第 23 章　Web 应用的 MVC 设计模式 234
23.1　MVC 设计模式简介 234
23.2　JSP Model1 和 JSP Model2 236
23.3　Spring MVC 概述 238
23.3.1　Spring MVC 的框架结构 238
23.3.2　Spring MVC 的工作流程 240
23.4　创建采用 Spring MVC 的 Web 应用 241
23.4.1　建立 Spring MVC 的环境 241
23.4.2　创建视图 241
23.4.3　创建模型 243
23.4.4　创建 Controller 组件 244
23.4.5　创建 web.xml 文件和 HelloWeb-servlet.xml 文件 247
23.5　运行 helloapp 应用 248
23.6　小结 249
23.7　思考题 249

第 23 章　Web 应用的 MVC 设计模式

MVC 是 Model-View-Controller 的简称，即模型-视图-控制器。MVC 是 Xerox PARC 在八十年代为编程语言 Smalltalk－80 发明的一种软件设计模式，至今已被广泛使用。最近几年被推荐为 JavaEE 平台的主流设计模式，受到越来越多的 Web 开发者的欢迎。

本章首先介绍了 MVC 设计模式的结构和优点，接着介绍了在 Java Web 开发领域的两种设计模式：JSP Model1 和 JSP Molde2，然后介绍了 Spring MVC 框架实现 MVC 的机制。

Spring 是轻量级的开源 JavaEE 框架，而 Spring MVC 是 Spring 框架的一个扩展功能，为 JavaWeb 应用提供了现成的通用的 MVC 框架结构。Spring MVC 框架可以大大提高 Web 应用的开发速度。如果没有 Spring MVC，开发人员将不得不首先花大量的时间和精力设计和开发自己的框架。如果在 Web 应用中恰到好处地使用 Spring MVC，将把从头开始设计框架的时间节省下来，使得开发人员可以把精力集中在如何解决实际业务问题上。

而且，Spring MVC 本身是一群经验丰富的 Web 开发专家的集体智慧的结晶，它在全世界范围内得到广泛运用，并得到一致认可。因此，对于开发大型复杂的 Web 应用，Spring MVC 是不错的框架选择。

本章最后以 helloapp 应用为例，简要介绍了在 Web 应用中使用 Spring MVC 的方法。

23.1　MVC 设计模式简介

MVC 是一种设计模式，它强制性地把应用程序的数据展示、数据处理和流程控制分开。MVC 把应用程序分成三个核心模块：模型、视图和控制器，它们分别担当不同的任务。图 23-1 显示了这几个模块各自的功能以及它们的相互关系。

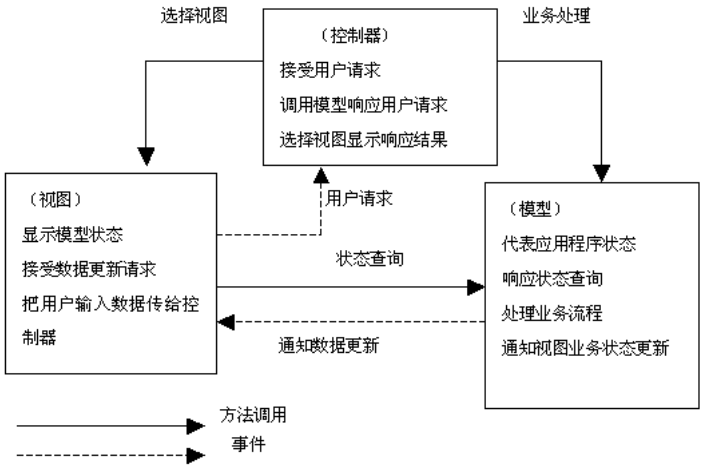

图 23-1　MVC 设计模式

视图

视图是用户看到并与之交互的界面。视图向用户显示相关的数据，并能接收用户的输入数据，但是它并不进行任何实际的业务处理。视图可以向模型查询业务状态，但不能改变模型。视图还能接受模型发出的数据更新事件，从而对用户界面进行同步更新。

对于基于请求/响应方式的 Web 应用，模型位于 Web 服务器端，视图位于用户浏览器端，目前无法做到模型向视图主动发出数据更新事件，使用户界面能自动刷新。

模型

模型是应用程序的主体部分。模型表示业务数据和业务逻辑。一个模型能为多个视图提供数据。由于同一个模型可以被多个视图重用，所以提高了模型的可重用性。

控制器

控制器负责应用的流程控制。所谓流程控制，这里是指接受用户的输入并调用相应的模型和视图去完成用户的需求。当 Web 用户单击 Web 页面中的提交按钮来发送 HTML 表单时，控制器接收请求并调用相应的模型组件去处理请求，然后调用相应的视图来显示模型返回的数据。

MVC 处理过程

现在总结一下 MVC 处理过程，首先控制器接收用户的请求，并决定应该调用哪个模型来进行处理；然后模型根据客户请求进行相应的业务逻辑处理，并返回数据；最后控制器调用相应的视图来格式化模型返回的数据，并通过视图呈现给用户。

MVC 的优点

在最初的 JSP 网页中，像数据库查询语句这样的数据库访问代码和像 HTML 这样的表示层代码混在一起。经验比较丰富的开发者会将数据库访问代码从表示层分离开来，但这通常不是很容易做到的，它需要精心的设计和不断地尝试。MVC 从根本上强制性地将它们

分开。尽管构造 MVC 应用程序需要一些额外的工作，但是它给开发人员带来的诸多优点是无庸质疑的。

首先，多个视图能共享一个模型。如今，同一个 Web 应用程序会提供多种用户界面，例如用户希望既能通过浏览器来收发电子邮件，还希望通过手机来访问电子邮箱，这就要求 Web 网站同时提供 Web 界面和 WAP 界面。在 MVC 设计模式中，模型响应客户请求并返回响应数据，视图负责格式化数据并把它们呈现给用户，业务逻辑和表示层分离，同一个模型可以被不同的视图重用，所以大大提高了代码的可重用性。

其次，模型是自包含的，与控制器和视图保持相对独立，所以可以方便地改变应用程序的业务数据和业务规则。如果把数据库从 MySQL 移植到 Oracle，或者把 RDBMS（Relational Database Management System，关系数据库管理系统）数据源改变成 LDAP（Lightweight Directory Access Protocol,轻量级目录访问协议）数据源，只需改变模型即可。一旦正确地实现了模型，不管数据来自数据库还是 LDAP 服务器，视图都会正确地显示它们。由于 MVC 的三个模块相互独立，改变其中一个不会影响其他两个，所以依据这种设计思想能构造良好的松耦合的构件。

此外，控制器提高了应用程序的灵活性和可配置性。控制器可以用来联接不同的模型和视图去完成用户的需求，控制器可以为构造应用程序提供强有力的组合手段。给定一些可重用的模型和视图，控制器可以根据用户的需求选择适当的模型进行处理，然后选择适当的视图将处理结果显示给用户。

MVC 的适用范围

使用 MVC 需要精心的设计，由于它的内部原理比较复杂，所以需要花费一些时间去理解它。将 MVC 运用到应用程序中，会带来额外的工作量，增加应用的复杂性，所以 MVC 不适合小型应用程序。

但对于开发存在大量用户界面，并且业务逻辑复杂的大型应用程序，MVC 将会使软件在健壮性和代码可重用性方面上一个新的台阶。尽管在最初构建 MVC 框架时会花费一定的工作量，但从长远角度看，它会大大提高后期软件开发的效率。

23.2 JSP Model1 和 JSP Model2

尽管 MVC 设计模式很早就出现了，但在 Web 应用的开发中引入 MVC 却是步履维艰。主要原因是在早期的 Web 应用的开发中，程序代码和 HTML 代码的分离一直难以实现。例如在 JSP 网页中执行业务逻辑的程序代码和 HTML 表示层代码混杂在一起，因而很难分离出单独的业务模型。产品设计弹性力度很小，很难满足用户的变化性需求。

在早期的 JavaWeb 应用中，JSP 文件负责处理业务逻辑、控制网页流程并创建 HTML 页面，参见图 23-2。JSP 文件是一个独立的、自主完成所有任务的模块，这给 Web 开发带来一系列问题：

- HTML 代码和 Java 程序代码强耦合在一起：JSP 文件的编写者必须既是网页设计者，又是 Java 开发者。但实际情况是，多数 Web 开发人员要么只精通网页设计，能够设计出漂亮的网页外观，但是编写的 Java 代码很糟糕；要么仅熟悉 Java 编程，

能够编写健壮的 Java 代码，但是设计的网页外观很难看。这两种才能皆备的开发人员并不多见。

- 内嵌的流程控制逻辑：要理解应用程序的整个流程，必须浏览所有 JSP 页面，试想一下拥有 100 个网页的网站的错综复杂的流程控制逻辑。
- 调试困难：除了很糟的外观之外，HTML 标记、Java 代码和 JavaScript 代码都集中在一个网页中，使调试变得相当困难。
- 可维护性差：更改业务逻辑或控制流程往往牵涉相关的多个 JSP 页面。
- 可读性差：设想有 1000 行代码的网页，其编码样式看起来杂乱无章。即使有彩色语法显示，阅读和理解这些代码仍然比较困难。

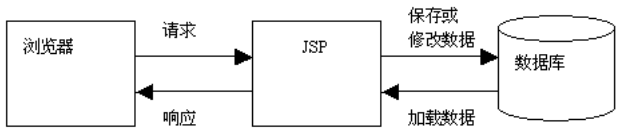

图 23-2　JSP 作为自主独立的模块

为了解决以上问题，在 Java Web 开发领域先后出现了两种设计模式，称为 JSP Model1 和 JSP Model2。虽然 Model1 在一定程度上实现了 MVC 中的视图和模型，但是它的运用并不理想；直到基于 JavaEE 的 JSP Model2 问世才得以改观。JSP Model 2 用 JSP 技术实现视图的功能，用 Servlet 技术实现控制器的功能，用 JavaBean 技术实现模型的功能。

JSP Model 1 和 JSP Model 2 的本质区别在于负责流程控制的组件不同。在 Model 1 中，如图 23-3 所示，JSP 页面负责调用模型组件来响应客户请求，并将处理结果返回用户。JSP 即要负责流程控制，还要负责产生用户界面，因此同时充当视图和控制器的功能，未能实现这两个模块之间的独立和分离。尽管 Model 1 十分适合简单应用的需要，但它不适合开发复杂的大型应用程序。不加选择地随意运用 Model 1，仍然会导致 JSP 页内嵌入大量的 Java 代码。尽管这对于 Java 程序员来说可能不是什么大问题，但如果 JSP 页面是由网页设计人员开发并维护的（通常这是开发大型项目的规范），这就确实是个问题了。从根本上讲，将导致角色定义不清和职责分配不明，给项目管理带来很多麻烦。

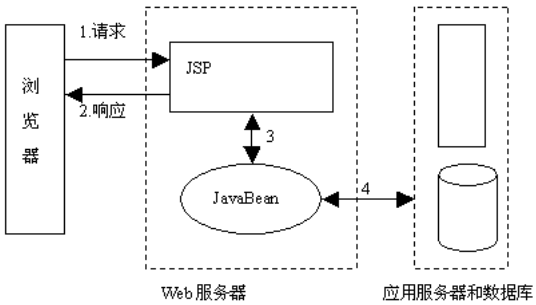

图 23-3　JSP Model1

本书介绍的 bookstore 应用采用了 JSP Model1 体系结构。JSP 负责生成视图和流程控制。模型层可以用 JavaBean、EJB 组件或者 Web 服务来实现。

JSP Model 2 体系结构，如图 23-4 所示，是一种联合使用 JSP 与 Servlet 来提供动态内容服务的方法。它吸取了 JSP 和 Servlet 两种技术各自的突出优点，用 JSP 生成表示层的内容，让 Servlet 完成深层次的处理任务。在这里，Servlet 充当控制器的角色，负责处理客户请求，创建 JSP 页面需要使用的 JavaBean 对象，根据客户请求选择合适的 JSP 页面返回给用户。在 JSP 页面内没有流程控制逻辑，它仅负责检索原先由 Servlet 创建的 JavaBean 对象，把 JavaBean 对象包含的数据作为动态内容插入到静态模板。这是一种有突破性的软件设计方法，它清晰地分离了数据展示、数据处理和流程控制，明确了角色定义以及软件开发者与网页设计者的分工。事实上，项目越复杂，使用 Model 2 设计模式的好处就越大。

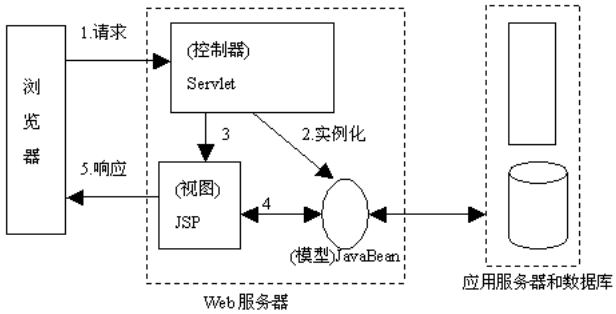

图 23-4　JSP Model2

23.3　Spring MVC 概述

当建筑师开始一个建筑项目时，首先要设计该建筑的框架结构，有了这份蓝图，接下来的实际建筑过程才会有条不紊，井然有序。同样，软件开发者开始一个软件项目时，首先也应该构思该软件应用的框架，规划软件模块，并定义这些模块之间的接口和关系。框架可以提高软件开发的速度和效率，并且使软件更便于维护。

对于开发 Web 应用，要从头设计并开发出一个可靠、稳定的框架并不是件容易的事。幸运的是，随着 Web 开发技术的日趋成熟，在 Web 开发领域出现了一些现成的优秀的框架，开发者可以直接使用它们，Spring MVC 就是一种不错的选择，它是基于 MVC 的 Web 应用框架。

23.3.1　Spring MVC 的框架结构

Spring MVC 是基于 JSP Model2 的一个 MVC 框架。在 Spring MVC 框架中，模型由实现业务逻辑的 JavaBean 或 EJB 组件构成，控制器由 Spring MVC 自带的 DispatcherServlet 类和由用户自定义的一系列 Controller 组件来实现，视图由一组 JSP 文件构成。图 23-5 显示了 Spring MVC 的框架结构。

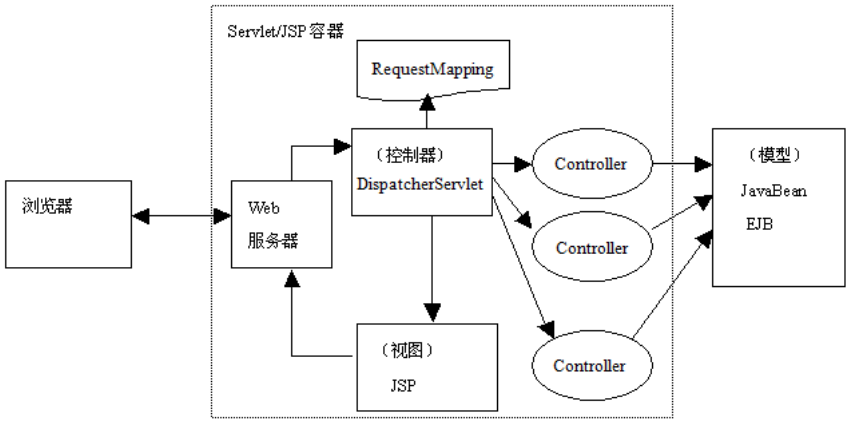

图 23-5 Spring MVC 的框架结构

视图

视图就是一组 JSP 文件，负责生成客户界面。在这些 JSP 文件中没有业务逻辑，也没有流程控制逻辑，只有 HTML 标记和标签，这些标签可以是标准的 JSP 标签或自定义 JSP 标签，如 Spring 标签库中的标签。

模型

模型表示应用程序的业务数据和业务逻辑。对于大型应用，业务逻辑通常由 JavaBean 或 EJB 组件实现。

控制器

控制器由 Spring MVC 提供的 DispatcherServlet 类和用户自定义的 Controller 组件来实现。org.springframework.web.servlet.DispatcherServlet 类是 Spring MVC 框架中的核心组件。DispatcherServlet 实现了javax.servlet.http.HttpServlet接口，它在 MVC 模型中扮演中央控制器的角色。DispatcherServlet 主要负责接收 HTTP 请求信息，根据 RequetMapping（请求映射，即用户请求的 URL 与实际的 Controller 组件的对应关系）信息，把请求转发给适当的 Controller 组件。如果该 Controller 组件还不存在，DispatcherServlet 会先创建这个 Controller 组件。

Controller 组件负责调用模型的方法，更新模型的状态，并帮助控制应用程序的流程。对于小型简单的应用，Controller 本身也可以完成一些实际的业务逻辑。

对于大型应用，Controller 充当客户请求和业务逻辑处理之间的适配器（Adaptor），其功能就是将数据展示与业务逻辑分离。Controller 根据客户请求调用相关的业务逻辑组件。业务逻辑由 JavaBean 或 EJB 来完成，Controller 侧重于控制应用程序的流程，而不是实现应用程序的业务逻辑。通过将业务逻辑放在单独的 JavaBean 或 EJB 组件 中，可以提高应用程序的灵活性和可重用性。

创建用户自定义的 Controller 组件非常简单，只要把一个类加上@Controller 标注，它就成为一个 Controller 组件。以下例程 23-1 的 SampleController 类就是一个简单的 Controller 组件：

例程 23-1　SampleController.java

```
@Controller
@RequestMapping("/hello")
public class SampleController {
   @RequestMapping(method = RequestMethod.GET)
   public String printHello(ModelMap model) {
      model.addAttribute("message", "Hello Spring MVC Framework!");
      return "result";
   }
}
```

以上 pirntHello()方法的返回值“result”表示一个 Web 组件的逻辑名字，接下来，DispatcherServlet 参考 Spring MVC 配置文件（参见本章 23.4.5 节），获取与逻辑名字“result”对应的目标组件的实际 URL，然后再把请求转发给该目标组件。

RequestMapping 信息

上面讲到客户请求是通过 DispatcherServlet 处理和转发的。那么，DispatcherServlet 如何决定把客户请求转发给哪个 Controller 组件呢？这就需要先设定客户请求的 URL 路径和 Controller 组件之间的映射关系。在以上例程 23-1 的 SampleController 类中，@RequestMapping 标注用于设定这种映射关系：

- 位于 SampleController 类前面的@RequestMapping("/hello")表明当客户端请求访问“/hello”URL 时，DispatherServlet 控制器就会调用这个 SampleController 组件。也就是说，为 SampleController 组件映射的 URL 为“/hello”。
- 位于 printHello()方法前面的@RequestMapping(method = RequestMethod.GET)表明，当客户端通过 HTTP GET 方式请求访问 SampleController 组件时，DispatherServlet 控制器就会调用该 pringHello()方法。

例程 23-1 的 SampleController 类也可以改写为：

```
@Controller
public class SampleController {
   @RequestMapping(value = "/hello", method = RequestMethod.GET)
   public String printHello(ModelMap model) {
      model.addAttribute("message", "Hello Spring MVC Framework!");
      return "result";
   }
}
```

以上@RequestMapping 标注的 value 属性设定访问 SampleController 组件的 printHello()方法的 URL 为“/hello”。

23.3.2　Spring MVC 的工作流程

对于采用 Spring MVC 框架的 Web 应用，在 Web 应用启动时就会加载并初始化 DispatcherServlet。当 DispatcherServlet 接收到一个要访问特定 Controller 组件的客户请求时，

将执行如下流程。

（1）检索和客户请求匹配的 Controller 组件，如果不存在，就先创建这个组件。然后调用 Controller 组件的相关方法。

（2）Controller 组件的相关方法调用模型层的有关组件来处理业务逻辑，再指定下一步负责处理请求的目标组件的逻辑名字。

（3）DispatcherServlet 参考 Spring MVC 配置文件（参见本章 23.4.5 节），获取与 Controller 组件指定的逻辑名字对应的目标组件的实际 URL，然后再把请求转发给该目标组件。如果该目标组件是 JSP 文件，那么该 JSP 文件会把包含响应结果的视图呈现给客户。

23.4 创建采用 Spring MVC 的 Web 应用

为了把 Spring MVC 运用到 Web 应用中，首先需要下载与操作系统对应的 Spring 软件包，下载地址为 https://repo.spring.io/libs-release-local/org/springframework/spring/，本书的技术支持网页（www.javathinker.net/javaweb.jsp）上也提供了 Spring 软件包的下载。

23.4.1 建立 Spring MVC 的环境

把 Spring 软件包 spring-framework-X.RELEASE-dist.zip 解压到本地，把其中 libs 目录下的 JAR 文件考拷贝到 Web 应用的 WEB-INF/lib 目录下。图 23-6 展示了基于 Sping MVC 的 helloapp 应用的目录结构。

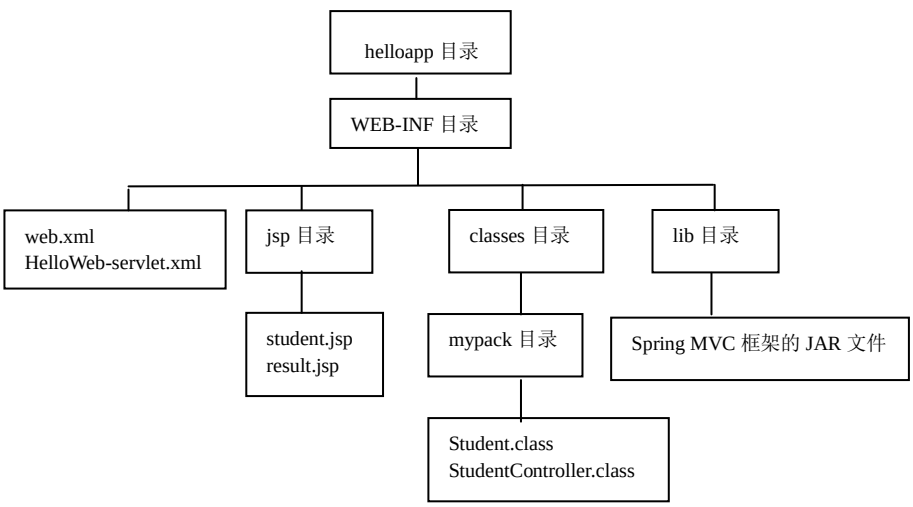

图 23-6 helloapp 应用的目录结构

23.4.2 创建视图

Spring MVC 的视图是一组包含了 Spring 标签的 JSP 文件。在本例中，视图层包括 student.jsp 和 result.jsp 两个文件。student.jsp 负责生成一个 HTML 表单，让客户端输入学生

信息。student.jsp 的 HTML 表单由 URL 为“/helloapp/addStudent”的 Web 组件来处理：

```
<form:form method = "POST" action = "/helloapp/addStudent">
 …
</form:form>
```

student.jsp 使用了 Spring 标签库中的标签。以下例程 23-2 是 student.jsp 的代码。

例程 23-2　student.jsp

```
<%@page contentType = "text/html;charset = UTF-8" language = "java" %>
<%@taglib uri = "http://www.springframework.org/tags/form"
                                        prefix = "form"%>
<html>
  <head>
    <title>Spring MVC Sample</title>
  </head>

  <body>
    <h2>Student Information</h2>
    <form:form method = "POST" action = "/helloapp/addStudent">
      <table>
        <tr>
          <td><form:label path = "name">Name</form:label></td>
          <td><form:input path = "name" /></td>
        </tr>
        <tr>
          <td><form:label path = "age">Age</form:label></td>
          <td><form:input path = "age" /></td>
        </tr>
        <tr>
          <td><form:label path = "id">ID</form:label></td>
          <td><form:input path = "id" /></td>
        </tr>
        <tr>
          <td colspan = "2">
            <input type = "submit" value = "Submit"/>
          </td>
        </tr>
      </table>
    </form:form>
  </body>
</html>
```

以上 student.jsp 代码中的<form:form>、<form:label>和<form:input>标签来自于 Spring 标签库，用来生成 HTML 表单。

result.jsp 负责显示客户端输入的学生信息，例程 23-3 是它的源代码。

例程 23-3　result.jsp

```
<%@page contentType = "text/html;charset = UTF-8" language = "java" %>
<%@page isELIgnored = "false" %>
<%@taglib uri = "http://www.springframework.org/tags/form"
                                              prefix = "form"%>
<html>
  <head>
   <title>Spring MVC Sample</title>
  </head>

  <body>
    <h2>Submitted Student Information</h2>
    <table>
      <tr>
        <td>Name:</td>
        <td>${name}</td>
      </tr>
      <tr>
        <td>Age:</td>
        <td>${age}</td>
      </tr>
      <tr>
        <td>ID:</td>
        <td>${id}</td>
      </tr>
    </table>
  </body>
</html>
```

23.4.3　创建模型

在 Spring MVC 的模型层，可以创建表示业务数据或实现业务逻辑的 JavaBean 组件。以下例程 23-4 的 Student 类是一个 JavaBean，它表示本范例应用的业务数据。

例程 23-4　Student.java

```
package mypack;
public class Student {
  private Integer age;
  private String name;
  private Integer id;

  public void setAge(Integer age) {
    this.age = age;
  }
  public Integer getAge() {
    return age;
```

```
  }
  public void setName(String name) {
    this.name = name;
  }
  public String getName() {
    return name;
  }
  public void setId(Integer id) {
    this.id = id;
  }
  public Integer getId() {
    return id;
  }
}
```

对于非常简单的 JavaWeb 应用，业务逻辑也可以直接由控制器层的 Controller 来完成。在本例中，业务逻辑将直接由 StudentController 来完成。

23.4.4 创建 Controller 组件

下面创建一个类名叫 StudentController 的 Controller 组件，参见例程 23-5。StudentController 类有两个方法：

- student()方法：对应的 URL 为“/student”，请求方式为 HTTP GET 方式。
- addStudent()方法：对应的 URL 为“/addStudent”，请求方式为 HTTP POST 方式。

例程 23-5　StudentController.java

```
package mypack;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.ui.ModelMap;

@Controller
public class StudentController {

  @RequestMapping(value ="/student", method =RequestMethod.GET)
  public ModelAndView student() {
    return new ModelAndView("student", "command", new Student());
  }

  @RequestMapping(value ="/addStudent", method =RequestMethod.POST)
  public String addStudent(
      @ModelAttribute("SpringWeb")Student student,ModelMap model){
```

```
        model.addAttribute("name", student.getName());
        model.addAttribute("age", student.getAge());
        model.addAttribute("id", student.getId());

        return "result";
    }
}
```

当客户端以 HTTP GET 方式请求访问 http://localhost:8080/helloapp/student，Spring MVC 的 DispatcherServlet 就会把请求转发給 StudentController 的 student()方法，这个方法返回一个 ModelAndView 对象，它表示把模型数据和视图绑定在一起的对象。在本例中，“new ModelAndView("student", "command", new Student())”中的三个参数的含义如下：

- 第一个参数“student”表示视图组件的逻辑名字为“student”，实际上对应 WEB-INF/jsp/student.jsp 文件。本章 23.4.5 节会介绍如何在 Spring MVC 配置文件中配置这种对应关系。
- 第二个参数“command”表明逻辑名为“student”的视图组件中的 HTML 表单需要与第三个参数指定的 Student 对象绑定。
- 第三个参数“new Student()”提供了一个新建的 Student 对象。Spring MVC 框架会负责把客户端在 HTML 表单中输入的数据填充到这个 Student 对象中。

DispatcherServlet 接收到 StudentController 的 student()方法返回的 ModelAndView 对象后，会把请求再转发給逻辑名字为“student”的视图组件，即 WEB-INF/jsp/student.jsp 文件。

以下图 23-7 显示了 Spring MVC 框架响应“/student”URL 的流程。

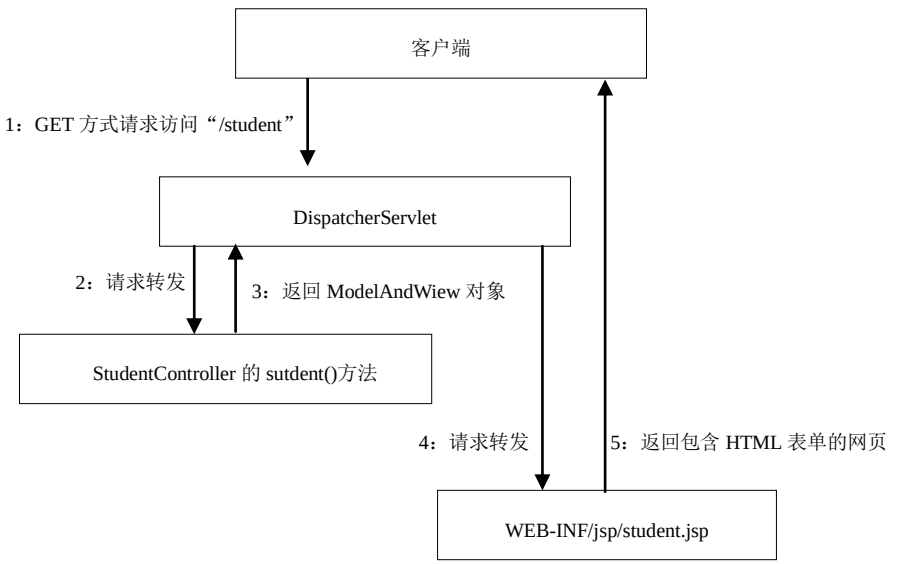

图 23-7　Spring MVC 框架响应“/student”URL 的流程

student.jsp 生成的网页如图 23-8 所示。

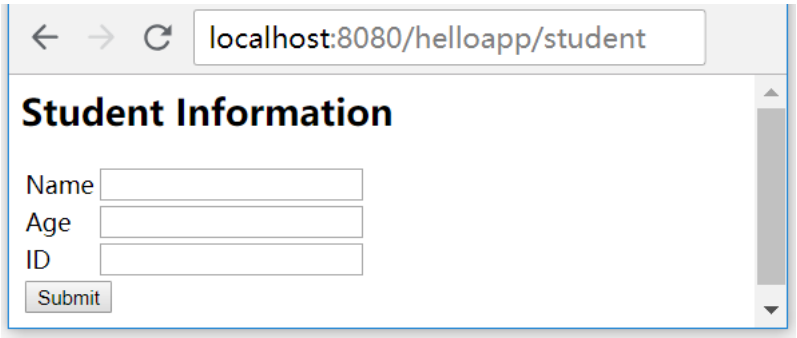

图 23-8　student.jsp 生成的网页

客户在图 23-8 所示的 HTML 表单中输入学生的相关信息，然后提交表单，这时浏览器会以 POST 方式请求访问“/helloapp/addStudent”URL。

Spring MVC 框架的 DispatcherServlet 接受到客户端的请求后，先把包含学生信息的 HTML 表单数据填充到表示模型数据的 Student 对象中，接下来 DispatcherServlet 就把请求转发給 StudentController 的 addStudent()方法。

StudentController 的 addStudent()方法读取 Student 对象的各个属性，再把它存放到一个 ModelMap 对象中：

```
        //model 变量为 ModelMap 类型
        model.addAttribute("name", student.getName());
        model.addAttribute("age", student.getAge());
        model.addAttribute("id", student.getId());
```

StudentController 的 addStudent()方法接下来返回一个字符串“result”，它是一个 Web 组件的逻辑名字，实际上对应 WEB-INF/jsp/result.jsp 文件。DispatcherServlet 再把请求转发給 result.jsp 文件。result.jsp 文件中的${name}、${age}和${id}标记会显示由 StudentController 存放在 ModelMap 对象中的 name、age 和 id 属性的值。由此可见，控制层可以借助 ModelMap 对象向视图层传递数据。

以下图 23-9 是 result.jsp 返回的包含学生信息的网页。

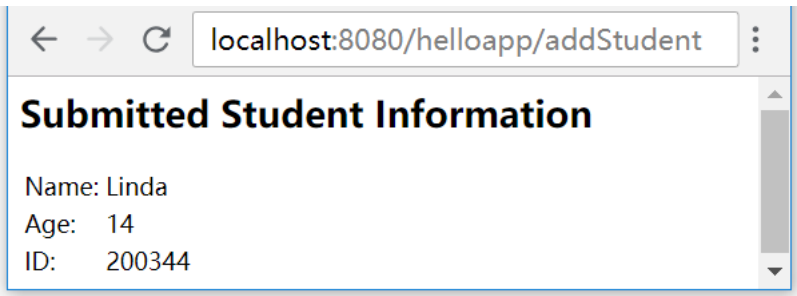

图 23-9　result.jsp 返回的包含学生信息的网页

以下图 23-10 显示了 Spring MVC 框架响应“/helloapp/addStudent”URL 的流程。

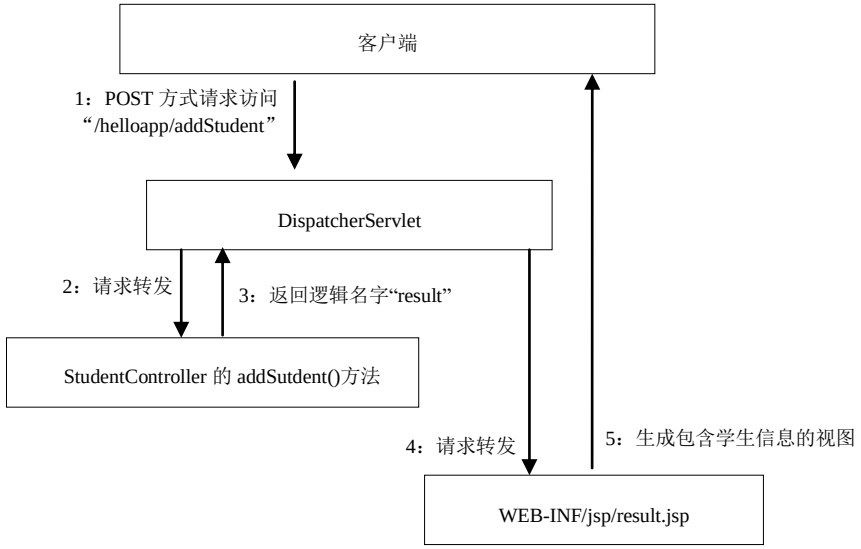

图 23-10　Spring MVC 框架响应“/helloapp/addStudent”URL 的流程

23.4.5　创建 web.xml 文件和 Spring MVC 配置文件

在 web.xml 文件中，应该对 Spring MVC 框架的中央控制枢纽 DispatcherServlet 进行配置：

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
     http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
  version="4.0" >

  <display-name>Spring MVC Sample</display-name>

  <servlet>
    <servlet-name>HelloWeb</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>HelloWeb</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

以上代码为 DispatcherServlet 映射的 URL 为“/”，这意味着所有访问 helloapp 应用的客户请求都会先由 DispatcherServlet 来预处理，然后再由 DispatcherServlet 转发给后续组件。

以上代码为 DispatcherServlet 设置的 Servlet 名字为“HelloWeb”，与此对应，必须为 Spring MVC 框架提供一个名为 HelloWeb-servlet.xml 配置文件，它也存放在 WEB-INF 目录下。例程 23-6 是 HelloWeb-servlet.xml 文件的代码。

例程 23-6　HelloWeb-servlet.xml

```
<beans xmlns = "http://www.springframework.org/schema/beans"
  xmlns:context = "http://www.springframework.org/schema/context"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
  http://www.springframework.org/schema/context
  http://www.springframework.org/schema/context
                                  /spring-context-3.0.xsd">

  <context:component-scan base-package = "mypack" />

  <bean class =  "org.springframework.web.servlet.view
                    .InternalResourceViewResolver">

    <property name = "prefix" value = "/WEB-INF/jsp/" />
    <property name = "suffix" value = ".jsp" />
  </bean>

</beans>
```

以上代码指定负责解析视图组件的逻辑名字的类为“InternalResourceViewResolver”。它的 prefix 和 suffix 属性分别设定了视图文件的前缀与后缀。

例如，对于 StudentController 的 addStudent()方法返回的逻辑名字“result”，将被解析为“/WEB-INF/jsp/result.jsp”文件。

再例如，StudentController 的 student()方法返回一个 ModelAndView 对象，它包含的视图组件的逻辑名字为“student”，“student”将被解析为“/WEB-INF/jsp/student.jsp”文件。

23.5　运行 helloapp 应用

按以上步骤创建好 helloapp 应用后，就可以启动 Tomcat 服务器，运行 helloapp 应用。在本书配套源代码包的 sourcecode/chapter23/helloapp 目录下，提供了这个应用的所有源文件，可以直接将整个 helloapp 目录拷贝到<CATALINA_HOME>/webapps 目录下，就会发布这个应用。

通过浏览器访问 http://localhost:8080/helloapp/student ，就可以访问 helloapp 应用了。

23.6 小结

Spring MVC 把 MVC 设计模式运用到 Web 应用中，它由一组相互协作的类以及自定义 JSP 标签库组成。本章介绍了 Spring MVC 的框架体系和工作流程。Spring MVC 的最主要的组件包括：

- DispatcherServlet：它担当控制器角色，客户请求都通过 DispatcherServlet 来转发。
- Controller：它负责调用适当的 JavaBean 或 EJB 组件来完成业务逻辑，以及调用适当的 JSP 文件来展示响应结果。对于简单的 Web 应用，Controller 本身也可以完成业务逻辑。
- Spring MVC 配置文件：假定在 web.xml 文件中为 DispatcherServlet 设定的 Servlet 名字为“HelloWeb”，那么 Spring MVC 配置文件的名字为“HelloWeb-servlet.xml”，它的默认存放路径是 WEB-INF 目录。

本章通过 helloapp 应用介绍了在 Web 应用中使用 Spring MVC 的方法。在运行 helloapp 应用时，还详细介绍了 Spring MVC 的各个组件的工作流程，帮助读者进一步理解 Spring MVC 的工作原理。

23.7 思考题

1．在 Spring MVC 框架的视图中可包含哪些组件？（多选）
（a）JSP （b）Servlet （c）Controller 组件 （d）代表业务逻辑或业务数据的 JavaBean
（e）EJB 组件 （f）自定义 JSP 标签

2．在 Spring MVC 框架的控制器中可包含哪些组件？（多选）
（a）JSP （b）Controller 组件 （c）代表业务逻辑或业务数据的 JavaBean
（d）DispatcherServlet （e）自定义 JSP 标签

3．在 Spring MVC 框架的模型中可包含哪些组件？（多选）
（a）JSP （b）Controller 组件 （c）代表业务逻辑或业务数据的 JavaBean
（d）EJB 组件 （e）自定义 JSP 标签

4．一个 Web 应用中包含这样一段逻辑：

```
if(用户还未登录)
  把请求转发给 login.jsp 登录页面;
else
  把请求转发给 shoppingcart.jsp 购物车页面;
```

以上逻辑应该由 MVC 的哪个模块来实现？（单选）

（a）视图 （b）控制器 （c）模型

5．一个 Web 应用中包含这样一段逻辑：

```
if(在数据库中已经包含特定用户信息)
  throw new BusinessException("该用户已经存在");
else
  把用户信息保存到数据库中;
```

以上逻辑应该由 MVC 的哪个模块来实现？（单选）

（a）视图　（b）控制器　（c）模型

6．一个 Web 应用中包含这样一段逻辑：

```
if(如果购物车为空)
  用红色字体显示文本“购物车为空”;
else
  显示购物车中的所有内容;
```

以上逻辑应该由 MVC 的哪个模块来实现？（单选）

（a）视图　（b）控制器　（c）模型

7．在 Spring MVC 中，视图层的 JSP 文件主要借助哪个技术，从而把负责业务逻辑和流程控制的 Java 程序代码从 JSP 文件中分离出去？（单选）

（a）自定义 JSP 标签　（b）HTML 技术　（c）JSP 程序片段　（d）JavaScript

参考答案

1．a,f　2．b,d　3．c,d　4．b　5．c　6．a　7．a