和按钮事件处理方法（边界类）
LoginBO	登录业务逻辑类，封装实现登录功能的业务逻辑（控制类）
IUserDAO	抽象数据访问类接口，声明对User表的数据操作方法，省略除查询外的其他方法（实体类）
UserDAO	具体数据访问类，实现对User表的数据操作方法，省略除查询外的其他方法（实体类）
MainForm	主窗口（边界类）

方法说明：

方法名	说 明
LoginForm类的LoginForm()方法	LoginForm构造函数，初始化实例成员
LoginForm类的validate()方法	界面类的验证方法，通过调用业务逻辑类LoginBO的validate()方法实现对用户输入信息的验证
LoginBO类的validate()方法	业务逻辑类的验证方法，通过调用数据访问类的findUserByAccAndPwd()方法验证用户输入信息的合法性
LoginBO类的setIUserDAO()方法	Setter方法，在业务逻辑对象中注入数据访问对象（注意：此处针对抽象数据访问类编程
IUserDAO接口的findUserByAccAndPwd()方法	业务方法声明，通过用户账号和密码在数据库中查询用户信息，判断该用户身份的合法性
UserDAO类的findUserByAccAndPwd()方法	业务方法实现，实现在IUserDAO接口中声明的数据访问方法

实例分析2——注册模块

某基于Java语言的C/S软件需要提供注册功能，该功能简要描述如下：

用户通过注册界面(RegisterForm)输入个人信息，用户点击“注册”按钮后将输入的信息通过一个封装用户输入数据的对象(UserDTO)传递给操作数据库的数据访问类，为了提高系统的扩展性，针对不同的数据库可能需要提供不同的数据访问类，因此提供了数据访问类接口，如IUserDAO，每一个具体数据访问类都是某一个数据访问类接口的实现类，如OracleUserDAO就是一个专门用于访问Oracle数据库的数据访问类。

根据以上描述绘制类图。为了简化类图，个人信息仅包括账号(userAccount)和密码(userPassword)，且界面类无需涉及界面细节元素。

参考解决方案：

在以上功能说明中，可以分析出该系统包括三个类和一个接口，这三个类分别是注册界面类RegisterForm、用户数据传输类UserDTO、Oracle用户数据访问类OracleUserDAO，接口是抽象用户数据访问接口IUserDAO。它们之间的关系如下：

(1) 在RegisterForm中需要使用UserDTO类传输数据且需要使用数据访问类来操作数据库，因此RegisterForm与UserDTO和IUserDAO之间存在关联关系，在RegisterForm中可以直接实例化UserDTO，因此它们之间可以使用组合关联。

(2) 由于数据库类型需要灵活更换，因此在RegisterForm中不能直接实例化IUserDAO的子类，可以针对接口IUserDAO编程，再通过注入的方式传入一个IUserDAO接口的子类对象（在本书后续章节中将学习如何具体实现），因此RegisterForm和IUserDAO之间具有聚合关联关系。

(3) OracleUserDAO是实现了IUserDAO接口的子类，因此它们之间具有类与接口的实现关系。

(4) 在声明IUserDAO接口的增加用户信息方法addUser()时，需要将在界面类中实例化的UserDTO对象作为参数传递进来，然后取出封装在UserDTO对象中的数据插入数据库，因此addUser()方法的函数原型可以定义为：public boolean addUser(UserDTO user)，在IUserDAO的方法addUser()中将UserDTO类型的对象作为参数，故IUserDAO与UserDTO存在依赖关系。

通过以上分析，该实例参考类图如图1所示：

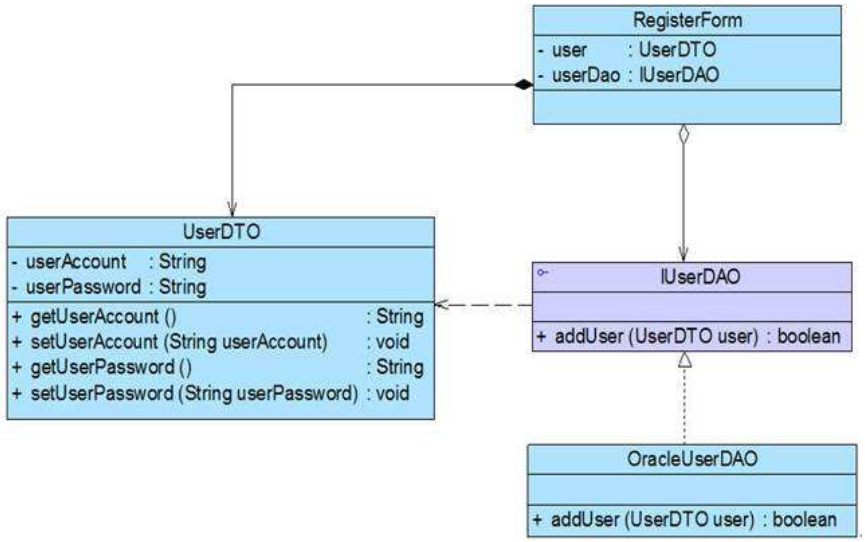


图1 注册功能参考类图

注意：在绘制类图或其他UML图形时，可以通过注释(Comment)来对图中的符号或元素进行一些附加说明，如果需要详细说明类图中的某一方法的功能或者实现过程，可以使用如图2所示表示方式：

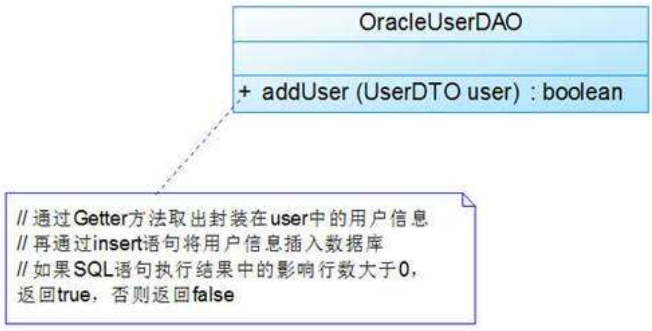
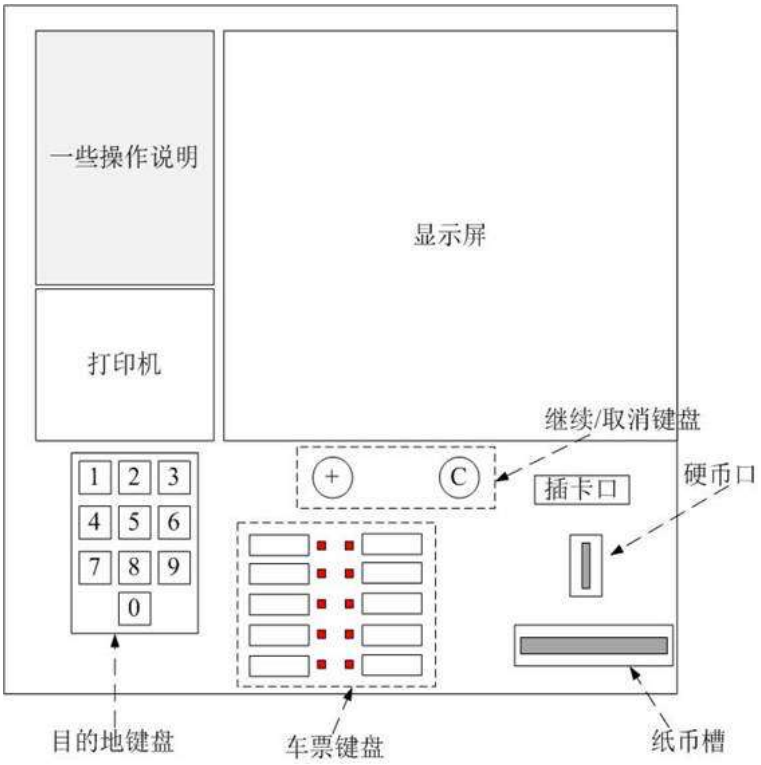


图2 类图注释实例

实例分析3——售票机控制程序

某运输公司决定为新的售票机开发车票销售的控制软件。图1给出了售票机的面板示意图以及相关的控制部件。



图I 售票机面板示意图

售票机相关部件的作用如下所述：

(1) 目的地键盘用来输入行程目的地的代码（例如，200表示总站）。

(2) 乘客可以通过车票键盘选择车票种类（单程票、多次往返票和座席种类）。

(3) 继续/取消键盘上的取消按钮用于取消购票过程，继续按钮允许乘客连续购买多张票。

(4) 显示屏显示所有的系统输出和用户提示信息。

(5) 插卡口接受MCard（现金卡），硬币口和纸币槽接受现金。

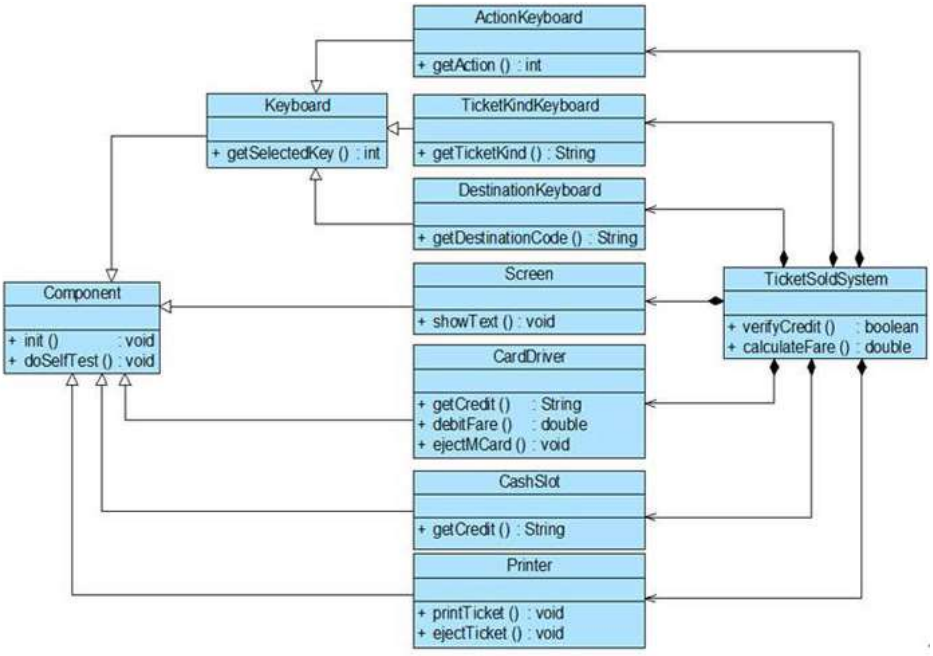
(6) 打印机用于输出车票。

(7) 所有部件均可实现自检并恢复到初始状态。

现采用面向对象方法开发该系统，使用UML进行建模，绘制该系统的初始类图。

参考解决方案：

参考类图如下：



类说明：

类 名	说 明
Component	抽象部件类，所有部件类的父类

Keyboard	抽象键盘类
ActionKeyboard	继续/取消键盘类
TicketKindKeyboard	车票种类键盘类
DestinationKeyboard	目的地键盘类
Screen	显示屏类
CardDriver	卡驱动器类
CashSlot	现金（硬币/纸币）槽类
Printer	打印机类
TicketSoldSystem	售票系统类

方法说明：

方法名	说 明
Component 的init()方法	初始化部件
Component 的doSeltTest()方法	自检
Keyboard的getSelectedKey()方法	获取按键值
ActionKeyboard的getAction()方法	继续/取消键盘事件处理
TicketKindKeyboard的getTicketKind()方法	车票种类键盘事件处理
DestinationKeyboard的getDestinationCode()方法	目的地键盘事件处理
Screen的showText()方法	显示信息
CardDriver的getCredit()方法	获取金额
CardDriver的debitFare()方法	更新卡余额
CardDriver的ejectMCard()方法	退卡
CashSlot的getCredit()方法	获取金额
Printer的printTicket()方法	打印车票
Printer的ejectTicket()方法	出票
TicketSoldSystem的verifyCredit()方法	验证金额
TicketSoldSystem的calculateFare()方法	计算费用

分享到

讲座：基于模型的系统工程 MBSE

2月26日 在线直播