Servlet和JSP的集成：模型-视图-控制器构架

议程

- 模型-视图-控制器(MVC)的优点
- 用RequestDispatcher实现MVC
- 从servlet向JSP页面转发请求
- 相对URL的处理
- 不同显示选项的选取
- 数据共享策略的对比
- 从JSP页面转发请求
- 用包含取代转发

不同JSP构造的应用

简单应用

复杂应用

- 脚本元素直接调用servlet代码
- 脚本元素间接调用servlet代码（通过实用工具类）
- bean
- servlet/JSP的组合(MVC)
- MVC连同JSP表达式语言
- 定制标签

为什么要组合使用servlet & JSP?

- 典型的做法：使用JSP来简化HTML内容的开发与维护
  - 对于简单的动态代码，使用由脚本元素调用servlet代码来完成。
  - 对于稍微复杂一些的应用，则可使用脚本元素调用定制的类来完成。
  - 对于比较复杂的应用，则使用bean和定制标签。
- 但，这些是不够的
  - 对于复杂的处理过程，从JSP开始做起会难以处理。
  - JSP除了能够带来将实际的代码隔离成单独的类、bean、和定制标签的便利以外，它所隐含的假定是单个页面给出单个基本视图。

处理单个请求的可能方式

- **仅servlet**
  - 输出为二进制类型，例如：一幅图像。
  - 没有输出，如搜索引擎例子中的转发与重定向。
  - 页面的格式/布局变化很大，例如：门户网站。
- **仅JSP**
  - 输出大部分为字符数据，如HTML。
  - 格式/布局大部分固定。
- **二者的组合**
  - 单个请求可能会得到外观相差较大的多种结果。
  - 数据处理过程复杂，但布局相对固定。
- **这些仅适用于单个请求的处理**
  - 在整个应用中，我们依旧需要使用servlet和JSP。

对MVC的误解

- **必须采用复杂的框架**
  - 框架有时很有用
    - Struts
    - JavaServer Faces (JSF)
  - 但并非必需!
    - 对于大多简单或者适度复杂的应用来说，使用内建的RequestDispatcher 就能够很好地实现MVC。
- **MVC影响整个系统的设计**
  - 我们可以用MVC来处理单个请求。
  - 可以将它认为是MVC方案，而非MVC框架。
    - 也被称为是模型2方案

用RequestDispatcher 实现MVC

1. 定义用以表示数据的bean
2. 使用一个servlet处理请求
   - servlet读取请求参数，检查数据的缺失或异常等。
3. 填充bean
   - 该servlet调用业务逻辑（与具体应用相关的代码）或数据访问代码得到最终的结果。得出的结果被放在第一步中定义的bean中。
4. 将bean存储在请求、会话或servlet的上下文中
   - 该servlet调用请求、会话或servlet上下文对象的setAttribute存储表达请求结果的bean的引用。

用RequestDispatcher 实现MVC (续)

5. 将请求转发到JSP页面
   - 该servlet确定哪个JSP页面适合于处理当前的情形，并使用RequestDispatcher的forward方法将控制转移到那个页面。
6. 从bean中提取数据
   - JSP页面使用jsp:useBean和与第4步匹配的位置访问之前存储的bean，然后使用jsp:getProperty输出bean的属性。
   - JSP页面并不创建或修改bean；它只是提取并显示由servlet创建的数据。

转发请求的例子

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
  throws ServletException, IOException {
  String operation = request.getParameter("operation");
  if (operation == null) {
    operation = "unknown";
  }
  String address;
  if (operation.equals("order")) {
    address = "/WEB-INF/Order.jsp";
  } else if (operation.equals("cancel")) {
    address = "/WEB-INF/Cancel.jsp";
  } else {
    address = "/WEB-INF/UnknownOperation.jsp";
  }
  RequestDispatcher dispatcher =
    request.getRequestDispatcher(address);
  dispatcher.forward(request, response);
}
```


jsp:useBean在MVC中的使用与在独立 JSP页面中有什么不同

- **JSP页面不应该创建对象**
  - 应该由servlet，而非JSP页面，创建所有的数据对象。因此，为了保证JSP页面不会创建对象，我们应该使用<jsp:useBean ... type="package.Class" />

    而不是

    <jsp:useBean ... class="package.Class" />
- **JSP页面也不应该修改已有的对象**
  - 因此，我们应该只使用jsp:getProperty，不使用jsp:setProperty。

提示：jsp:useBean的scope选项

- **request**
  - <jsp:useBean id="..." type="..." scope="request" />
- **session**
  - <jsp:useBean id="..." type="..." scope="session" />
- **application**
  - <jsp:useBean id="..." type="..." scope="application" />
- **page**
  - <jsp:useBean id="..." type="..." scope="page" />
    或者仅仅使用
    <jsp:useBean id="..." type="..." />
  - MVC (Model 2)构架不使用这个scope。

基于请求的数据共享

- **Servlet**

```
ValueObject value = new ValueObject(...);
request.setAttribute("key", value);
RequestDispatcher dispatcher =
  request.getRequestDispatcher
                              ("/WEB-INF/SomePage.jsp");
dispatcher.forward(request, response);
```

- **JSP**

```
<jsp:useBean id="key" type="somePackage.ValueObject"
             scope="request" />
<jsp:getProperty name="key" property="someProperty" />
```

基于会话的数据共享

- **Servlet**

```
ValueObject value = new ValueObject(...);
HttpSession session = request.getSession();
session.setAttribute("key", value);
RequestDispatcher dispatcher =
  request.getRequestDispatcher
                    ("/WEB-INF/SomePage.jsp");
dispatcher.forward(request, response);
```

- **JSP**

```
<jsp:useBean id="key" type="somePackage.ValueObject"
             scope="session" />
<jsp:getProperty name="key" property="someProperty" />
```


基于会话的数据共享：变种

- **使用response.sendRedirect取代RequestDispatcher.forward**
- **差别：使用sendRedirect时**
  - 用户可以看到JSP的URL（使用RequestDispatcher.forward时用户只能看到servlet的URL）
  - 客户程序要经过两次往返（而forward只需一次）
- **sendRedirect的优点**
  - 用户可以单独访问JSP页面
    - 用户能够保存JSP页面的地址
- **sendRedirect的缺点**
  - 由于用户可以在不首先经过servlet的情况下访问JSP页面，所以，JSP页面所需的数据有可能不存在。
    - 因此，JSP页面需要编写代码检查这种情况。

基于ServletContext的数据共享

- **Servlet**

```
synchronized(this) {
  ValueObject value = new ValueObject(...);
  getServletContext().setAttribute("key", value);
  RequestDispatcher dispatcher =
    request.getRequestDispatcher
                        ("/WEB-INF/SomePage.jsp");
  dispatcher.forward(request, response);
}
```

- **JSP**

```
<jsp:useBean id="key" type="somePackage.ValueObject"
             scope="application" />
<jsp:getProperty name="key" property="someProperty" />
```


JSP页面中的相对URL

- **问题：**
  - 使用请求分配器进行的转发对客户来说是透明的。初始的URL是浏览器惟一知道的URL。
- **为什么这会比较重要？**
  - 浏览器会如何处理类似下面的这些标签：

```
<IMG SRC="foo.gif" …>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
<A HREF="bar.jsp">…</A>
```

  - 答案：浏览器将会把它们看作是相对于servlet的URL
- **最简单的解决方案：**
  - 使用以斜杠开始的URL

MVC的应用：银行账户余额

- **bean**
  - BankCustomer
- **填写bean并把请求转发给恰当JSP页面的servlet**
  - 读取客户的ID，调用数据访问代码填充BankCustomer
  - 使用当前的余额确定相应的结果页面
- **由JSP页面显示结果**
  - 负的余额：警示页面
  - 正常余额：标准页面
  - 高余额：添加广告的页面
  - 未知的客户ID：错误页面

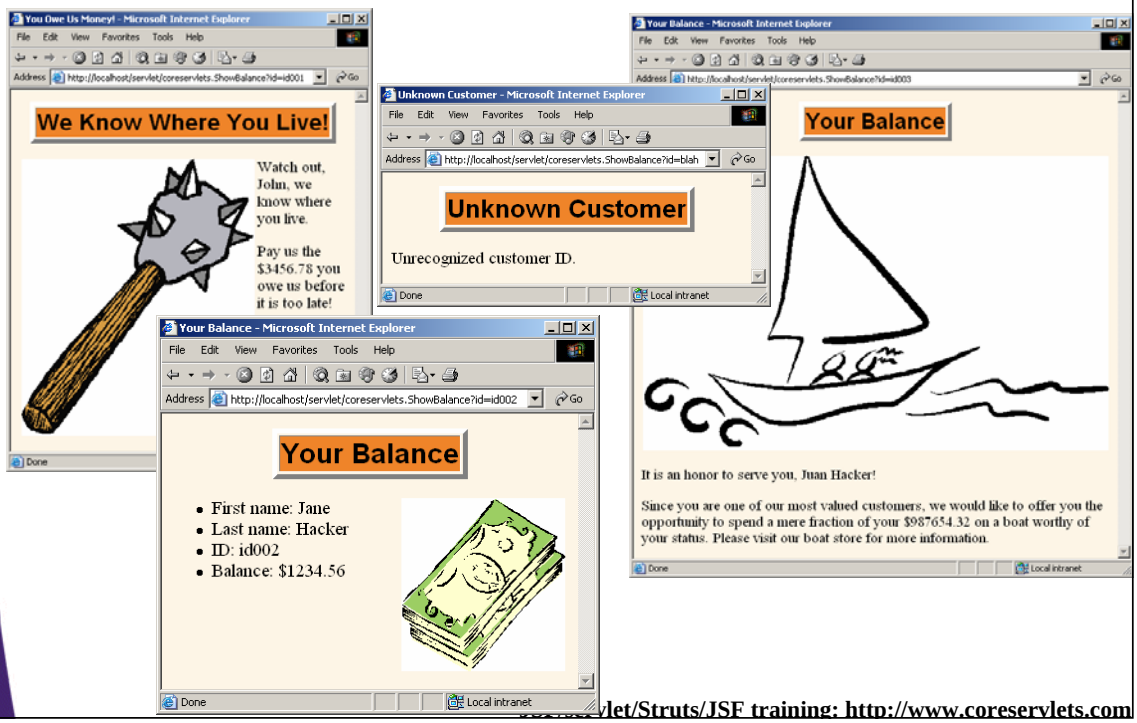
银行账户余额：servlet代码

```
public class ShowBalance extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    BankCustomer customer =
      BankCustomer.getCustomer
                        (request.getParameter("id"));
    String address;
    if (customer == null) {
      address =
        "/WEB-INF/bank-account/UnknownCustomer.jsp";
    } else if (customer.getBalance() < 0) {
      address =
        "/WEB-INF/bank-account/NegativeBalance.jsp";
      request.setAttribute("badCustomer", customer);
    }
    …
    RequestDispatcher dispatcher =
      request.getRequestDispatcher(address);
    dispatcher.forward(request, response);
```

银行账户余额：JSP代码(负余额)

```
…
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      We Know Where You Live!</TABLE>
<P>
<IMG SRC="/bank-support/Club.gif" ALIGN="LEFT">
<jsp:useBean id="badCustomer"
             type="coreservlets.BankCustomer"
             scope="request" />
Watch out,
<jsp:getProperty name="badCustomer"
                 property="firstName" />,
we know where you live.
<P>
Pay us the $<jsp:getProperty name="badCustomer"
                             property="balanceNoSign" />
you owe us before it is too late!
</BODY></HTML>
```


银行账户余额：结果



不同数据共享方式的对比：请求

- **目标**
  - 向用户显示一个随机的数字。
- **共享类型**
  - 由于每次请求应该产生新的数字，因而基于请求的共享是恰当的。

基于请求的共享：数据

```
package coreservlets;

public class NumberBean {
  private double num = 0;

  public NumberBean(double number) {
    setNumber(number);
  }

  public double getNumber() {
    return(num);
  }

  public void setNumber(double number) {
    num = number;
  }
}
```

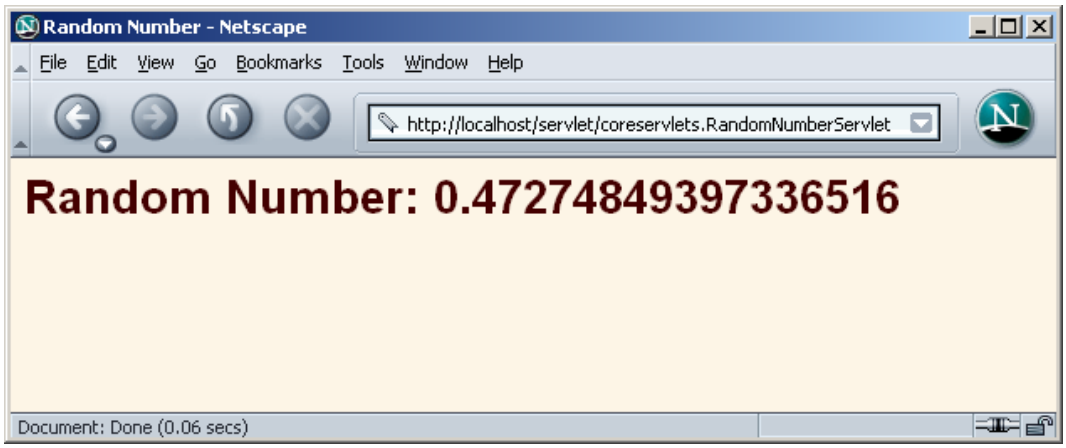
基于请求的共享：servlet

```
public class RandomNumberServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    NumberBean bean =
      new NumberBean(Math.random());
    request.setAttribute("randomNum", bean);
    String address =
      "/WEB-INF/mvc-sharing/RandomNum.jsp";
    RequestDispatcher dispatcher =
      request.getRequestDispatcher(address);
    dispatcher.forward(request, response);
  }
}
```


基于请求的共享： JSP

```
…
<BODY>
<jsp:useBean id="randomNum"
             type="coreservlets.NumberBean"
             scope="request" />
<H2>Random Number:
<jsp:getProperty name="randomNum"
                 property="number" />
</H2>
</BODY></HTML>
```

基于请求的共享：结果



不同数据共享方式的对比：会话

- **目标**
  - 显示用户的姓和名。
  - 如果用户没有给出姓名，我们希望使用他们之前给出的姓名信息。
  - 如果用户没有明确指定名称，并且没有找到之前的姓名，则显示一段警告。
- **共享类型**
  - 数据要为每个客户存储，因而基于会话的共享比较适用。

基于会话的共享：bean

```
package coreservlets;

public class NameBean {
  private String firstName = "Missing first name";
  private String lastName = "Missing last name";

  public NameBean() {}

  public NameBean(String firstName, String lastName) {
    setFirstName(firstName);
    setLastName(lastName);
  }

  public String getFirstName() {
    return(firstName);
  }

  …
}
```


基于会话的共享：servlet

```
public class RegistrationServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    HttpSession session = request.getSession();
    NameBean nameBean =
      (NameBean)session.getAttribute("nameBean");
    if (nameBean == null) {
      nameBean = new NameBean();
      session.setAttribute("nameBean", nameBean);
    }
```

基于会话的共享：servlet(续)

```
    String firstName =
      request.getParameter("firstName");
    if ((firstName != null) &&
        (!firstName.trim().equals(""))) {
      nameBean.setFirstName(firstName);
    }
    String lastName =
      request.getParameter("lastName");
    if ((lastName != null) &&
        (!lastName.trim().equals(""))) {
      nameBean.setLastName(lastName);
    }
    String address =
      "/WEB-INF/mvc-sharing/ShowName.jsp";
    RequestDispatcher dispatcher =
      request.getRequestDispatcher(address);
    dispatcher.forward(request, response);
  }
}
```


基于会话的共享：JSP

```
…
<BODY>
<H1>Thanks for Registering</H1>
<jsp:useBean id="nameBean"
             type="coreservlets.NameBean"
             scope="session" />
<H2>First Name:
<jsp:getProperty name="nameBean"
                 property="firstName" /></H2>
<H2>Last Name:
<jsp:getProperty name="nameBean"
                 property="lastName" /></H2>
</BODY></HTML>
```

基于会话的共享：结果

不同数据共享方式的对比：ServletContext

- 目标
  - 显示一个指定长度的质数。
  - 如果用户未给出期望有长度，我们希望使用我们为任何用户计算出的任意质数。
- 共享类型
  - 数据在多个客户间共享，因此，基于应用的共享比较恰当。

基于ServletContext的共享：bean

```
package coreservlets;
import java.math.BigInteger;

public class PrimeBean {
  private BigInteger prime;

  public PrimeBean(String lengthString) {
    int length = 150;
    try {
      length = Integer.parseInt(lengthString);
    } catch(NumberFormatException nfe) {}
    setPrime(Primes.nextPrime(Primes.random(length)));
  }

  public BigInteger getPrime() {
    return(prime);
  }
  …
}
```

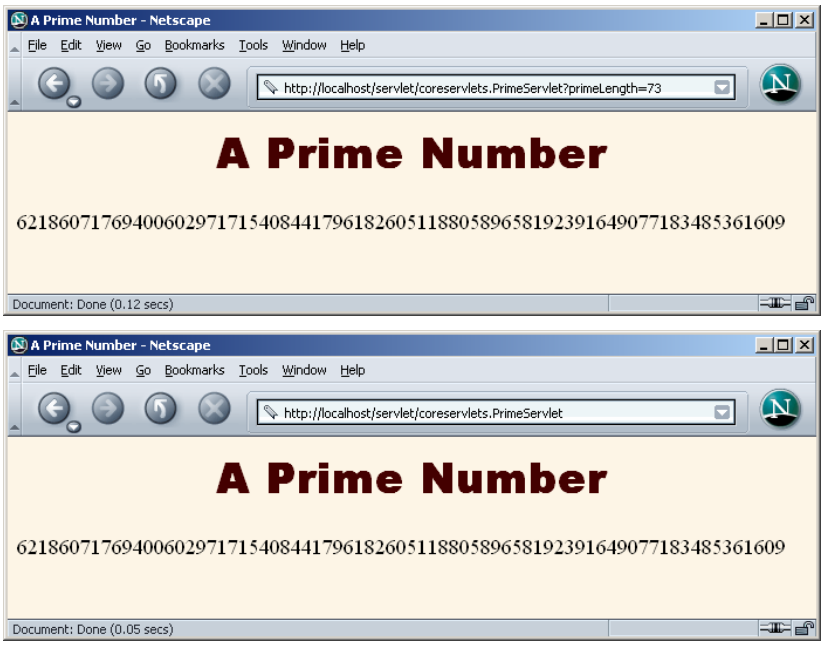

基于ServletContext的共享：servlet

```
public class PrimeServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    String length = request.getParameter("primeLength");
    ServletContext context = getServletContext();
    synchronized(this) {
      if ((context.getAttribute("primeBean") == null) ||
          (length != null)) {
        PrimeBean primeBean = new PrimeBean(length);
        context.setAttribute("primeBean", primeBean);
      }
      String address =
        "/WEB-INF/mvc-sharing/ShowPrime.jsp";
      RequestDispatcher dispatcher =
        request.getRequestDispatcher(address);
      dispatcher.forward(request, response);
    }
  }
}
```

基于ServletContext的共享： JSP

```
…
<BODY>
<H1>A Prime Number</H1>
<jsp:useBean id="primeBean"
             type="coreservlets.PrimeBean"
             scope="application" />
<jsp:getProperty name="primeBean"
                 property="prime" />
</BODY></HTML>
```


基于ServletContext的共享：结果



从JSP页面向别处转发

```
<% String destination;
   if (Math.random() > 0.5) {
     destination = "/examples/page1.jsp";
   } else {
     destination = "/examples/page2.jsp";
   }
%>
<jsp:forward page="<%= destination %>" />
```

- 合法，但不是个好主意
  - 业务和控制逻辑应属于servlet
  - JSP应该专注于表示

用包含取代转发

- 使用RequestDispatcher的`forward`方法：
  - 控制权永久地转移到新的页面
  - 初始的页面不能产生任何输出
- 使用RequestDispatcher的`include`方法：
  - 控制暂时地转移到新的页面
  - 初始页面可以在被包含页面之前或之后生成输出
  - 初始的servlet不会看到被包含页面的输出（有关这部分内容，参见后面有关servlet/JSP过滤器的主题）
  - 对于门户网站很有用：JSP用以表示不同的内容块，但针对不同的用户，这些内容以不同的次序进行组织。

用包含取代转发

```
response.setContentType("text/html");
String firstTable, secondTable, thirdTable;
if (someCondition) {
  firstTable = "/WEB-INF/Sports-Scores.jsp";
  secondTable = "/WEB-INF/Stock-Prices.jsp";
  thirdTable = "/WEB-INF/Weather.jsp";
} else if (...) { ... }
RequestDispatcher dispatcher =
  request.getRequestDispatcher("/WEB-INF/Header.jsp");
dispatcher.include(request, response);
dispatcher =
  request.getRequestDispatcher(firstTable);
dispatcher.include(request, response);
dispatcher =
  request.getRequestDispatcher(secondTable);
dispatcher.include(request, response);
dispatcher =
  request.getRequestDispatcher(thirdTable);
dispatcher.include(request, response);
dispatcher =
  request.getRequestDispatcher("/WEB-INF/Footer.jsp");
dispatcher.include(request, response);
```


Summary

- **MVC (Model 2) 方式适用于：**
  - 单次提交会产生多个基本外观。
  - 几个页面拥有大量公共的处理过程。
- **构架**
  - 由一个servlet应答初始的请求
  - Servlet完成实际的数据处理并将结果存储在bean中。
    - Bean存储在HttpServletRequest, HttpSession, 或ServletContext中
  - Servlet使用RequestDispatcher的forward方法将请求转发到JSP页面
  - JSP页面通过使用jsp:useBean和相应的作用域（request, session, 或application ）从bean中读出数据。

问题?

JSP, Servlet, & Struts Training Courses: http://courses.coreservlets.com
Available in US, China, Taiwan, HK, and Worldwide

JSP and Servlet Books from Sun Press: http://www.coreservlets.com
Available in English, Chinese (simplified and traditional script), and 12 other languages