




软件开发环境

主讲教师 刘凡

fanliu@hhu.edu.cn



第八章

MVC模式



本章主要内容

◆ 8.1 MVC模式介绍

◆ 8.2 JSP中MVC模式

◆ 8.3 模型的生命周期与视图更新

◆ 8.4 MVC的简单实例

8.1 MVC模式介绍

MVC是一种通过3个不同部分构造一个软件或组件的理想办法：

- 模型（Model）——用于存储数据的对象。
- 视图（View）——向控制器提交所需数据、显示模型中的数据。
- 控制器（Controller）——负责具体的业务逻辑操作，即控制器根据视图提出的要求对数据做出处理，并将有关结果存储到模型中，同时负责让模型和视图进行必要的交互，当模型中的数据变化时，让视图更新显示。

8.2 JSP中MVC模式

- **模型（Model）** 一个或多个Javabeen对象，用于存储数据。Javabeen主要提供简单的setXxx方法和getXxx方法，在这些方法中不涉及对数据的具体处理细节，以便增强模型的通用性。
- **视图（View）** 一个或多个JSP页面，其作用是向控制器提交必要的数据和显示数据。JSP页面可以使用HTML标记、Javabeen标记以及Java程序片或Java表达式来显示数据。
- **控制器（Controller）** 一个或多个servlet对象，根据视图提交的要求进行数据处理操作，并将有关的结果存储到Javabeen中，然后servlet使用转发或重定向的方式请求视图中的某个JSP页面显示数据，比如让某个JSP页面通过使用Javabeen标记、Java程序片或Java表达式显示控制器存储在Javabeen中的数据。

8.2 JSP中MVC模式

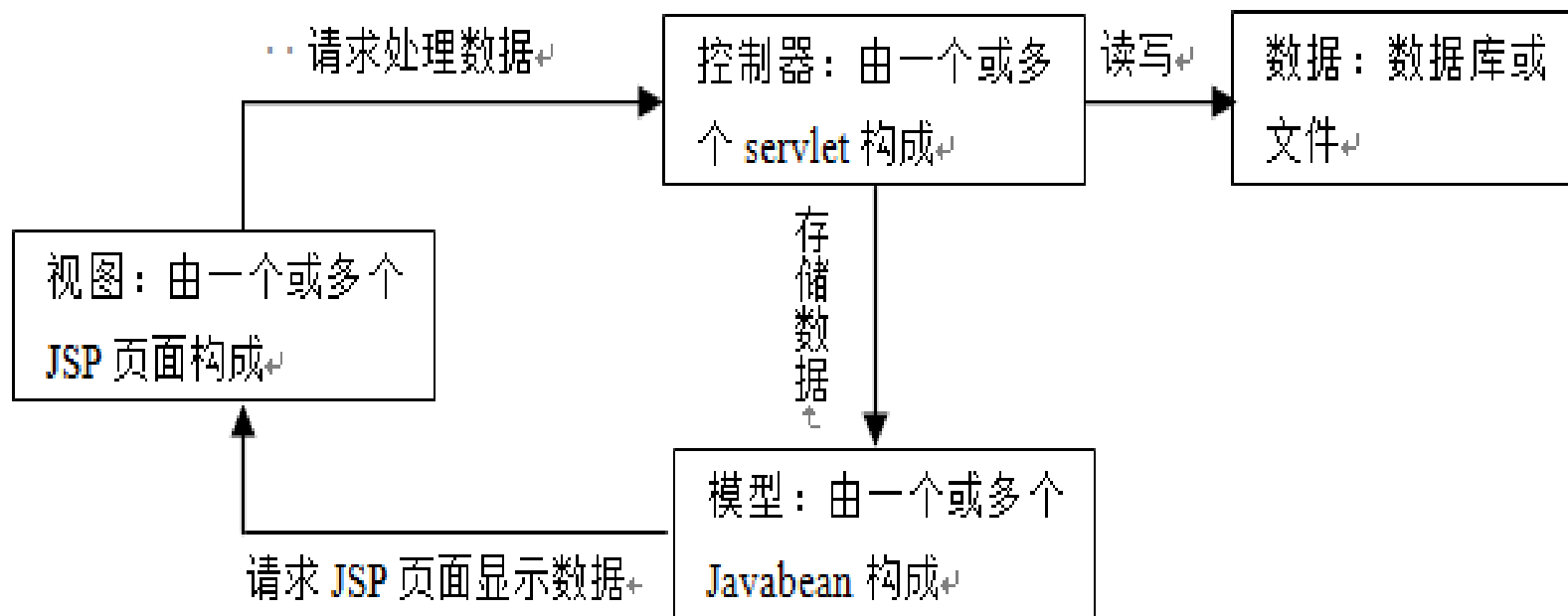


图 6.2 JSP 中的 MVC 模式

8.3 模型的生命周期与视图更新

- 在JSP+Javabeen模式中，由JSP页面通过使用useBean标记创建JavaBean：

`<jsp:useBean id="名字" class="创建bean的类" scope="生命周期"/>`

- 在JSP中的MVC模式中，也可以由控制器servlet创建Javabeen。在JSP中的MVC模式中，servlet创建的Javabeen也涉及到生命周期(有效期限)，生命周期分为request、session和application。
- 当用servlet创建JavaBean时，就可以使用JavaBean类的带参数构造方法，类的方法命名继续保留”get”规则，但可以不遵守”set”规则，因为我们不希望JSP页面修改JavaBean的数据，只需要它显示JavaBean的数据。

8.3 模型的生命周期与视图更新

1、Request周期的JavaBean

➤ JavaBean的创建

(1) 用BeanClass类的某个构造方法创建bean对象，例如：

```
BeanClass bean=new BeanClass();
```

(2) 将所创建的bean对象存放到request中，并指定查找该bean的关键字：
`request.setAttribute("keyWord", bean);`

➤ 视图的更新

servlet使用请求转发让JSP页面显示bean中的数据, bean的周期为request, 因此只对servlet所请求的JSP页面有效。

8.3 模型的生命周期与视图更新

1、Request周期的JavaBean

- JSP页面使用bean

```
<jsp:useBean id="keyWord" type="user.yourbean.BeanClass"  
scope="request"/>
```

或

```
<jsp:useBean id="keyWord" class="user.yourbean.BeanClass"  
scope="request"/>
```

- **type属性**使得JSP页面不负责创建bean
- **class属性**会创建bean对象，但如果servlet已经创建了id为“keyWord”生命周期为request的对象，那么即使JSP页面事先已经创建了id为“keyWord”生命周期为request的bean，这个bean也会被servlet创建的bean对象替换。

8.3 模型的生命周期与视图更新

2、Session周期的JavaBean

➤ JavaBean的创建

(1) 用BeanClass类的某个构造方法创建bean对象，例如：

```
BeanClass bean=new BeanClass();
```

(2) 将所创建的bean对象存放到session中，并指定查找该bean的关键字：
`session.setAttribute("keyWord", bean);`

➤ 视图的更新

servlet使用请求转发或重定向让JSP页面显示bean中的数据, bean的周期为session, 因此只要用户的session没有消失, 该bean就一直存在, 但不同用户的session生命周期的bean是互不相同的。

8.3 模型的生命周期与视图更新

2、session周期的JavaBean

- JSP页面使用bean

```
<jsp:useBean id="keyWord" type="user.yourbean.BeanClass"  
scope="request"/>
```

或

```
<jsp:useBean id="keyWord" class="user.yourbean.BeanClass"  
scope="session"/>
```

- **type属性**使得JSP页面不负责创建bean
- **class属性**会创建bean对象，但如果servlet已经创建了id为“keyWord”生命周期为session的对象，那么即使请求的JSP页面或其他页面事先已经创建了id为“keyWord”生命周期为session的bean，这个bean也会被servlet创建的bean对象替换。

8.3 模型的生命周期与视图更新

3、application周期的JavaBean

➤ JavaBean的创建

(1) 用BeanClass类的某个构造方法创建bean对象，例如：

```
BeanClass bean=new BeanClass();
```

(2) 将所创建的bean对象存放到application中，并指定查找该bean的关键字：

```
getServletContext().setAttribute("keyWord", bean);
```

➤ 视图的更新

servlet使用请求转发或重定向让JSP页面显示bean中的数据，只要Web程序不结束，该bean就一直存在，但不同用户的application生命周期的bean是相同的。

8.3 模型的生命周期与视图更新

3、application周期的JavaBean

- JSP页面使用bean

```
<jsp:useBean id="keyWord" type="user.yourbean.BeanClass"  
scope="application"/>
```

或

```
<jsp:useBean id="keyWord" class="user.yourbean.BeanClass"  
scope="application"/>
```

- **type属性**使得JSP页面不负责创建bean
- **class属性**会创建bean对象，但如果servlet已经创建了id为“keyWord”生命周期为application的对象，那么即使请求的JSP页面或其他页面事先已经创建了id为“keyWord”生命周期为application的bean，这个bean也会被servlet创建的bean对象替换。

8.4 MVC模式的简单实例

本节结合几个简单的实例体现MVC三个部分的设计与实现。

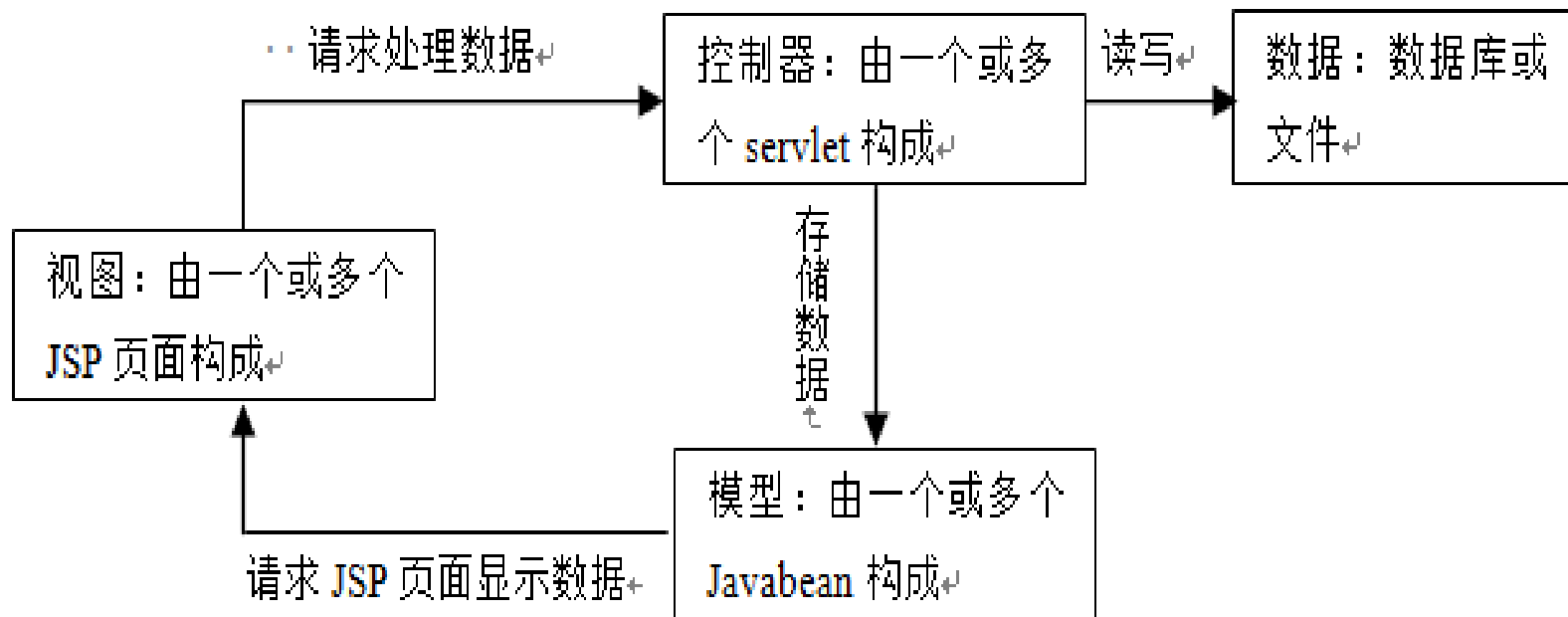


图 6.2-JSP 中的 MVC 模式

8.4 MVC模式的简单实例

- 将Javabeen类和Servlet类源文件分别保存到Web服务目录的下述目录中：

WEB-INF\classes\mybean\data

WEB-INF\classes\myservlet/control

- 然后进入包名的父目录classes, 按如下格式分别编译Javabeen和sevlet的源文件：

classes> javac mybean\data\Javabean的源文件

classes> javac myservlet/control\servlet的源文件

8.4 MVC模式的简单实例

example6_1.jsp

```
<%@ page contentType="text/html;charset=gb2312" %>
```

```
<HTML><body bgcolor=#FFBBFF>
```

```
<font size=2>
```

```
<form action="computeSum" method="post" >
```

等差数列求和:

```
<BR>输入首项:<input type=text name="firstItem" size=4>
```

```
    输入公差:<input type=text name="var" size=4>
```

```
    求和项数:<input type=text name="number" size=4>
```

```
<input type=submit value="提交">
```

```
</form>
```

```
<form action="computeSum" Method="get" >
```

等比数列求和:

```
<BR>输入首项:<input type=text name="firstItem" size=4>
```

```
    输入公比:<input type=text name="var" size=4>
```

```
    求和项数:<input type=text name="number" size=4>
```

```
<input type=submit value="提交">
```

```
</form>
```

```
</font></body></HTML>
```


8.4 MVC模式的简单实例

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
<servlet>
  <servlet-name>computeSum</servlet-name>
  <servlet-
class>myservlet.control.Example6_1_Servlet</servi
et-class>
</servlet>
<servlet-mapping>
  <servlet-name>computeSum</servlet-name>
  <url-pattern>/computeSum</url-pattern>
</servlet-mapping>
</web-app>
```

8.4 MVC模式的简单实例

example6_1_bean.java

```
package mybean.data;

public class Example6_1_Bean{
    double firstItem; //数列首项
    double var;       //公差或公比
    int number;       //求和项数
    double sum;       //求和结果
    String name="";   //数列类别
    public void setFirstItem(double a){
        firstItem=a;
    }
    public double getFirstItem(){
        return firstItem;
    }
    public void setVar(double b){
        var=b;
    }
}
```

8.4 MVC模式的简单实例

example6_1_bean.java

```
public double getVar(){
    return var;
}
public void setNumber(int n){
    number=n;
}
public double getNumber(){
    return number;
}
public void setSum(double s){
    sum=s;
}
public double getSum(){
    return sum;
}
public void setName(String na){
    name=na;
}
public String getName(){
    return name;
}
}
```

8.4 MVC模式的简单实例

- Example6_1_Bean.java中的getXxx和setXxx方法不涉及对数据的具体处理细节，以便增强模型的通用性。比如，setSum(double s)仅仅将参数s的值赋给属性sum，因此，模型即可以存储等差数列的和也可以存储等比数列的和。
- 如果setSum(double s)参与具体的计算，比如，计算等差数列的和，然后将和赋给属性sum，那么该模型就不能存储等比数列的和，减弱了模型的通用性。

8.4 MVC模式的简单实例

example6_1_servlet.java

```
public void doPost(HttpServletRequest request, HttpServletResponse
response)throws ServletException,IOException{
Example6_1_Bean seriesData=new Example6_1_Bean();
request.setAttribute("seriesData",seriesData);//存到request对象中
double a=Double.parseDouble(request.getParameter("firstItem"));
double d=Double.parseDouble(request.getParameter("var"));
int n=Integer.parseInt(request.getParameter("number"));
seriesData.setFirstItem(a);    //将数据存储在数据模型seriesData中
seriesData.setVar(d);
seriesData.setNumber(n);
double sum=0,item=a;
int i=1;
seriesData.setName("等差数列的公差");
while(i<=n){                //计算等差数列的和
    sum=sum+item;
    i++;
    item=item+d; }
seriesData.setSum(sum);
RequestDispatcher dispatcher=
request.getRequestDispatcher("example6_1_show.jsp");
dispatcher.forward(request,response);
}
```

8.4 MVC模式的简单实例

example6_1_servlet.java

```
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException{
    Example6_1_Bean seriesData=new Example6_1_Bean();
    request.setAttribute("seriesData",seriesData);
    double a=Double.parseDouble(request.getParameter("firstItem"));
    double d=Double.parseDouble(request.getParameter("var"));
    int n=Integer.parseInt(request.getParameter("number"));
    seriesData.setFirstItem(a);
    seriesData.setVar(d);
    seriesData.setNumber(n);
    double sum=0,item=a;
    int i=1;
    seriesData.setName("等比数列的公比");
    while(i<=n){                //计算等比数列的和
        sum=sum+item;
        i++;
        item=item*d; }
    seriesData.setSum(sum);
    RequestDispatcher dispatcher=
    request.getRequestDispatcher("example6_1_show.jsp");
    dispatcher.forward(request,response);
}
```

8.4 MVC模式的简单实例

example6_1_show.jsp

```
<%@ page contentType="text/html;charset=gb2312" %>
<jsp:useBean id="seriesData"
    type="mybean.data.Example6_1_Bean" scope="request"/>
<HTML><body bgcolor=#EEFF88>
    <table border=1>
        <tr>
            <th> 数列的首项</th>
            <th> <jsp:getProperty name="seriesData" property="name"/></th>
            <th> 所求项数</th>
            <th> 求和结果</th>
        </tr>
        <tr>
            <td><jsp:getProperty name="seriesData"
                property="firstItem"/></td>
            <td><jsp:getProperty name="seriesData" property="var"/></td>
            <td><jsp:getProperty name="seriesData"
                property="number"/></td>
            <td><jsp:getProperty name="seriesData" property="sum"/></td>
        </tr>
    </table>
</body></HTML>
```

8.4 MVC模式的简单实例

example6_2.jsp

```
<%@ page contentType="text/html;charset=gb2312" %>
<jsp:useBean id="digitBean" class="mybean.data.Example6_2_Bean"
    scope="session"/>
<HTML><body bgcolor=#EFDDFA><font size=2>
    <form action="compute" method=post name=form>
        <table>
            <tr><td> 输入两个数:</td>
                <td> <input type=text name="numberOne" value=0 size=6></td>
                <td> <input type=text name="numberTwo" value=0 size=6></td>
            </tr>
            <tr><td>选择运算符号:</td>
                <td> <select name="operator">
                    <option value="+">+(加)
                    <option value="-">-(减)
                    <option value="*">*(乘)
                    <option value="/">/ (除)
                </select>
            </td>
            <td> <input type="submit" value="提交" name="sub"></td>
        </tr>
    </table>
</form>
```


8.4 MVC模式的简单实例

example6_2.jsp

运算结果：

```
<jsp:getProperty name="digitBean" property="numberOne"/>  
<jsp:getProperty name="digitBean" property="operator"/>  
<jsp:getProperty name="digitBean" property="numberTwo"/> =  
<jsp:getProperty name="digitBean" property="result"/>  
</font></body></HTML>
```

8.4 MVC模式的简单实例

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<web-app>  
  <servlet>  
    <servlet-name>compute</servlet-name>  
    <servlet-class>myservlet.control.Example6_2_Servlet</servlet-class>  
  </servlet>  
  <servlet-mapping>  
    <servlet-name>compute</servlet-name>  
    <url-pattern>/compute</url-pattern>  
  </servlet-mapping>  
</web-app>
```

8.4 MVC模式的简单实例

example6_2_bean.java

```
package mybean.data;
public class Example6_2_Bean{
    double numberOne,numberTwo,result;
    String operator="+";
    public void setNumberOne(double n){
        numberOne=n;}
    public double getNumberOne(){
        return numberOne; }
    public void setNumberTwo(double n){
        numberTwo=n;}
    public double getNumberTwo(){
        return numberTwo; }
    public void setOperator(String s){
        operator=s.trim();}
    public String getOperator(){
        return operator; }
    public void setResult(double r){
        result=r; }
    public double getResult(){
        return result; }
}
```

8.4 MVC模式的简单实例

example6_2_servlet.java

```
public void doPost(HttpServletRequest request, HttpServletResponse
response)throws ServletException,IOException{
    Example6_2_Bean digitBean=null;
    HttpSession session=request.getSession(true);
    try{ digitBean=(Example6_2_Bean)session.getAttribute("digitBean");
        if(digitBean==null){
            digitBean=new Example6_2_Bean(); //创建Javabean对象
            session.setAttribute("digitBean",digitBean);}}
    catch(Exception exp){
        digitBean=new Example6_2_Bean(); //创建Javabean对象
        session.setAttribute("digitBean",digitBean); //存储到session中}
    String str1=request.getParameter("numberOne");
    String str2=request.getParameter("numberTwo");
    if(str1==null || str2==null) return;
    if(str1.length()==0 || str2.length()==0) return;
    double numberOne=Double.parseDouble(str1);
    double numberTwo=Double.parseDouble(str2);
```

8.4 MVC模式的简单实例

example6_2_servlet.java

```
String operator=request.getParameter("operator");
double result=0;
if(operator.equals("+"))
    result=numberOne+numberTwo;
else if(operator.equals("-"))
    result=numberOne-numberTwo;
else if(operator.equals("*"))
    result=numberOne*numberTwo;
else if(operator.equals("/"))
    result=numberOne/numberTwo;
digitBean.setNumberOne(numberOne);    //将数据存储在digitBean中
digitBean.setNumberTwo(numberTwo);
digitBean.setOperator(operator);
digitBean.setResult(result);
//请求example6_2.jsp显示digitBean中的数据
response.sendRedirect("example6_2.jsp"); //重定向
}
```

小结

- MVC模式的核心思想是有效的组合“视图”、“模型”和“控制器”。在JSP 技术中，视图是一个或多个JSP页面，其作用主要是向控制器提交必要的数据和为模型提供数据显示；模型是一个或多个JavaBean对象，用于存储数据；控制器是一个或多个servlet对象，根据视图提交的要求进行数据处理操作，并将有关的结果存储到Javabean中，然后servlet使用重定向或转发方式请求视图中的某个JSP页面更新显示。
- 在MVC模式中，模型也可以由控制器负责创建和初始化。