第 22 章　模型-视图-控制器（MVC）框架

在前面章节中，读者已经了解到了两种开发 JSP 的方式。第一种是纯 JSP 文件形式，这种方式只适合开发超小规模的 Web 应用，它会显得代码零乱、不易后期维护和扩展；第二种是使用 JavaBean 来改进 Web 应用开发过程，JavaBean 可以实现代码重用，使逻辑业务操作代码和显示代码适当分离。在第 12 章介绍的实例中，已经根据 JSP 页面所承担的功能进行了适当分离，例如 index.jsp、register.jsp 和 login.jsp 页面更多承担视图层功能；而 chklogin.jsp 和 do_register.jsp 页面则更多承担控制层功能。后来引用 JavaBean，JavaBean 则相当于数据模型层，但是 JSP＋JavaBean 还是属于 Model1 模式开发（虽然有人建议起名为 Model1.5），还没有彻底实现 MVC 模式开发，也只能适合中小型 Web 开发。这一章将向读者介绍 MVC（模型-视图-控制器）三层开发模式的概念、原理和过程。

本章要点包括以下内容：

- 框架的概念
- 纯 JSP 文件开发方式
- JSP＋JavaBean 开发方式
- Model2（MVC）模式概念
- Model1 与 Model2 模式的比较
- 一个简单的 MVC 实例

22.1　什么是框架

在了解 MVC 之前，先明白什么是框架（Framework）。框架是一种可重用的、半完成的应用程序体系，开发者可以使用它来快速地生成专门的定制程序。

如果你对各种应用程序有过详细研究，就会发现其中用于应用程序的组件大致有两种类型：一类是专门处理相关事务的，可以把它成为业务组件，这些组件由于要处理的业务性质各不相同而不能得到很好的重用。例如税务系统和图书系统，由于要处理的业务不一样，所以它们的组件并不能在更大范围内得到重用；另一类组件是与程序流程的控制、输入校验、错误处理以及标签库等相关，这些组件都只是与程序本身相关而与系统需要处理什么业务并没有关联，所以在所有的应用系统能得到很好的重用。

随着软件技术的发展和需要，人们自然想到要把一些在不同应用系统中的共性东西抽取出来，做成一个半成品程序，这样的半成品程序就是所谓的程序框架。在程序开发过程中使用框架的好处：

- 可以使新手很容易上手。
- 开发一个新的系统没有必要每次都白手起家，而是可以在这个框架基础上开始搭建。
- 使得整个应用的结构清晰明朗，易于后期的维护和扩展。
- 如果框架标准化之后，有利于开发者的沟通和交流。

22.2　Model1 模式的概念和原理

Sun 公司总结出了有两种使用 JSP 开发 Web 应用系统的模式（Model）：一种称为 Model1(模式一，例如纯 JSP 或者 JSP+JavaBean 开发的应用程序)；另外一种称为 Model2（模式二，最为经典就是 Struts 技术）。本书之前所举的实例都是基于 Model1 模式进行开发的。

掌握和理解这些模式的使用，是一个开发者设计和构建高质量的 Web 应用程序的必要条件。下面首先认识一下 Model1 的概念和基本原理。

22.2.1　纯JSP文件开发方式

最为直观的使用 JSP 开发 Web 应用的方法即为在 JSP 文件中直接嵌入 Java 语句，以及使用标签库来实现动态 HTML 的目的。例如本书第 12 章的用户注册系统，就是直接使用纯 JSP 实现的，所有的逻辑控制以及业务操作（包括数据操作）都是在 JSP 页面中实现的。这样的开发模式使得 JSP 页面显得非常的混乱，并且不易于系统的后期维护和扩展。使用纯 JSP 开发 Web 应用系统的程序结构如图 22.1 所示。

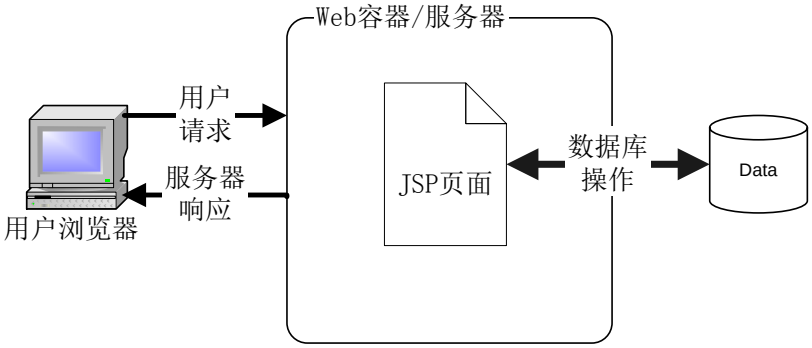

图 22.1　纯 JSP 调用结构

这样的结构优点就是简单方便，适合搭建非常小型的 Web 应用。但是这样模式的缺点是很多的：

- 这样模式开发的 JSP 页面中混合交织了 HTML 和 Java 代码，不仅使得 Web 应用开发带来了极大的困难，而且也给阅读代码带来麻烦。
- 使得系统的后期维护和扩展带来很多困难。例如直接在 JSP 页面进行数据库连接和操作，如果需要对数据库进行任何的修改，都必须打开所有操作数据库的 JSP 页面进行相应的改动。当 Web 应用系统中的页面成千上万时，工作量是相当大的。
- 这种模式下开发的应用系统并不易于调试。由于 HTML、Java 甚至 Javascript 都交织在一起，如果要对某个单独功能进行测试是非常不便的，而且必须要启动服务器，并调用 JSP 页面才能查看运行效果。

在前面章节的实例讲解中，就已经提到这种纯 JSP 开发方式只适用于初学者的入门，在实际应用开发中非常不推荐。

22.2.2　JSP＋JavaBean开发方式

在第 14 章中，本书使用 JSP+JavaBean 的技术对纯 JSP 开发模式进行了很大的改进。使用

JSP+JavaBean 开发的应用程序结构如图 22.2 所示。

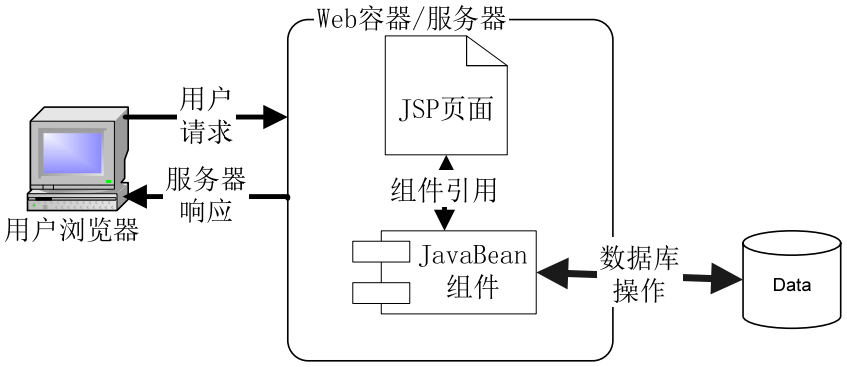

图 22.2　JSP+JavaBean 调用结构

改进之后的开发模式使业务逻辑和数据库操作从 JSP 页面中分离出来，并封装在 JavaBean 中。这样体现出了很多优点：

- 页面更加简洁：因为大量的业务逻辑和数据库操作已经封装在 JavaBean 类中，从而 JSP 页面中只需要嵌入少量的 Java 代码，使得页面比较简洁。
- 可重用的 JavaBean 组件：把共用的或者重要的业务操作使用 JavaBean 类封装起来，当某 JSP 页面需要进行此类操作时，直接调用相应的 JavaBean 类即可。这样大大减少了开发人员的工作量，加快了项目开发的进度。
- 易于后期代码的扩展：正是由于共用的业务操作从 JSP 中分离出来，并由单独的 JavaBean 类封装起来，当需要对业务功能进行修改或者升级时，直接修改相应的 JavaBean 类即可，而没有必要再重新修改和编译所有的调用 JSP。
- 方便进行调试：由于复杂和重要的业务操作都已经封装在一个个 JavaBean 类中，对这些 Java 类进行单独调试即可，甚至没有必要启动 Web 服务器和调动 JSP 页面。

虽然 JSP+JavaBean 已经克服纯 JSP 开发应用的很多缺点，但是还有很多的限制（控制层和视图层还完全是由 JSP 来完成的），并且更多的还是使用在中小型项目中。下面将向读者介绍的是 MVC 编程模式。

22.3　Model2（MVC）模式的概念和原理

正是由于 Model1 模式所存在的不足，当页面之间的程序流非常复杂的时候，修改一个页面可能会影响到很多相关的页面，这时就不适宜使用 Model1 模式开发 Web 应用。

为了克服 Model1 存在的缺陷，人们引入了 Model2 开发模式。Model2 架构基本上就是基于模型－视图－控制器（MVC，Model－View－Control）的设计模式，这样的设计模式集成了 JSP 与 Servlet，非常适合架构复杂的大型 Web 应用程序。如图 22.3 所示的 Model2 程序结构图。

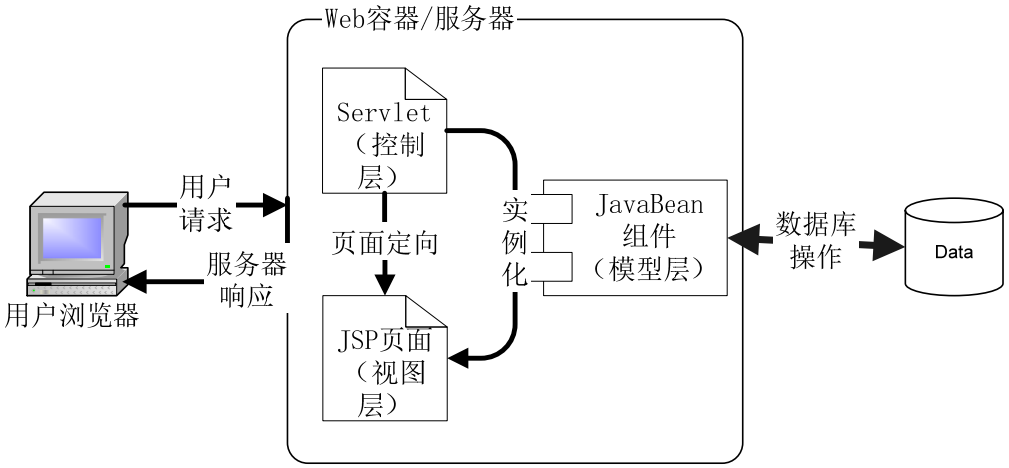

图 22.3　Model2 模式结构

这里的 Servlet 担任了控制器的责任，客户端的请求不再直接交付给一个处理业务逻辑的 JSP 页面，而是送给这个 Servlet 控制器，由这个控制器根据客户端的具体请求调用不同的事务逻辑，并将处理结构返回给合适的页面进行显示。

因此，这里的 Servlet 控制器为应用程序提供了一个进行前－后端处理的中枢。一方面为输入数据进行相应的校验、身份验证以及为实现国际化编程提供一个合适的切入点；另一方面也提供了将业务逻辑从 JSP 文件分离出来的可能。

业务逻辑从 JSP 页面分离出来以后，JSP 文件就完全成为了一个显示数据的层，即 View 视图层。而独立出来的事务逻辑处理类（JavaBean）就成为 Model 模型层，加上 Servlet 控制器 Controller，共同构成了 MVC 模式。下面对 MVC 模式中的各层进行单独讲解：

（1）视图层（View）

代表用户交互界面，在 Web 应用中，可以是 HTML、XHTML、XML 界面或者 Applet 小 Java 程序。使用 MVC 模式设计的视图层仅仅进行数据的采集和处理，以及用户的请求，而业务流程的控制和处理则是交给控制器（Controller）和模型层（Model）。

（2）模型层（Model）

模型层更多是 JavaBean 类编写的，它接受视图层请求的数据，然后进行相应的业务处理，并返回最终的处理结果。模型层担负的责任最为核心，并使得应用程序得到重用和可扩展。

业务模型还有一个比较重要的模型就是数据模型，使用数据模型可以方便地进行实体对象的数据保存。

（3）控制层（Controller）

控制层是从用户端接收请求，将模型和视图匹配在一起，共同完成用户地请求。它的作用就是告诉容器应该选择什么样的视图以及选择什么样的模型。例如，当一个用户点击一个链接时，控制层接受到请求之后，并不直接进行任何的数据操作，而是把用户的请求信息传递给相应的模型层，并告诉模型应该进行什么样的操作，最后根据操作结果选择符合要求的视图返回给请求用户。

控制器在 MVC 设计模式中就是一个中间枢纽的作用，协调着视图和模型层的操作。

22.4　Model1 和Model2 模式比较

Model1 模式开发带来的缺陷在前面小节已经有所介绍，和 Model1 模式相比较，Model2（MVC）具体有如下多个优点：

- 各施其责，互不干涉：在 MVC 模式下，View、Mode 以及 Controller 三层各施其职，所以一旦哪一个层发生变化，只需要对相应层的代码进行修改而不会影响到其他层中的代码。
- MVC 开发模式有利于责任分工：可以根据三层进行责任的划分，网页设计人员可以专门进行视图层中的 JSP 页面开发，而对业务熟练的开发人员进行相应的模型层开发，其他人员开发控制层。责任分工可以提供人员的工作效率。
- 组件可以得到很好的重用：由于分工明确，各层的组件可以独立成一个可重用的组件。

但是使用 Model2 开发模式相对 Model1 来说比较复杂，所以 Model2 比较适合开发大中型 Web 应用，而 Model1（特指 JSP＋JavaBean 开发方式）适合开发中小型 Web 应用。

22.5　MVC简单实例

在 MVC 三层框架中，JSP 已经完全退缩成只有显示（View）功能。而更多使用 Servlet 来实现控制层，处理用户请求，并进行页面重定向。JavaBean 实现数据模型层，大部分逻辑、数据库操作都在这一层完成。下面通过一个简单实例的演示，让加深读者对 MVC 框架和原理的理解。

22.5.1　实现数据模型层（Model）

首先需要创建相应的数据库表 article，为了简单起见，这张表只有两个字段，content 字段保存评论内容；time 为发表时间。详细的 SQL 语句如下：

```
use mysql;
create table article(
    content varchar(100) not null,
    time datetime default '0000-00-00 00:00:00' not null
)TYPE=MyISAM,default character set gbk;

insert into article(content,time)values('小兵传奇','2005-01-22 10:22:35');
insert into article(content,time)values('剑仙游记','2005-11-11 12:02:22');
```

首先使用 create 语句来生成一个 article 数据库表（该表生成在之前章节所创建的 mysql 数据库中）。然后使用 insert 语句来插入两条数据。

下面创建模型层（Model）中的数据库操作类 ArticleCommand.java：

```
package com.simplestruts;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.lang.*;
import java.sql.*;
import javax.sql.*;
import com.simplestruts.CommentItem;
```

```
public class ArticleCommand
{
    public ArticleCommand () {}
    public String getAllArticle(HttpServletRequest request, HttpServletResponse response)
            throws javax.servlet.ServletException, java.io.IOException
    {
      Connection conn=null;
      String con_user = "root";                    //数据库登录用户名
      String con_password = "12345";               //数据库登录密码
      String con_dburl =
      "jdbc:mysql://localhost:3306/mysql?autoReconnect=true&useUnicode=true&characterEncoding=GBK";
        String con_driver = "com.mysql.jdbc.Driver";
        PreparedStatement pstmt=null;
        ResultSet rsComment=null;
        Vector vectorComment = new Vector();
        String selectSQL= "SELECT content, time FROM article ORDER BY time DESC";
        try
        {
            DriverManager.registerDriver(new com.mysql.jdbc.Driver());
            Class.forName(con_driver);
            conn = DriverManager.getConnection(con_dburl,con_user,con_password);
            pstmt=conn.prepareStatement(selectSQL);
            rsComment=pstmt.executeQuery();
            while(rsComment.next())                    //循环读取数据库查询结果
            {
                CommentItem commentItem = new CommentItem();
                commentItem.setContent(rsComment.getString(1));
                commentItem.setTime(rsComment.getDate(2));
                  //将一个 commentItem 实例数据添加到 vectorComment 集合类中
                vectorComment.add(commentItem);
            }
            vectorComment.trimToSize();
          }
        catch (Exception e){}                          //做相应的处理
        request.setAttribute("vectorComment ", vectorComment);
        return "/mvc/showallarticle.jsp";
    }
}
```

代码说明：ArticleCommand.java 封装了数据库连接，取所有 article 信息的方法。将所有 article 数据获取到之后，封装在 vectorComment 集合类中，并通过 request 类发送给客户端进行显示。该类的 getAllArticle()方法最终返回了需要重定向的 URL 地址，交付给 Servlet 控制进行相应处理。

下面创建 article 的数据封装类 CommentItem.java，详细代码如下：

```
package com.simplestruts;
import java.util.Date;
public class CommentItem
{
private String content; //评论内容
private java.sql.Date time; //评论时间
```

```
public CommentItem() {
this.content="";
this.time=null;
}
/**
* 设置评论内容
*/
public void setContent(String content)
{this.content = content;};
/**
* 取得评论内容
*/
public String getContent()
{return content;};
/**
* 设置评论时间
*/
public void setTime(java.sql.Date time)
{this.time=time;};
/**
* 取得评论时间
*/
public java.sql.Date getTime()
{return time;};
}
```

代码说明：该 CommentItem.java 类定义了与 article 数据库表字段相对应的 content 和 time 属性变量。另外，针对这两个属性变量，定义了相对应的 setXxxx()和 getXxxx()方法。

22.5.2　实现控制层（Control）

控制层是使用 Servlet 来实现的，该 Servlet 类的详细代码如下：

```
package com.simplestruts;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import com.simplestruts. ArticleCommand;
public class Controller extends HttpServlet
{
    public void init(ServletConfig config) throws ServletException                    //Servlet 类的初始化方法
    {
        super.init(config);
    }
    public void destroy() {}

    /** 用于处理 HTTP 的 GET 和 POST 请求的函数
    * @param request servlet request
    * @param response servlet response
```

```
    */
    protected void processRequest(HttpServletRequest request,HttpServletResponse response)
        throws ServletException, java.io.IOException
    {
        //代码（1）通过 if 来实现对不同请求的分发
        if(request.getParameter("command").equals("showarticle"))
        {
            ArticleCommand command = new ArticleCommand ();
            String page = command. getAllArticle (request, response);
          //代码（2）
          dispatch(request, response, page);
        }
    }
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)          //定义 get 提交处理方法
        throws ServletException, java.io.IOException
    {
        processRequest(request, response);
    }

    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)         //定义 post 提交处理方法
        throws ServletException, java.io.IOException
    {
        processRequest(request, response);
    }
    /** 一个实现了分发者模式的函数
    * @param request servlet request
    * @param response servlet response
    */
    protected void dispatch(HttpServletRequest request,
                            HttpServletResponse response,
                            String page)
        throws   javax.servlet.ServletException, java.io.IOException
    {
        RequestDispatcher dispatcher =
            getServletContext().getRequestDispatcher(page);
        dispatcher.forward(request, response);
    }
}
```

代码说明：如果用户使用 GET 方式提交请求，则该 Servlet 将执行 doGet()方法，如果使用 POST 方法提交，则执行 doPost()，当这两个方法都相似的执行 processRequest()方法。方法 processRequest()中没有直接处理用户提交的请求，而是将相应数据库操作交给了 ArticleCommand.java。请求处理完之后，Servlet 通过 dispatch()方法进行相应页面重定向，该实例是跳转到 showallarticle.jsp 页面来显示所有的文章信息。

22.5.3　实现视图层（View）

下面创建的 request.jsp 页面，是用来向后台服务器发送查询所有文章信息的请求，该页面程序非常

简单，详细代码如下：

```
<html>
<%@ page contentType="text/html;charset=gb2312"%>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>实例首页</title>
</head>
<body>
<table border="0" width="100%">
  <tr>
<td><div align="center">
<a href="/Controller?command=showarticle">显示文章</a>
</div></td>
  </tr>
</table>
</body>
</html>
```

代码说明：用户单击“显示文章”链接，将向后台发送一个浏览所有文章的请求。

下面再创建用于显示所有文章信息的页面，详细代码如下：

```
<html>
<%@ page contentType="text/html;charset=gb2312"%>
<%@ page import="java.util.*, java.lang.*"%>
<%@ page import="com.simplestruts.CommentItem"%>
<jsp:useBean id="vectorComment" type="java.util.Vector" scope="request"/>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>显示文章</title>
</head>
<body>
<table border="0" width="100%">
  <tr>
    <td>发表时间</td>
        <td>文章内容</td>
  </tr>
<%
    if (vectorComment!=null && vectorComment.size()>0)
    {
        int counter=vectorComment.size();
        CommentItem commentlist = null;
        for (int i=0;i<counter;i++)
        {
            commentlist=null;
            commentlist=(CommentItem)(vectorComment.get(i));
%>
  <tr>
    <td><%=commentlist.getTime()%></td>
        <td><%=commentlist.getContent()%></td>
  </tr>
```

```
<%
        }
    }
%>
</table>
</body>
</html>
```

代码说明：在 ArticleCommand 类中使用 request.setAttribute("vectorComment ", vectorComment)方法将 vectorComment 集合类中的信息发送给客户端；而在客户端使用<jsp:useBean id="vectorComment" type="java.util.Vector" scope="request"/>来获取 id 为 vectorComment 的集合类信息。然后循环输入这个集合类中的所有文章信息，并显示在页面中。

至此该实例还没有完成，还需要在 Web 模块的 web.xml 配置文件中的创建的 Servlet 进行配置。

22.5.4　Servlet配置

将创建的名为 Controller 的 Servlet 控制类在 Web 模块的 web.xml 配置文件进行如下所有的配置：

```
<servlet>
        <servlet-name>Controller</servlet-name>
        <servlet-class>nepalon.simplestruts.Controller</servlet-class>
</servlet>
<servlet-mapping>
      <servlet-name>Controller</servlet-name>
      <url-pattern>/Controller</url-pattern>
</servlet-mapping>
```

在浏览器中浏览 request.jsp 页面，单击“显示文章”来浏览所有的文章信息。

22.5.5　总结

该实例的一个序列图可以显示成如图 22.3 所示。

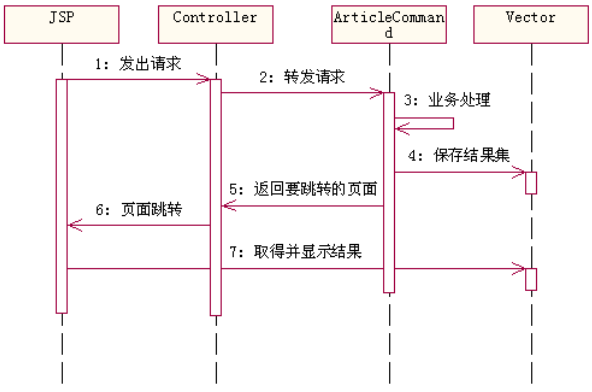

图 22.3　MVC 实例序列图

从该序列图可以看出该 MVC 简单实例的基本执行过程如下：

（1）在 Veiw 层的 test.jsp 中提交一个请求/Controller?command=showarticle。

（2）在 Controller 层的 Controller 对象中，根据请求的类型来调用相应的业务处理类，在这里，

command 值为 showarticle 的请求的业务处理类为 ArticleCommand 类，所以调用该类的对象的相应函数；

（3）在 Model 层的 ArticleCommand 类主要实现请求的取得所有文章的业务功能，把结果保存在 request 中，并返回跳转页面作为返回值；

（4）回到 Controller 层的 Controller 对象，根据上一步骤的返回值进行页面转发。

（5）转发到 View 层的 showallarticle.jsp 页面，这个页面从 request 中取得结果并进行显示。在这个 JSP 中虽然也有 Java 代码，但这些代码只是用于显示结果，并没有涉及到任何业务逻辑。

22.6　本章小结

本章重点向读者介绍了 Model1 和 Model2 模式的概念和原理，并对这两种模型进行了比较。Model1 模式开发的 Web 应用由于代码比较集中，易于管理，但是随着 Web 页面的增多，会显得代码混乱和结构不清晰；Model2 模式实现了三层结构，使得 Web 应用的整体结构清晰，易于后期代码维护和扩展。所以总结起来，Model1 适合开发小型项目，而 Model2 更适合使用在大型 Web 项目开发过程中。本章并以一个简单的 MVC 实例为示范，来加深读者对 MVC 框架的理解。