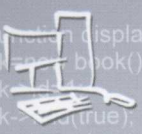

对MVC设计模式实现算法与核心技术的深度积累是本书的核心

MVC 程序设计

罗维 张华 著

中国水利水电出版社
www.waterpub.com.cn

MVC 程序设计

罗维　张华　著

中国水利水电出版社
www.waterpub.com.cn

内 容 提 要

本书是作者多年研究成果与教学经验的总结，内容涉及面向对象程序设计理论、MVC 设计模式、MVC 开发框架，单一入口算法、路由算法、面向对象数据库中间件核心技术、模板引擎核心技术、缓存与静态化等关键技术内容，还对 Web 开发中常用的三种组件实现技术进行了分析。

本书核心在于对 MVC 设计模式实现算法与核心技术上的深度积累，对从原理分析，到算法设计，再到算法实现的全过程进行了深入浅出的撰写，思路清晰，层次分明，重点突出，并对关键算法与核心代码段给出了详尽描述。

本书面向有一定编程基础的读者，适合于从事基础架构研究或设计模式研究的相关人员参考，可作为高等学校计算机专业教材，也可作为中高级程序员培训教材。

图书在版编目（CIP）数据

MVC程序设计 / 罗维，张华著. -- 北京 : 中国水利水电出版社，2013.5
ISBN 978-7-5170-0878-1

Ⅰ. ①M… Ⅱ. ①罗… ②张… Ⅲ. ①程序设计 Ⅳ. ①TP311.1

中国版本图书馆CIP数据核字(2013)第107442号

策划编辑：寇文杰　责任编辑：张玉玲　加工编辑：程　蕊　封面设计：李　佳

书　　名	MVC 程序设计
作　　者	罗维　张华　著
出版发行	中国水利水电出版社 （北京市海淀区玉渊潭南路 1 号 D 座　100038） 网址：www.waterpub.com.cn E-mail：mchannel@263.net（万水） sales@waterpub.com.cn 电话：（010）68367658（发行部）、82562819（万水）
经　　售	北京科水图书销售中心（零售） 电话：（010）88383994、63202643、68545874 全国各地新华书店和相关出版物销售网点
排　　版	北京万水电子信息有限公司
印　　刷	北京蓝空印刷厂
规　　格	170mm×240mm　16 开本　10.5 印张　208 千字
版　　次	2013 年 5 月第 1 版　2013 年 5 月第 1 次印刷
印　　数	0001—3000 册
定　　价	36.00 元

前　言

企业级应用系统的开发注重的是系统的稳定性、可靠性、可维护性与可重用性，MVC（Model-View-Controller）设计模式则是实现上述要求的架构保证。现在基于 MVC 模式的 Web 应用程序已成为企业应用系统的主流，而 MVC 模式要依赖于其宿主语言提供的 MVC 应用程序开发框架才能实现，Java 有 Struts、Velocity 等著名的框架，微软的 ASP.NET 也已内置了 MVC 开发框架。PHP 作为一种专注于服务器端开发的语言，已在全球范围内广泛使用，在网站、博客、论坛及团购等非事务性系统开发上已成为中流砥柱。随着 PHP 面向对象编程能力的成熟，PHP 在企业级应用领域也逐渐普及，为满足企业级开发的需要，PHP 如何实现 MVC 模式就成为 PHP 编程领域内重要的基础性研究课题，研究内容涉及 MVC 实际模型、路由、控制器、DAO 组件及模板引擎等方面。此项研究对于提高 PHP 在企业级应用软件开发中的规范性、可靠性与可扩展性都有大的促进作用，并且基于框架的规范性开发能显著降低企业信息化的总体拥有成本。

MVC 框架研究是一个复杂的系统工程，属于长周期的底层研究课题。此项研究不但需要对语言本身非常熟悉，还要对设计模式、面向对象程序设计、重写机制、模板编译、静态化与缓存、数据持久化等算法方面有着深入的理解与掌握。作为本书研究成果与技术支撑的 MVC 框架耗费近 6 年时间研发与完善。在框架已稳定并逐渐成熟之际，为了让更多的软件技术人员了解 MVC，学习 MVC 开发，笔者特编著本书，希望通过此书，使应用级开发人员能快速掌握 MVC 开发模式；使高级程序员有一个二次开发的基础框架；使架构师有一个较为完整的研究与参照对象。

为了方便读者更好地学习与研究 PHP-MVC 高级开发，本书从面向对象程序设计入手，结合笔者多年的开发经验及研究成果，对面向过程与面向对象编程进行了对比分析，使读者能真正转型到面向对象思维方式，然后去学习面向对象程序设计的高级特征，最后在此基础上理解 MVC 编程思想，掌握 MVC 开发模式，了解 MVC 框架的核心实现技术，从整体上提高软件开发的水平。

编　者

2013 年 4 月

目　　录

第 1 章　PHP 面向对象程序设计

面向对象程序设计是现代程序设计语言的重要特征，是程序员应该掌握的编程思维能力。PHP 从面向过程发展成为优秀的面向对象程序设计语言也是适应技术发展趋势与满足实际需要的结果。本章将从面向对象程序设计的概念、PHP 面向对象程序设计方法等方面分析 PHP 面向对象程序设计，注重引导面向对象程序设计思维方式，使读者能更好地从面向过程编程过渡到面向对象编程。

1.1　面向对象程序设计思维概述

1.1.1　面向对象程序设计概念

任何新事物的出现都是与人类认知的进步和需求的产生分不开的。在面向对象引入之前，人们用面向过程的方式开发程序，所关注的是流程，所封装的是功能片段。随着社会的发展，软件的应用领域越来越广，所面对的系统也越来越复杂，人们逐渐认识到，传统的以流程为中心、以功能为基础的设计方式已不能满足日益复杂的实际需要。在这个过程中，人们对程序设计的方式进行了深度的思考与不断的改进，逐步认识到从事物的本性出发去反映事物，去模拟实现事物所拥有的功能才能更好地适应现代程序设计的需要，面向对象程序设计就这样诞生并在越来越多的编程语言中实现。PHP 在 5.0 版本后已基本实现了面向对象的主要特征，正式成为面向对象设计语言家族中的一员。表 1.1 展示了两种设计方式在思维上的差异。

1.1.2　面向对象程序设计语言的特征

面向对象程序设计不仅是思维方式上的改变，具体到程序设计上，也带来重大的变化。类、对象、封装、继承、多态等都是面向对象程序设计的基本要点。PHP 5.0 以后的版本支持绝大部分重要的面向对象特征。

（1）类：同类事物共性的抽象。比如人类、鸟类等都是对具有共有属性的事物的高度抽象后得到的定义。在面向对象程序设计模式中，类也是对具有共同属性的对象的统一描述，通过类来定义对象的属性与方法。

（2）对象：类的实例化，是真正可完成具体动作的事物，与类有密切的关系。在面向对象程序设计模式中，类必须实例化为一个对象后，通过这个对象才执行类所拥有的功能。

表 1.1

思维方式	
面向过程	面向对象
对于传统的面向过程程序设计来说，我们的思维方式是从业务流程与功能模块的设计开始的。这种方式从本质上来说是正确的，也符合人的传统思维习惯。比如学生在学校学习期间，其大致的业务流程为报到注册、安排住宿、选课、学习、考试与实习。对应的是首先考虑学生管理系统有哪些顶层功能与哪些子功能，比如学生管理信息系统包括学生学籍管理、生活与纪律管理、选课管理、成绩管理、实习管理。每一个模块又包含若干个子功能，如选课又包括教学机构报教学计划、资源评估(即老师与教室等是否足够)、教学计划挂网、学生选课、选课确认等；实习管理又包括实习单位与学生的双向选择、实习过程管理、实习评分等功能。这样，在传统的开发过程中，我们就是按这种方式去完成信息系统的开发的。在这种方式下，设计师每天想的就是如何给系统增加新的功能，并且评估新功能与已有功能之间的数据关联	当进化到面向对象程序设计的时候，人们的思维方式有了改变。人们不再去以流程为主导去思考问题，而是转为以事物为主导去思考。也就是说，当要实现学生管理信息系统的时候，首先我们想到的是：学生是一个人，而一个人应该具有姓名、出生日期、性别、籍贯等所谓的属性；有吃饭、说话、运动、谈恋爱等动作。而学生属于人类，则应该拥有人类所有的特征。除此之外，学生有自己特性的地方，因此学生还应该在人类基本属性的基础之上添加班级、寝室号、考试成绩、奖惩情况等属性，拥有选课、做好人好事、打人等动作。对于大学生，则属于学生类更精细的子类，就应该在学生类的基础之上包含专业、导师、实习单位等属性，拥有选择导师、选实习单位等动作。这样，我们对于学生管理信息系统的设计，不再以流程为中心，而是以对象为中心，围绕对象所自然拥有的属性、动作来实现系统，并且根据不同的场景进行合理的继承，在此基础上扩充更多的属性与动作，使之能更好地满足信息管理的需要。这就是面向对象程序设计思维方式。在这种方式下，设计师每天想的就是如何去发现对象拥有而原来没有包含进去的属性与动作

(3) 封装：封装是面向对象直观特征之一。将事物共有的属性与拥有的操作组织在一起，根据属性与操作各自的特性提供不同的对内、对外访问权限，达到较佳的与内外环境进行信息交换与数据处理的目的。

(4) 继承：面向对象最吸引人的能力就是继承。通过继承我们可以很好地实现“站在巨人肩上”这一名言。在封装基础上实现的继承是面向对象可以胜任复杂任务最有力的武器。在父类已经具备一定功能的基础上，子类继承父类的能力，并根据当前的实际需要进行增加或扩充新的功能、改进已有的操作、去掉不适用的方法等。继承使得面向对象编程活起来了。

PHP 引入面向对象编程模式后，一直在努力实现上述面向对象程序设计语言的特征，并且在 5.0 版本以后达到相当的水平，现在我们已可以使用 PHP 进行复杂的面向对象程序设计。当然，在面向对象程序设计语言的行列中，还有很多比 PHP 做得好的语言，如 Java、C#等，但 PHP 的魅力就在于其多样性，既能用传

统的面向过程方式来开发应用系统，也能用面向对象的方式来完成项目，只需要在项目管理上多下功夫，就能充分发挥 PHP 的潜力，更好地为程序设计服务。

1.2　PHP 面向对象程序设计基础

1.2.1　类的定义与实例化

在 PHP 中，定义类的命令格式如下：

```
class 类名{
    访问控制符 成员变量名;
    访问控制符 function 成员函数名{
    }
}
```

访问控制符（可见性修饰）在 PHP 中有三种：public、protected、private，分别代表了公有访问、保护访问与私有访问，具体的含义及应用将在第三节讲述。

对于人类，我们是最熟悉的了，人类一般具有姓名、性别与出生日期等属性，拥有取名与获取名称的方法，还有哭与吃的动作，就此我们使用 PHP 来定义一个人的类：

```
class Person{
    private $name;
    private $gender;
    private $birthday;
    public function setName($name){
        $this->name=$name;
        return true;
    }
    public function getName(){
        return $this->name;
    }
    public function fight(){
        echo 'person fight';
    }
    public function eat(){
        echo 'person eat';
    }
}
```

在上述代码片段中，Person 是类名，对于 Person 类，我们定义了三个私有成员变量、两个可操作成员变量的公有成员方法、两个公有成员动作。当然在 PHP 中并不区分方法与动作，它们都是一个含义：功能的封装。通过“人类”的定义，

我们学习了 PHP 中类的定义，了解了面向对象的封装形式，下面来研究如何使用这个类。

要使用类所定义的这些方法，必须首先要实例化类。类的实例化方法如下：

```
$person_one=new Person();
```

通过 new 关键字我们得到了人类的一个实例，也就是一个对象，这个对象才能真正执行定义在类中的各种操作。

```
//给这个人取个名
$person_one->setName('php 成都');
//获取人的名字
$person_one->getName();
//战斗
$person_one->fight();   //输出 person fight
```

当实例化一个对象后，就可以通过这个对象去使用定义在类中的各种公用方法。当然类的成员方法也能调用该类中其他成员方法，实现方法如下：

```
    public function tired(){
        $this->eat();
        echo 'have a rest';
    }
```

定义一个“累了”的方法，当人累的时候，就必须吃点东西再休息，因此在类的方法中可以让人先吃东西再睡觉，通过$this->eat()去调用本类中其他的成员方法，然后执行自身的相关操作。

$this 在 PHP 面向对象程序设计中是一个非常重要的关键字，在类的方法中操作该类的属性或调用该类的其他成员方法时都要用到$this，比如$this->name 或 $this->getName()。学习 PHP 面向对象编程要深刻理解$this 的含义与正确的使用方法。

1.2.2 类的继承

面向对象程序设计中，继承机制是最引人入胜的特性，通过继承机制，新的系统可在原有系统基础之上快速地实现与创新，通过不断的演变与修正，最终将得到一个较为完善、功能齐全的系统。

PHP 实现了类的继承机制，对于被继承的类，我们称为父类；对于继承类，我们称为子类，格式如下：

```
class 子类 extends 父类
```

下面以学生类为例来说明：

```
class Student extends Person {
    private $major;
    private $class_name;
```

```
    private $teacher;
    private $computer_score;
    private $chinese_score;
    public function setMajor($major){
        $this->major=$major;
        return true;
    }
    public function getMajor(){
        return $this->major;
    }
}
```

学生类中，定义了专业、班级、教师及两门成绩等属性，实现了设置专业与获取专业名称两个方法。虽然 Student 类没有定义对学生姓名等的处理代码，但我们知道，学生类是继承于人类的，因此将拥有人类全部的公有与保护类方法（详见 6.3 节），所以可以按下述代码实例化学生类，并对学生进行相关操作。

```
$student_one=new Student();
$student_one->setName('php 学员');
$student_one->setMajor('LAMP');
echo $student_one->getName().':'.$student_one->getMajor();  //输出 php 学员: LAMP
```

类在继承过程中，如果子类的行为方法与父类差异较大，这种情况下，可采用重写（overwrite）来实现。需要注意的是，PHP 暂时不支持重载（overload）。下面分别举例说明。

比如学生是禁止打架的，我们这样来继承：

```
public function fight(){
    echo " fight prohibit ";
}
$student_one->fight();  //输出 fight prohibit
```

在子类里建立与父类同名的成员方法实现了重写，在调用的时候就不会再去执行父类的方法，而是执行子类的方法，适用于子类与父类行为差异较大的情况。但如果子类的行为是在父类的基础上有所增强，还是需要父类的功能，则可采用重载来实现，如下：

```
public function fight(){
    parent::fight();
    echo "fight for freedom";
}
$student_one->fight();  //输出 person fight fight for freedom
```

通过 parent 引用当前类的父类，使用“::”操作符可以在子类的方法里执行父类的方法，这样就实现了既保留父类的功能又增加满足子类新功能的目的。

PHP 不支持多继承，只能从一个父类派生子类，请在使用时注意。关于继承的更多的内容，将在下一章中介绍。

1.2.3 成员属性与成员方法的可见性

在第一节中引入了 public、protected、private 三个关键词，它们是面向对象程序设计语言中的访问控制命令，通过这三个关键词就能控制哪些资源是对外的，哪些资源是对内的，能更好地实现封装。

（1）public：公有成员属性。由 public 修饰的成员属性与成员方法可以被类的实例、类的方法、子类的实例与子类的方法所访问，是限制最少的一类资源。

（2）protected：保护类成员属性。可在类的方法程序中访问，可在子类中访问，但不能由类的实例访问。

（3）private：私有成员属性。只能由当前类的方法访问，其子类与类的实例都不能访问。

```
class Student{
    public $name;
    protected $age;
    private $address;
    public function setName($name){
        if($this->checkName($name)){
            $this->name=$name;
            return true;
        }else{
            return false;
        }
    }
    public function setAge($age){
        if($this->checkAge($age)){
            $this->age=$age;
            return true;
        }else{
            return false;
        }
    }
    public function setAddress($address){
        if($this->checkAddress($address)){
            $this->address=$address;
            return true;
        }else{
            return false;
        }
```

```
    }
    public function getName(){
        return $this->name;
    }
    public function getAge(){
        return $this->name;
    }
    public function getAddress(){
        return $this->name;
    }
    public function checkName($name){
        if($name==''){
            return false;
        }else {
            return true;
        }
    }
    protected function checkAge($age){
        if($age==0){
            return false;
        }else {
            return true;
        }
    }
    private function checkAddress($address){
        if($address==''){
            return false;
        }else {
            return true;
        }
    }

}
$student_one=new Student();
$student_one->name='php 成都';  //直接访问公有属性变量
echo $student_one->getName();  //输出"php 成都"
//$student_one->age=18;     //出错，错误信息 Fatal error: Cannot access protected property
                            //student::$age
//$student_one->address='成都';   //出错，错误信息 Fatal error: Cannot access private property
                                  //student::$address
$student_one->setName('phpchengdu');
echo $student_one->getName();  //输出 phpchengdu
$student_one->checkName($student_one->getName());  //正确执行
```

```
//$student_one->checkAge($student_one->getAge());     //出错，错误信息 Fatal error: Call to
                                                      //protected method student::
                                                      //checkAge() from context
//$student_one->checkAddress($student_one->getAddress()); //出错，错误信息 Fatal error:
                                                          //Call to protected method
                                                          //student::checkAddress() from
                                                          //context
```

本例中，因为 name 属性是 public 的，所以可以通过对象$student_one->name 直接访问。方法都是 public 的，所以都能直接调用。而 private 与 protected 修饰的成员属性就不能通过对象实例直接访问，只能在定义的类中使用。如在 setAge 中可以使用 checkAge，在 setAddress 中可以使用 checkAddress。

现在来分析子类中对父类各种属性与方法的访问限制。

```
class college_student extends student{
    public function setData(){
        $this->name='php 成都';
        $this->age=18;
        $this->address='成都';
        echo $this->getAddress();   //无输出，这时的 getAddress 取的是父类的 address，
                                    //但没有值
        echo $this->address;        //输出"成都"
    }
    public function checkData(){
        if($this->checkName($this->getName())){
            echo "姓名正确";
        }
        if($this->checkAge($this->getAge())){
            echo "年龄正确";
        }
        //Call to private method student::checkAddress() from context
        if($this->checkAddress($this->getAddress())){
            echo "地址正确";
        }
    }
}
$college_student=new college_student();
//测试直接访问继承的父类的属性
$college_student->name="php 成都";
//$college_student->age=18;             //出错，错误信息 Cannot access protected property
                                        //college_student::$age
//$college_student->address="成都";     //出错，错误信息 Fatal error: Cannot access private
                                        //property student::$address
//$college_student->checkAge($student_one->getAge());  //出错，错误信息 Fatal error: Call to
```

```
protected method student::checkAge() from context
    //$college_student->checkAddress($student_one->getAddress());   //出错，错误信息 Fatal
                                                                    //error: Call to protected
                                                                    //method student::
                                                                    //checkAddress() from
                                                                    //context
    $college_student->setData();        //输出"成都"
    $college_student->checkData();      //输出“姓名正确”与“年龄正确”后，执行到
                                        //if($this->checkAddress($this->getAddress()))时
                                        //抛出错误
```

通过上例可以看到，在子类中，对父类资源的引用可归纳如下：对于用 public 修饰的属性与方法，使用上与父类一致，可在类里使用，也能通过对象直接使用；对于用 private 修饰的属性与方法，则无法在子类中直接使用；对于 protected 修饰的属性与方法，可在子类中使用，但不能通过对象使用。

本例中还有一个很值得注意地对方，就是 private 修饰的成员属性与成员方法对于子类来说相当于透明。在 college_student 的 setData 方法中，使用了$this->address='成都'，但这句里的 address 与其父类中的 address 无任何关系，因此通过 echo$this->getAddress()得不到任何值，而是直接通过 echo $this->address;输出"成都"，这是 PHP 运行时刻的动态成员属性定义功能，将在下一章中专门讲解。

1.2.4　构造函数与析构函数

构造函数与析构函数是类的两个特殊成员方法，当类被实例化时，构造函数被自动调用，当实例执行完成或是页面被释放时，析构函数被自动调用。因此可以利用这两个函数的特殊属性把初始化的工作交给构造函数，把释放资源等收尾工作交给析构函数去完成。

```
class news{
    private $link;
    //在构造函数里连接数据库
    public function __construct($host,$user,$password,$database){
        $this->link=mysql_connect($host,$user,$password);
        mysql_select_db($database);
        return true;
    }
    //新增信息的动作
    public function newsAddAction(){
        $rs=mysql_query("select * from news",$this->link);
    }
    //显示信息的动作
    public function newsDisplayAction(){
```

```
    }
    //编辑信息的动作
    public function newsEditAction(){

    }
    //信息列表的动作
    public function newsListAction(){

    }
    //删除信息的动作
    public function newsDeleteAction(){

    }
    //释放数据库连接
    public function __destruct(){
        mysql_close ($this->link);
    }
}
$news=new TNews('localhost','root','','news');
```

上述代码片段展示了构造函数与析构函数的使用，需要注意的是，构造函数与析构函数并不一定需要成对出现，主要是看当前功能本身的要求。此外，构造函数本身支持形参，因此可在实例化类的时候传参数给构造函数，例如：

```
$news=new TNews('localhost','root','','news');
```

利用构造函数的传参能力，大大提高了面向对象的灵活性与解决实际问题的能力。

1.3 静态成员属性与成员方法

静态成员属性与静态成员方法主要用于需要在类的不同实例间传递数据的场景。比如共享数据库连接或统计访问量。静态成员属性与方法还是单例模式得以实现的基础，关于单例模式将在下一章专门介绍。

通过静态成员属性与静态成员方法存储数据库连接：

```
class news{
    //定义静态成员变量
    public static $link;
    public static $db;
    //定义静态成员函数
    public static function connect(){
        self::$link=mysql_connect('localhost','root','');
        self::$db=mysql_select_db('news');
```