DOI:10.19344/j.cnki.issn1671-5276.2019.06.033

MVC与设计模式在测控系统中的应用

耿红杰,徐超,赵世平

(四川大学 机械工程学院,四川 成都 610065)

摘　要:针对目前基于结构化或者面向对象程序设计方式测控系统的不足,采用一种模型-视图-控制器模式对软件架构进行重新开发。采用多种设计模式对系统软件模块进行设计,缩短了软件系统的开发周期,实现了软件模块的有效复用,提升了软件的可维护性、可扩展性。

关键词:测控系统;MVC;设计模式;可复用性

中图分类号:TP311.52　　**文献标志码**:B　　**文章编号**:1671-5276(2019)06-0127-03

Application of MVC and Design Pattern in Measurement and Control System

GENG Hongjie, XU Chao, ZHAO Shiping

(School of Mechanical Engineering, Sichuan University, Chengdu 610065, China)

Abstract: To overcome the shortcomings of current measurement and control system based on structured or object-oriented programming mode, a MVC model is used to redevelop the software architecture and a variety of design patterns are used to design the software module of the system, thus shortening the development cycle of the software system, realizing the effective reuse of the software module and improving the maintainability and extensibility of the software.

Keywords: measurement and control system; MVC; design patterns; reusability

0　引言

在早期的测控系统软件设计中[1],结构化程序设计(又称为面向过程的程序设计)是经常使用的一种设计方法,通过分析出解决问题所需的步骤后,用函数将这些步骤实现并按顺序调用,进而实现需求。这种自顶而下的设计方式具有很强的专用性,只适用于某种特定的场合,一旦用户需求发生变化,可能会导致整个系统的重新修改,需花费大量的时间、精力去重复相同的工作。随着面向对象技术的问世及发展[2],虽然结构化程序设计的不足得到了部分的弥补,但由于其思想只是将构成问题的事务抽象为各个对象,系统的输入、输出、数据处理、UI界面等模块的程序仍是相互混杂,需求的改变仍会造成系统大量的修改,增加额外的开发成本,并且不利于软件的复用和后期的维护。MVC(model-view-controller,模型-视图-控制器)模式将系统划分为相互独立的单元[3],使得数据处理与UI界面模块可以同时进行设计[4],缩短了系统的开发周期,并且大大减少了程序之间的耦合度,降低了系统的复杂度和维护难度。

本文根据某液压厂对某些液压产品超高压耐压性能测试的需求,研制了一套机电液一体化的测控系统,实现了对液压产品的高适应性、超高压自动控制。该系统采用了MVC框架并且在软件模块中采用了软件工程中的单例模式、适配器模式两种设计模式进行设计[5],从而提升了系统的可复用性以及可扩展性。

1　MVC模式与设计模式介绍

1.1　MVC模式简介

MVC模式将软件系统分为3个部分:Model(模型)、View(视图)、Controller(控制器),模型如图1所示[6]。

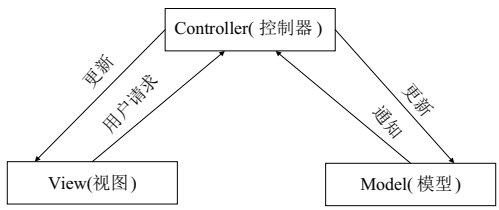

图1　MVC模型

图1中,Model对象代表了应用程序的业务数据和业务逻辑。业务数据通常用来存放系统数据,比如订单信息、用户信息等;业务逻辑主要包含业务操作的应用,比如订单的添加或者修改等。

View对象用来将模型的内容展现给用户,用户可以通过视图来请求模型进行更新。视图从模型获得要展示的数据,然后用自己的方式展现给用户,相当于提供界面来与用户进行人机交互;用户在界面上操作或者填写完成后,可点击提交按钮或是以其他触发事件的方式,来向控制器发出请求。

Controller对象用来控制应用程序的流程和处理视图

作者简介:耿红杰(1995—),男,江苏盐城人,硕士研究生,研究方向为仪器仪表工程。

所发出的请求。当控制器接收到用户的请求后,会将用户的数据和模型的更新相映射,也就是调用模型来实现用户请求的功能;然后控制器会选择用于响应的视图,把模型更新后的数据展示给用户。

在 MVC 模式中,模型和视图是分离的,通常视图里面不会有任何逻辑实现,而模型也不依赖于视图。同一个模型可能会有很多种不同的展示方式,同一个视图里可能也会有多种不同的模型,它们仅仅是通过 Controller 联系在一起。在这种模式下,系统的耦合度大大降低,从而增强了系统的可维护性。

1.2　设计模式简介

设计模式是软件开发人员在软件开发过程中面临的一般问题的解决方案[7-8]。这些解决方案是众多软件开发人员经过相当长的一段时间的试验总结出来的。使用设计模式可以使得程序更易于编写和修改[10],更容易被他人理解,保证代码的可靠性,并且可以提升软件的可复用性。一般而言,1 个模式具有 4 个基本要素:模式名称(pattern name)、问题(problem)、解决方案(solution)、效果(consequences)。

1) 单例模式

单例模式是 1 种创建型模式,它的意图是确保 1 个类只有 1 个实例,并提供 1 个访问它的全局访问点。通常,普通类的构造函数是公有的,外部类可以通过“new 构造函数()”来生成多个实例。由于产生了多个实例,很可能造成对多个对象管理不当而引起的程序异常后果。单例模式则是在类内部定义 1 个静态私有实例,并向外提供 1 个静态的公有函数用于获取该类的私有实例,这就可以保证整个程序内只存在唯一的 1 个实例。

2) 适配器模式

适配器模式是 1 种结构型模式,它的意图是将 1 个类的接口转换成客户希望的另外 1 个接口,使得原本由于接口不兼容的类可以一起工作。适配器模式一共包含 3 个角色:Target(目标抽象类)、Adapter(适配器类)、Adaptee(适配者类),其中 Target 类定义客户所需的接口;Adaptee 类定义 1 个已经存在的接口,这个接口需要配置包含了客户希望的业务方法;Adapter 类则作为 1 个转换器,对 Adaptee 和 Target 进行配置,是适配器模式的核心。

2　MVC 模式与设计模式在测控系统中的应用

2.1　MVC 模式在测控系统中的具体实现

在某试验台测控系统中,需要实时采集压力传感器的数值,进而对试验进行控制。该系统软件主要由 UI 界面、数据采集系统、试验控制系统和安全监测系统构成。MVC 模式在该软件的搭建过程中将软件分为 UI 界面、试验过程控制器、系统模型等几个部分[10],具体设计如下:

Model:在该测控系统软件中,核心 Model 便是压力传感器的数据,主要包括数据的获取、处理以及相关操作和逻辑。设计类 HighSpeedCollect 用来完成对数据的操作。在该类中定义方法 Start()并在该方法中调用函数 GetProductPressure()。在 GetProductPressure()方法中使用函数 device.ReadAi(0,ref value)调用板卡驱动实现采集并将采集到的传感器数据放入 value 中,value 中的数据即是 Model 之一。GetProductPressure()方法如下:

```
Public int GetProductPressure(ref double value){
Lock(monitor){
If(device.ReadAi(0,ref value))
Return 1;
Return 0;}}
```

View:在该系统中,“ExpRibbonForm”即是 View 部分,该界面实现了数据的显示和人机交互的功能,如图 2 所示。数据显示主要包括压力传感器的信号,压力随时间的曲线以及食品监控等;人机交互主要是通过点击界面中的 Button、RadioButton 等来实现用户的 Action 操作。

图 2　View 对象

Controller:在测控系统软件中,Model 和 View 是独立的,两者之间互不影响,需要通过 Controller 来粘合,而 Controller 的任务就是将 View 的 User Action 转发到 Model,并将 Model 的 Notify 呈现在 View 上。在该系统中,主要在“ExpRibbonFormControl”类中实现。

当用户在 View 上有操作时,如点击按钮等,通过事件(Event)机制实现将 View 向 Model 的转发[11]。在”ExpRibbonFormControl”类中定义各个操作的 Controller 操作,即下面代码中的 btnNewExp_ItemClick(object sender,ItemClickEventArgs e)等。当用户操作触发后,注册在其触发事件上的 Controller 操作也会相应激活,根据 Controller 操作内部的逻辑操作从而实现 View-Controller-Model 的转变。

```
Public void defineBehavior(){
expRibbonForm.btnNewExp.ItemClick+=this. btnNewExp_ItemClick;
expRibbonForm.btnExpRunStop. ItemClick+=this. btnExpRunStop_ItemClick;
expRibbonForm.btnRefreshStata.ItemClick+=this. btnRefreshStata_ItemClick;
//...省略部分代码}
#region Controller 操作
Public void btnNewExp_ItemClick(object sender,ItemClickEven-
```

```
tArgs e){
    //...相应 Model 操作}
    #endregion
```

在 Model 中,当采集到系统数据时,使用事件机制将数据传输到 HighSpeedCollectEventArgs 类中,代码如下所示,并在“ExpRibbonFormControl”类中利用 Timer 将 HighSpeedCollectEventArgs 中的数据更新到 View 中的 Label、TextBox 等控件中从而实现 Model-Controller-View 的更新。

```
Public void Start(){
//...省略部分代码
highSpeedCollectEventArgs.PumpPressure=PumpPressure;
highSpeedCollectEventArgs.ProductPressure=ProductPressure;
If(CollectThrowDataEvent! =null){
CollectThrowDataEvent(this,highSpeedCollectEventArgs);}}
```

2.2　设计模式在测控系统中的具体实现

1) 基于单例模式的设计

在本试验台的测控系统中,获取压力传感器的数据是其他一切工作的前提,这里采用了 PCI1716 采集卡对这一模拟量进行采集。为了系统的稳定性,在软件设计时应确保采集卡类的实例只有 1 个,如果产生了多个实例,则可能会造成对设备的误操作,从而影响试验台的正常运行。为了满足这一要求,采用了单例模式来对模块进行设计。

在 PCI1716 类中,首先定义了 1 个私有构造函数 private PIC1716()[12],这可以保证外部类无法通过 new PCI1716()来创建实例,接着通过定义 1 个公有静态函数 GetInstance()使得外部可以获得该类的唯一实例。该方法在第一次调用时创建了类的唯一实例,在之后调用中则返回该实例,这样可以保证设备对象从始至终只被创建 1 次,并且程序各部分都可以方便地获取设备对象进行操作。具体代码如下:

```
private static PCI1716 instance=null;
public static PCI1716 GetInstance(){
if(instance==null){
lock(syscRoot){
if(instance==null){
instance=new PCI1716();}}}
return instance;}
```

2) 基于适配器模式的设计

如前所说,对压力传感器的数据获取是通过操作 PCI1716 采集卡所得,而对 PCI1716 的操作主要是调用相应的 API 函数,但是如果直接在软件中对这些 API 函数进行调用,则程序与底层的 API 函数耦合过于紧密。如果需求发生了改变,比如采集频率的要求提高,那么就需要重新更换采集设备,相应的软件也需要大面积更改,不利于软件的可维护性和可扩展性。因此,希望能够将这些 API 函数进行封装,提供接口供上层调用,就可以降低耦合性。为此,采用了适配器模式进行设计。

按照适配器模式设计的采集模块类图如图 3 所示,其中 Automation.BDaq 类是其公司提供的 PCI1716 设备的 API 库函数,属于 Adaptee 类;IDevice 接口属于 Target 类[13],上层软件通过操作该接口来操作底层的设备。该接口中定义了 Open()、Close()、ReadAi()、WriteAo() 4 种方法和 IsReady 属性,分别对应启动初始化设备、关闭设备、获取采集卡的数据、向采集卡输出数据以及判断采集卡是否可以工作。PCI1716 类属于 Adapter 类,它继承自 IDevice 接口,通过调用供货商给的 API 函数来实现 Target 类的 5 种方法和属性,从而实现操作采集卡的目的。

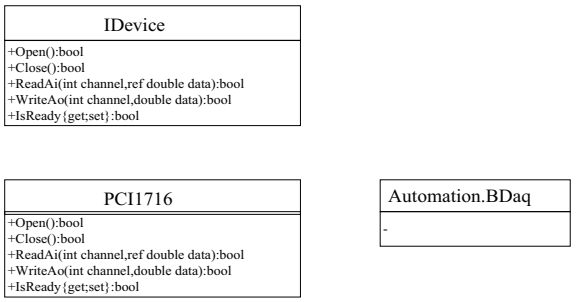

图 3　基于适配器模式的采集模块类图

3　结语

本文所采用的 MVC 模式能够很好地解决以往测控系统软件设计中耦合过高的问题,实现了 View 模块和 Model 模块的分离,极大地增加了系统的可维护性。采用单例模式、适配器模式等设计模式对软件模块进行设计,增强了系统的稳定性,并且提升了系统的可复用性和可扩展性。

参考文献:

[1] 田培培,金晓怡,黄立新,等. 基于 LabVIEW 的微扑翼机实验台测控系统设计[J]. 机械制造与自动化,2018,47(2):183-186.
[2] 石博文. 浅谈面向对象和面向过程程序设计[J]. 电子世界,2017(1):59-60.
[3] 梁远. 基于 MVC 的地下水监测系统的研究[D]. 淮南:安徽理工大学,2017.
[4] 王越,夏京川,祁正兴,等. 基于现代并发技术的测控系统软件设计[J]. 机械制造与自动化,2018,47(5):118-120,163.
[5] 陈天超. 单例设计模式研究[J]. 福建电脑,2016(8):14-15.
[6] 张勇军,熊庆国,黎学文. 基于 MVC 的项目管理系统设计与实现[J]. 自动化与仪表,2015,30(11):58-61.
[7] ERIC Freeman,BERT Bates. Head First 设计模式[M]. 北京:中国电力出版社,2004:91-93.
[8] 刘伟,胡志刚,阎明坤. C#设计模式[M]. 北京:清华大学出版社,2013:113-114.
[9] 杨静. 设计模式在敌我识别信号侦测显控软件的应用[J]. 电子设计工程,2019(4):38-42.
[10] 黄永平. 基于 MVC 架构的配电网基建工程项目管理系统的开发与设计[J]. 自动化与仪器仪表,2018(7):105-108.
[11] 唐磊. 基于 C#事件机制的自定义控件开发研究[J]. 电脑编程技巧与维护,2018(10):21-22.
[12] 何泓伟. 设计模式混合的构造方法研究及应用[J]. 计算机工程与设计,2007(5):999-1001,1042.
[13] 陈天超. 接口在设计模式中的应用[J]. 自动化与仪器仪表,2013(5):112-113.

收稿日期:2019-06-25