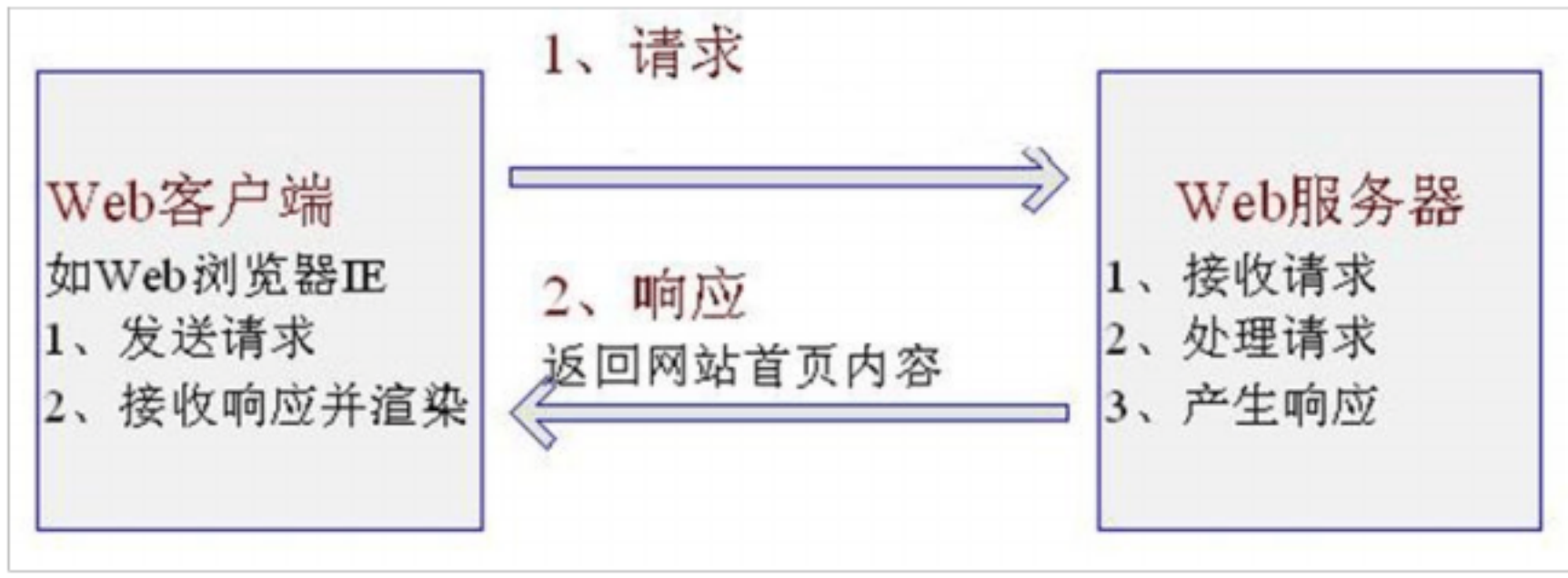
web mvc　简介

Web MVC　简介

Web　开发中的请求　-响应模型：



在 Web　世界里，具体步骤如下：

1、　Web　浏览器（如　IE）发起请求。

2、　Web　服务器（如 Tomcat　）接收请求，　处理请求（比如用户新增，　则将把用户保存一下）　，最后产生响应　（一般为　html　）。

3、web　服务器处理完成后，返回内容给　　web　客户端（一般就是我们的浏览器），客户端对接收的内容进行处理（如　　web　浏览器将会对接收到的　html　内容进行渲染以展示给客户）。

因此，在　Web　世界里：

都是 Web　客户端发起请求，　Web　服务器接收、处理并产生响应。

一般 Web　服务器是不能主动通知　Web　客户端更新内容。虽然现在有些技术如服务器推（如　　Comet　）、还有现在的　HTML5　websocket　可以实现　Web　服务器主动通知　Web　客户端。

到此我们了解了在　web　开发时的请求　/ 响应模型，接下来我们看一下标准的　　MVC　模型是什么。

标准　MVC　模型概述

MVC　模型：　是一种架构型的模式，本身不引入新功能，只是帮助我们将开发的结构组织的更加合理，使展示与模型分离、流程控制逻辑、业务逻辑调用与展示逻辑分离。如图　　1-2

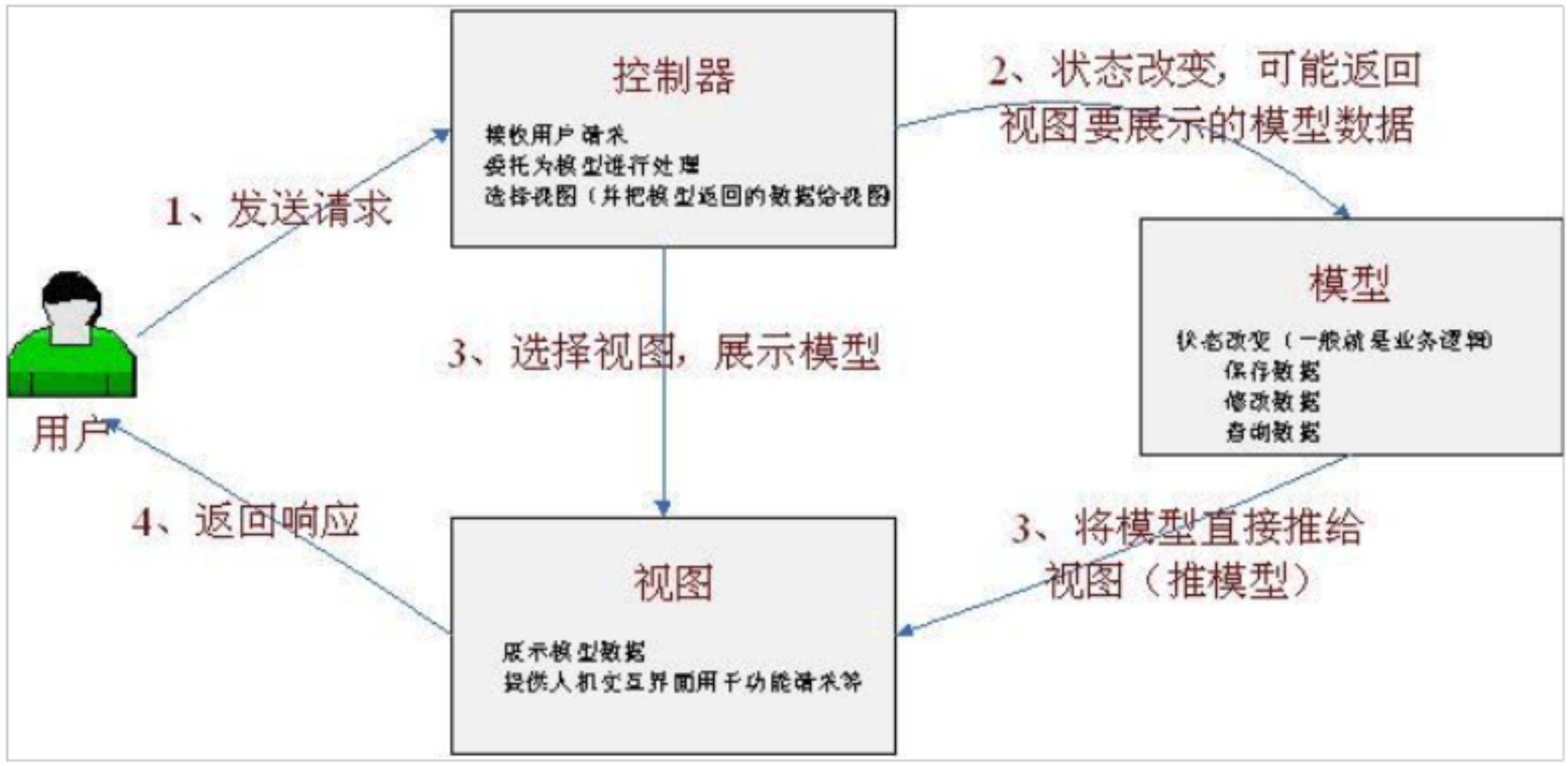


图 1-2

首先让我们了解下 MVC （Model-View-Controller ）三元组的概念：

Model （模型）： 数据模型，提供要展示的数据，因此包含数据和行为，可以认为是领域模型或 JavaBean 组件（包含数据和行为），不过现在一般都分离开来： Value Object （数据） 和 服务层（行为）。也就是模型提供了模型数据查询和模型数据的状态更新等功能，包括数据和业务。

View （视图）： 负责进行模型的展示，一般就是我们见到的用户界面，客户想看到的东西。

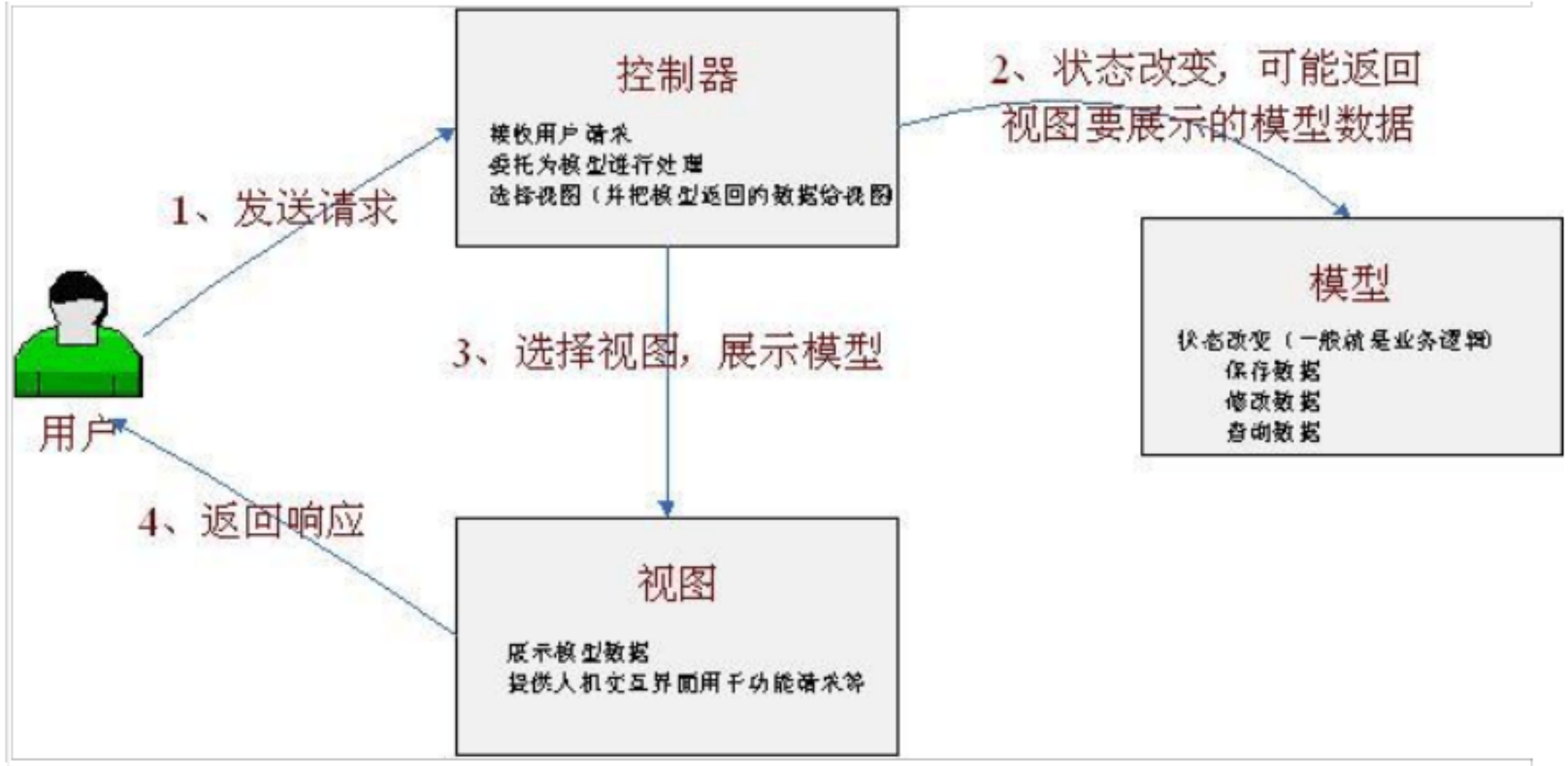
Controller （控制器）： 接收用户请求，委托给模型进行处理（状态改变），处理完毕后把返回的模型数据返回给视图，由视图负责展示。 也就是说控制器做了个调度员的工作，。

从图 1-1 我们还看到，在标准的 MVC 中模型能主动推数据给视图进行更新（观察者设计模式，在模型上注册视图，当模型更新时自动更新视图），但在 Web 开发中模型是无法主动推给视图（无法主动更新用户界面），因为在 Web 开发是请求 - 响应模型。

那接下来我们看一下在 Web 里 MVC 是什么样子，我们称其为 Web MVC 来区别标准的 MVC 。

Web MVC 概述

模型 - 视图 - 控制器概念和标准 MVC 概念一样，请参考 1.2，我们再看一下 Web MVC 标准架构，如图 1-3 ：



如图 1-3

在 Web MVC 模式下，模型无法主动推数据给视图，如果用户想要视图更新，需要再发送一次请求（即请求 -响应模型）。

概念差不多了，我们接下来了解下 Web 端开发的发展历程，和使用代码来演示一下 Web MVC 是如何实现的，还有为什么要使用 MVC 这个模式呢？

Web 端开发发展历程

此处我们只是简单的叙述比较核心的历程，如图 1-4

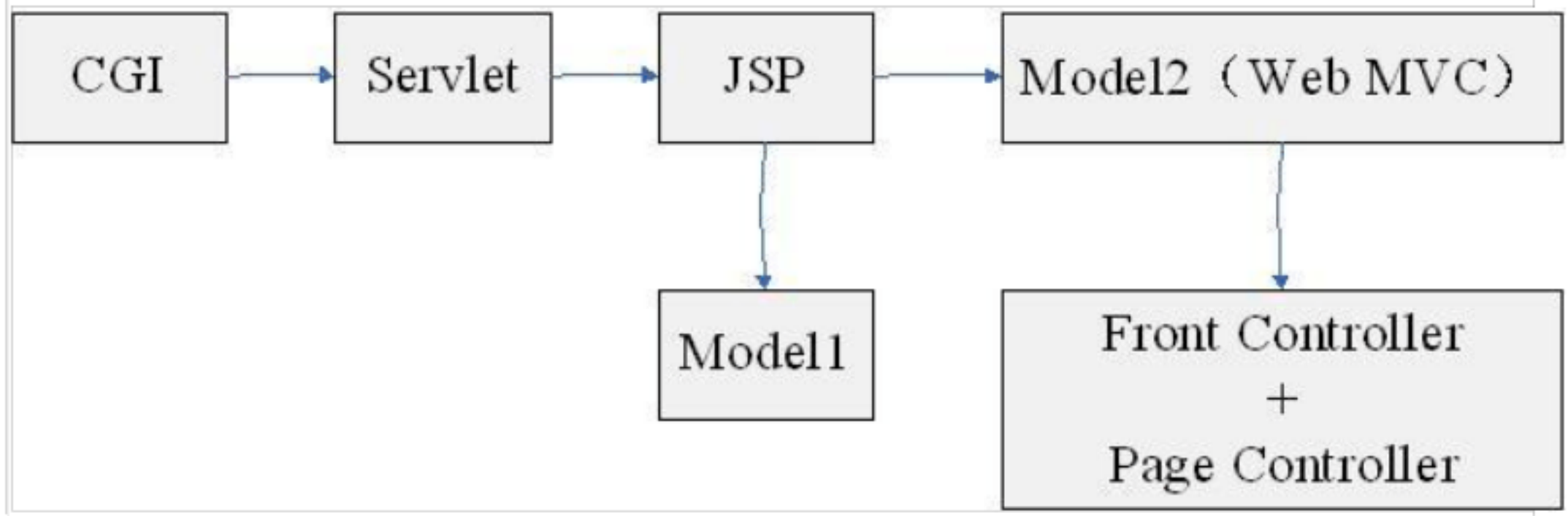


图 1-4

CGI：（Common Gateway Interface ）公共网关接口，一种在 web 服务端使用的脚本技术，使用 C 或 Perl 语言编写，用于接收 web 用户请求并处理，最后动态产生响应给用户，但每次请求将产生一个进程，重量级。

Servlet ：一种 JavaEE web 组件技术，是一种在服务器端执行的 web 组件，用于接收 web 用户请求并处理，最后动态产生响应给用户。但每次请求只产生一个线程（而且有线程池），轻量级。而且能利用许多 JavaEE 技术（如 JDBC 等）。本质就是在 java 代码里面 输出 html 流。但表现逻辑、控制逻辑、业务逻辑调用混杂。如图 1-5

```
public class LoginServlet extends HttpServlet {
        @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        doPost(req, resp);//为了简单，直接委托给 doPost进行处理
    }
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
    String submitFlag = req.getParameter("submitFlag");
    if("toLogin".equals(submitFlag)) {
        toLogin(req, resp);return;
    } else if("login".equals(submitFlag)) {
        login(req, resp);return;
    }
    toLogin(req, resp); //默认到登录页面
}
private void toLogin(HttpServletRequest req, HttpServletResponse resp)
    throws IOException {
    resp.setContentType("text/html");
    String loginPath = req.getContextPath() + "/servletLogin";
    PrintWriter write = resp.getWriter();
    write.write("<form action='" + loginPath + "' method='post'>");
    write.write("<input type='text' name='submitFlag' value='login'/>");
    write.write("username:<input type='text' name='username'/>");
    write.write("password:<input type='password' name='password'/>");
    write.write("<input type='submit' value='login'/>");
    write.write("</form>");
}
private void login(HttpServletRequest req, HttpServletResponse resp)
    throws IOException {
    //1收集参数
    String username = req.getParameter("username");
    String password = req.getParameter("password");
    //2验证并封装参数（重复的步骤）
    UserBean user = new UserBean();
    user.setUsername(username);
    user.setPassword(password);
    //3调用javabean对象（业务方法）
    if(user.login()) {
            //4根据返回值选择下一个页面
            resp.getWriter().write("login success");
    } else {
            resp.getWriter().write("login fail");
    }
  }
}
```

1、控制逻辑：根据请求参数选择要执行的功能方法

2、表现代码：页面展示直接放在我们的servlet里边

3、调用业务对象(javabean对象)进行登录：即模型，不仅包含数据还包含行为

图 1-5

如图 1-5，这种做法是绝对不可取的，控制逻辑、表现代码、业务逻辑对象调用混杂在一起，最大的问题是直接在 Java 代码里面输出 Html ，这样前端开发人员无法进行页面风格等的设计与修改， 即使修改也是很麻烦， 因此实际项目这种做法不可取。

JSP：（Java Server Page ）：一种在服务器端执行的 web 组件，是一种运行在标准的 HTML 页面中嵌入脚本语言（现在只支持 Java）的模板页面技术。 本质就是在 html 代码中嵌入 java 代码。JSP 最终还是会被编译为 Servlet ，只不过比纯 Servlet 开发页面更简单、方便。但表现逻辑、控制逻辑、业务逻辑调用还是混杂。如图 1-6

```jsp
<%@page import="cn.javass.chapter1.javabean.UserBean"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>登录</title>
</head>
<body>
<%
    String submitFlag = request.getParameter("submitFlag");
    if("login".equals(submitFlag)) {//登录
        //1收集参数
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        //2验证并封装参数
        UserBean user = new UserBean();
        user.setUsername(username);
        user.setPassword(password);
        //33调用javabean对象（业务方法）
        if(user.login()) {
            //4根据返回值选择下一个页面
            out.write("login success");
        } else {
            out.write("login fail");
        }
    } else {
%>

                <form action="" method="post">
                    <input type="hidden" name="submitFlag" value="login"/>

                    username:<input type="text" name="username"/><br/>
                    password:<input type="password" name="password"/><br/>
                    <input type="submit" value="login"/>
                </form>

<%          } %>

</body>
</html>
```

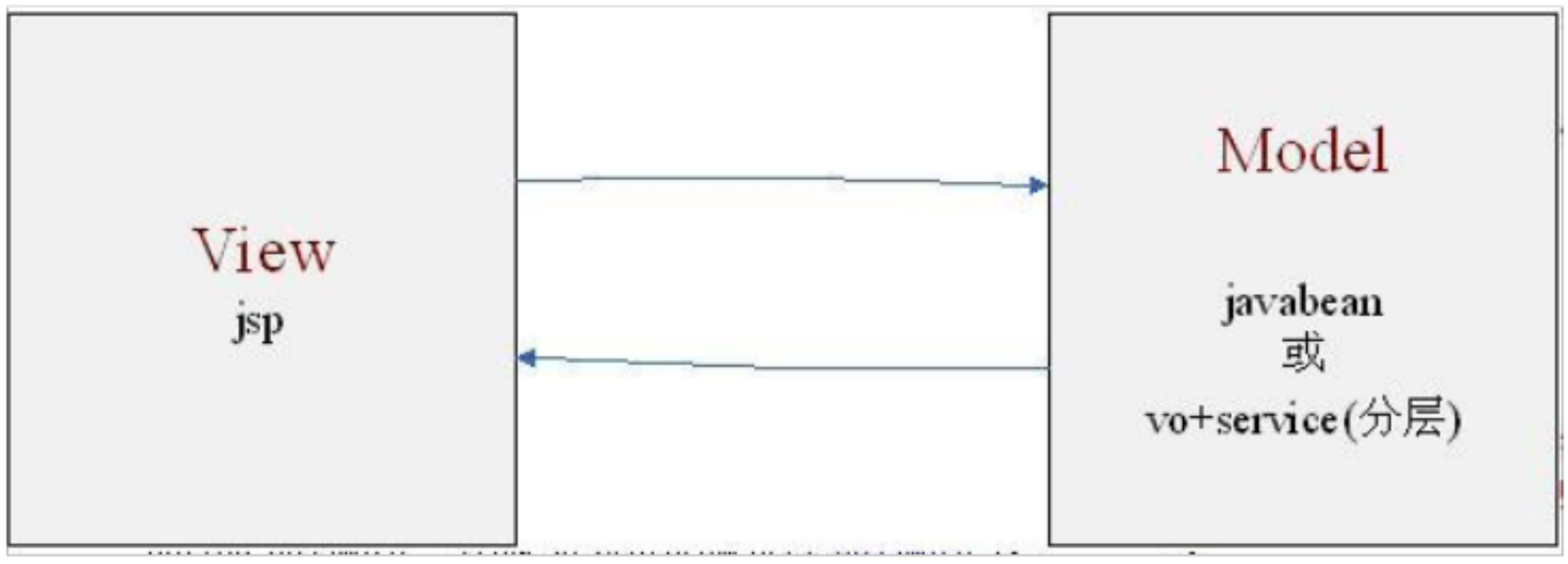
1、控制逻辑：根据请求参数选择要执行的功能方法

3、调用业务对象(javabean对象)进行登录：即模型，不仅包含数据还包含行为

表现代码：页面展示直接放在我们的servlet里边

图 1-6

如图 1-6，这种做法也是绝对不可取的，控制逻辑、表现代码、业务逻辑对象调用混杂在一起，但比直接在 servlet 里输出 html 要好一点，前端开发人员可以进行简单的页面风格等的设计与修改（但如果嵌入的 java 脚本太多也是很难修改的），因此实际项目这种做法不可取。



JSP 本质还是 Servlet，最终在运行时会生成一个 Servlet（如 tomcat，将在 tomcat\work\Catalina\web 应用名\org\apache\jsp 下生成），但这种使得写 html 简单点，但仍是控制逻辑、表现代码、业务逻辑对象调用混杂在一起。

Model1 ：可以认为是 JSP 的增强版，可以认为是 jsp+javabean 如图 1-7

特点：使用 <jsp:useBean> 标准动作，自动将请求参数封装为 JavaBean 组件；还必须使用 java 脚本执行控制逻辑。

```
<%@page import="cn.javass.chapter1.javabean.UserBean"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%-- 创建javabean --%>
<jsp:useBean id="user" class="cn.javass.chapter1.javabean.UserBean"/>
<%--1、收集参数并封装参数（也直接使用jsp，在这块是简单的） --%>
<jsp:setProperty name="user" property="*"/>

<%
    String submitFlag = request.getParameter("submitFlag");
    if("login".equals(submitFlag)) {//登录
        //3调用javabean对象（业务方法）
        if(user.login()) {
            //4根据返回值选择下一个页面
            out.write("login success");
        } else {
            out.write("login fail");
        }
    } else {
%>

        <form action="" method="post">
            <input type="hidden" name="submitFlag" value="login"/>
            username:<input type="text" name="username"/><br/>
            password:<input type="password" name="password"/><br/>
            <input type="submit" value="login"/>
        </form>

<%      } %>

</body>
</html>
```

1、控制逻辑：根据请求参数选择要执行的功能方法

3、调用业务对象(javabean对象)进行登录：即模型，不仅包含数据还包含行为

2、表现代码：页面展示直接放在我们的servlet里边

收集参数和组织参数简单了许多

图 1-7

此处我们可以看出，使用 <jsp:useBean> 标准动作可以简化 javabean 的获取 / 创建，及将请求参数封装到 javabean ，再看一下 Model1 架构，如图 1-8 。

图 1-8 Model1　架构

Model1　架构中，JSP 负责控制逻辑、表现逻辑、业务对象（　javabean　）的调用，只是比纯　JSP 简化了获取请求参数和封装请求参数。同样是不好的，在项目中应该严禁使用（或最多再　demo　里使用）。

Model2　：在 JavaEE 世界里，它可以认为就是　Web MVC　模型

Model2　架构其实可以认为就是我们所说的　Web MVC　模型，只是控制器采用　Servlet　、模型采用　JavaBean　、视图采用　JSP，如图 1-9

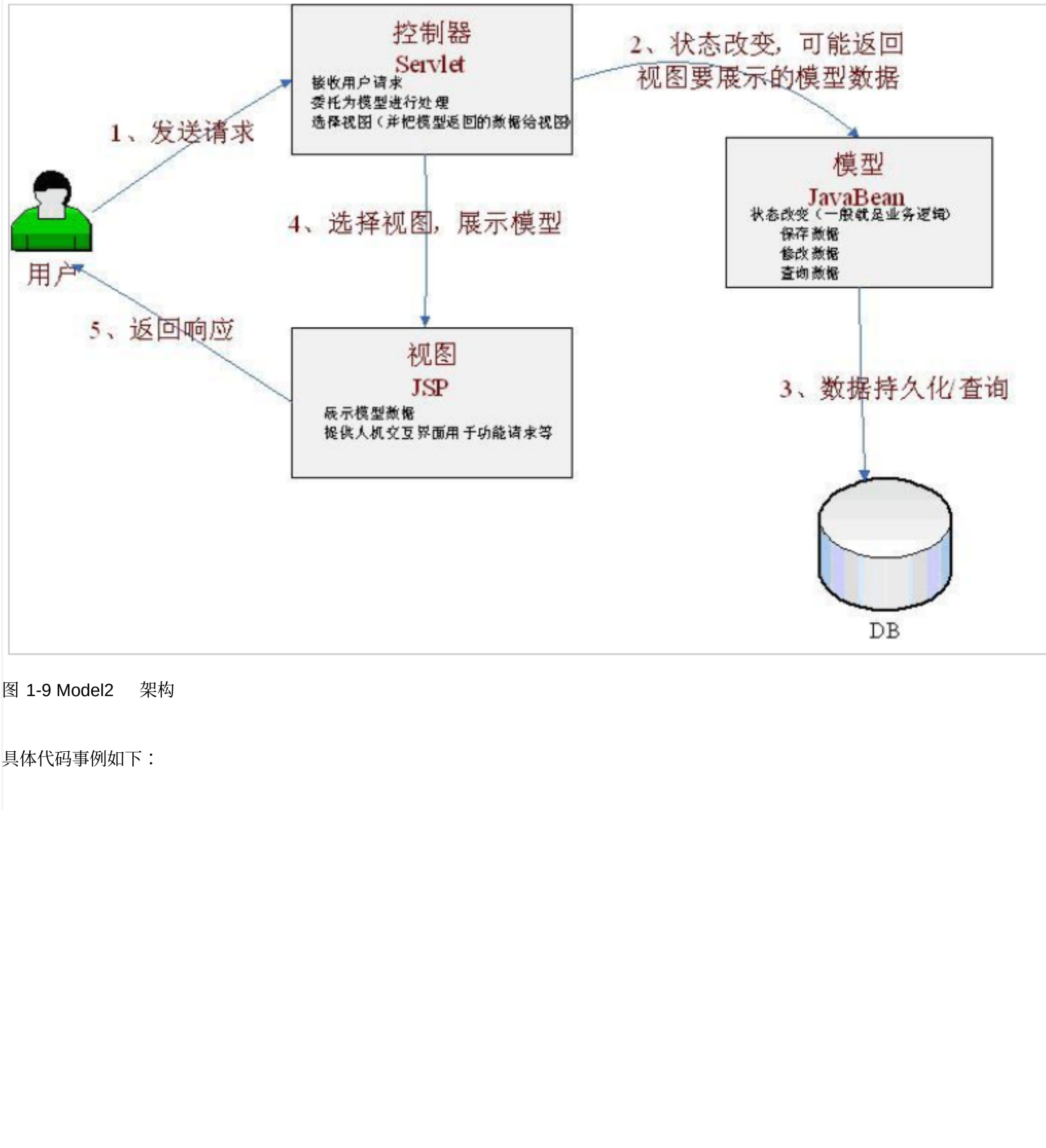


图 1-9 Model2　架构

具体代码事例如下：

模型

```
public class UserBean implements java.io.Serializable {

	private String username;
	private String password;
	public String getUsername() {
		return username;
	}
	public void setUsername(String username) {
		this.username = username;
	}
public String getPassword() {
		return password;
}
public void setPassword(String password) {
		this.password = password;
}

/**
 * 因为我们只关注表现层，此处只是模拟，实际项目需要改掉！
 * @param username 用户名
 * @param password 密码
 * @return
 */
public boolean login() {
		if("zhang".equals(this.username) && "123".equals(this.password)) {
			return true;
		}
		return false;
	}

}
```

模型: (业务对象 JavaBean对象)
包含设置/获取数据的方法
包含业务方法

视图

```
<%@page import="cn.javass.chapter1.javabean.UserBean"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>登录</title>
</head>
<body>

<form action="${pageContext.request.contextPath}/model2login" method="post">
    <input type="hidden" name="submitFlag" value="login"/>
    username:<input type="text" name="username" value="${user.username}"/><br/>
    password:<input type="password" name="password"/><br/>
    <input type="submit" value="login"/>
</form>

</body>
</html>
```

模型展示,可能包含一些展示逻辑

展示逻辑如在网站首页
如果用户已登陆,显示" 欢迎访问,sishuok"
如果用户未登陆,显示" 欢迎访问,游客"

控制器

```
public class Model2Servlet extends HttpServlet {
        @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        doPost(req, resp);//为了简单，直接委托给 doPost进行处理
    }
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        String submitFlag = req.getParameter("submitFlag");
        if("toLogin".equals(submitFlag)) {
            toLogin(req, resp);
            return;
        } else if("login".equals(submitFlag)) {
            login(req, resp);
            return;
        }
        toLogin(req, resp); //默认到登录页面
    }
    private void toLogin(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        //此处和JSP视图技术紧密耦合，更换其他视图技术几乎不可能
        req.getRequestDispatcher("/mvc/login.jsp").forward(req, resp);
    }
    private void login(HttpServletRequest req, HttpServletResponse resp)
        throws IOException, ServletException {
        //1收集参数
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        //2验证并封装参数（重复的步骤）
        UserBean user = new UserBean();
        user.setUsername(username);
        user.setPassword(password);
        //3调用javabean对象（业务方法）
        if(user.login()) {
            //4根据返回值选择下一个页面
            resp.sendRedirect(req.getContextPath() + "mvc/success.jsp");
        } else {
            //登陆失败再返回登陆页面  并显示上次输入的用户名
            //将视图要显示的模型数据放在请求里传递给视图，视图再来展示
            //此处也可以看出和Servlet API紧密耦合，更换视图技术也要重新设计请求数据
            req.setAttribute("user", user);
            toLogin(req, resp);
            return;
        }
    }
}
```

1、控制逻辑：根据请求参数选择要执行的功能方法

2、调用业务对象(javabean对象)进行登录：即模型，不仅包含数据还包含行为

3、选择下一个视图进行模型展示，模型数据直接放在request里

从 Model2 架构可以看出，视图和模型分离了，控制逻辑和展示逻辑分离了。

但我们也看到严重的缺点：

控制器：

1.控制逻辑可能比较复杂，其实我们可以按照规约，如请求参数 submitFlag=toAdd ，我们其实可以直接调用 toAdd 方法，

2.来简化控制逻辑；而且每个模块基本需要一个控制器，造成控制逻辑可能很复杂；

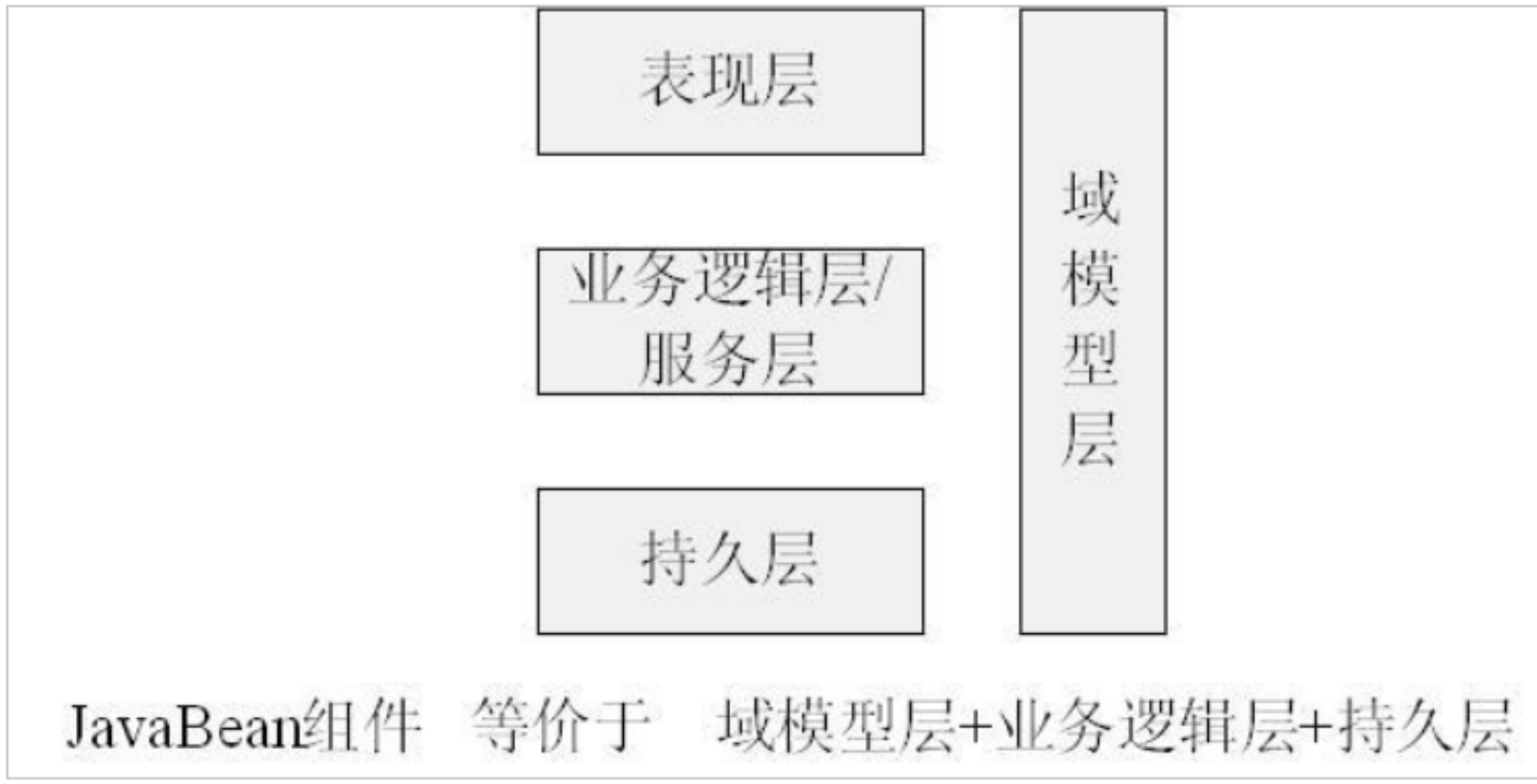
3.请求参数到模型的封装比较麻烦，如果能交给框架来做这件事情，我们可以从中得到解放；

4.选择下一个视图，严重依赖 Servlet API ，这样很难或基本不可能更换视图；

5.给视图传输要展示的模型数据，使用 Servlet API ，更换视图技术也要一起更换，很麻烦。

1 模型：

此处模型使用 JavaBean ，可能造成 JavaBean 组件类很庞大，一般现在项目都是采用三层架构，而不采用 JavaBean 。



视图

现在被绑定在 JSP，很难更换视图，比如 Velocity 、FreeMarker ；比如我要支持 Excel 、PDF 视图等等。

服务到工作者： Front Controller + Application Controller + Page Controller + Context

即，前端控制器 + 应用控制器 + 页面控制器（也有称其为动作） + 上下文，也是 Web MVC ，只是责任更加明确如图 1-10 ：

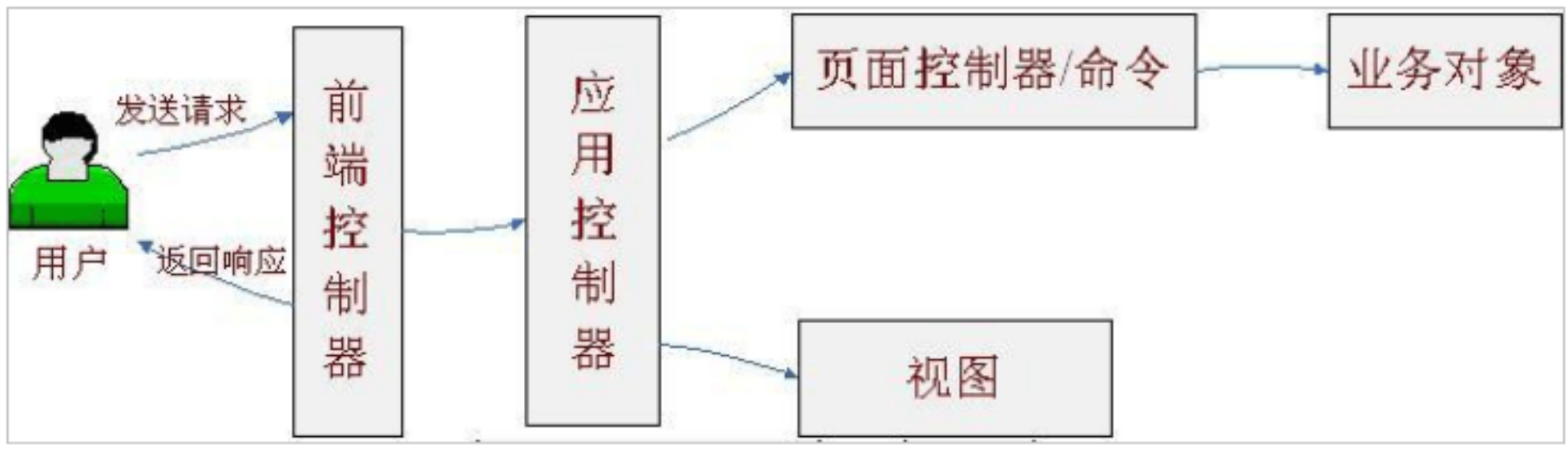
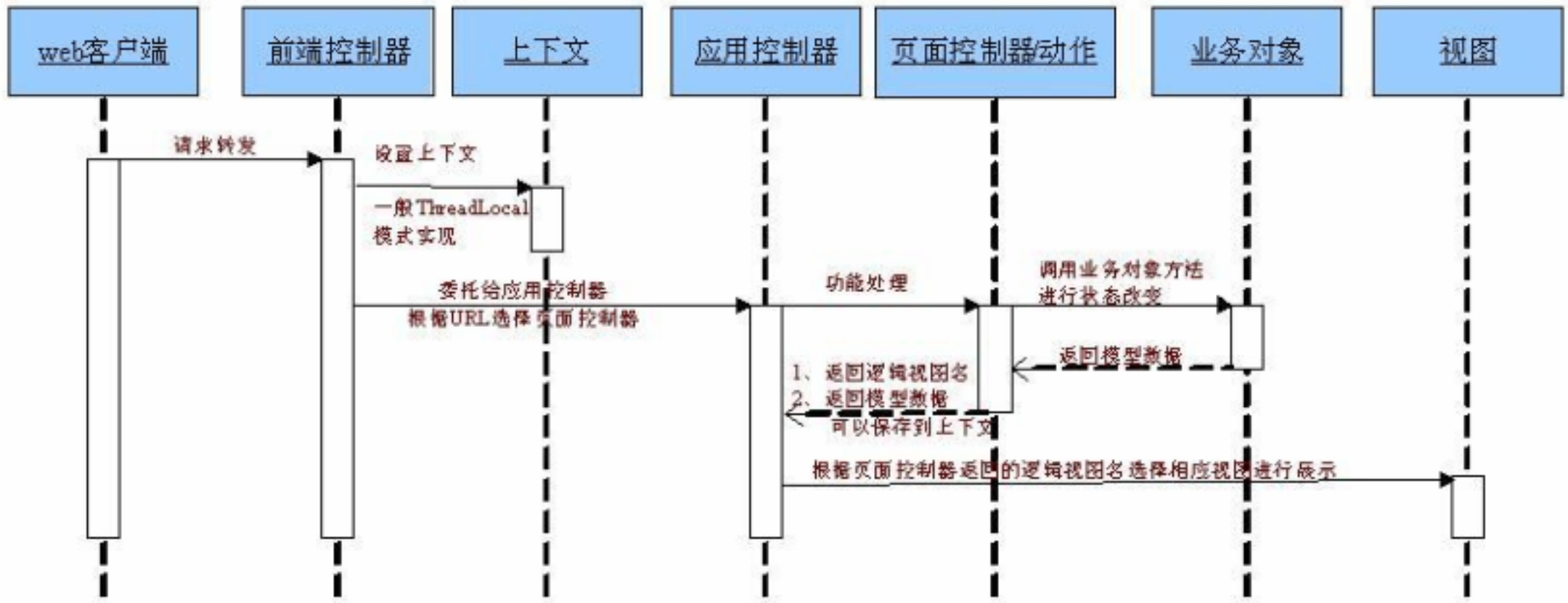


图 1-10

运行流程如下：



职责：

Front Controller ：前端控制器，负责为表现层提供统一访问点，从而避免 Model2 中出现的重复的控制逻辑（由前端控制器统一回调相应的功能方法， 如前边的根据 submitFlag=login 转调 login 方法）；并且可以为多个请求提供共用的逻辑 （如准备上下文等等），将选择具体视图和具体的功能处理（如 login 里边封装请求参数到模型，并调用业务逻辑对象）分离。

Application Controller ：应用控制器，前端控制器分离选择具体视图和具体的功能处理之后，需要有人来管理，应用控制器就是用来选择具体视图技术（视图的管理）和具体的功能处理（页面控制器 /命令对象 / 动作管理），一种策略设计模式的应用，可以很容易的切换视图 / 页面控制器，相互不产生影响。

Page Controller(Command) ：页面控制器 / 动作 / 处理器：功能处理代码，收集参数、封装参数到模型，转调业务对象处理模型，返回逻辑视图名交给前端控制器（和具体的视图技术解耦），由前端控制器委托给应用控制器选择具体的视图来展示，可以是命令设计模式的实现。页面控制器也被称为处理器或动作。

Context ：上下文，还记得 Model2 中为视图准备要展示的模型数据吗，我们直接放在 request 中（Servlet API 相关），有了上下文之后，我们就可以将相关数据放置在上下文，从而与协议无关（如 Servlet API ）的访问 / 设置模型数据，一般通过 ThreadLocal 模式实现。

到此，我们回顾了整个 web 开发架构的发展历程，可能不同的 web 层框架在细节处理方面不同，但的目的是一样的：

干净的 web 表现层：

模型和视图的分离；

控制器中的控制逻辑与功能处理分离（收集并封装参数到模型对象、业务对象调用）；

控制器中的视图选择与具体视图技术分离。

轻薄的 web 表现层：

做的事情越少越好，薄薄的，不应该包含无关代码；

只负责收集并组织参数到模型对象，启动业务对象的调用；

控制器只返回逻辑视图名并由相应的应用控制器来选择具体使用的视图策略；

尽量少使用框架特定 API，保证容易测试。

本文作者：张开涛