TURING 图灵程序设计丛书 Java 系列

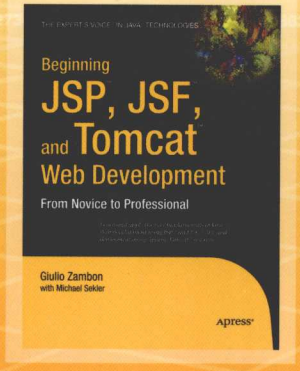
Apress®

Beginning JSP, JSF and Tomcat Web Development

JSP与JSF基础教程

[意] Giulio Zambon
[澳] Michael Sekler 著
石晓辉 苑永凯 译

- 从零开始学习Java Web开发
- 涵盖JSP、JSF、SQL、XML、HTML等网站开发核心
- 关键概念与实际案例紧密结合



人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书 Java系列

Beginning JSP, JSF and Tomcat Web Development

JSP与JSF基础教程

人民邮电出版社
北　京

图书在版编目（CIP）数据

JSP与JSF基础教程 /（意）赞博（Zambon, G.），（澳）塞克勒（Sekler, M.）著；石晓辉，苑永凯译．—北京：人民邮电出版社，2009.9
（图灵程序设计丛书）
书名原文：Beginning JSP，JSF and Tomcat Web Development
ISBN 978-7-115-21096-8

Ⅰ. J… Ⅱ. ①赞…②塞…③石…④苑… Ⅲ. JAVA语言－主页制作－程序设计－教材 Ⅳ. TP393.092

中国版本图书馆CIP数据核字（2009）第119513号

内 容 提 要

本书旨在教会你使用JSP（JavaServer Pages）和JSF（JavaServer Faces）进行Web开发。

书中结合网上书店实例，介绍Java和JSP、HTTP和HTML、SQL及数据库处理、JSF、XML和Tomcat技术中最基础、最重要的内容，并在附录中提供相关软件的下载安装说明和技术细节，以及Eclipse集成开发环境（IDE）和缩写词对照表。

本书适合有Java、HTML、SQL经验的初中级Web开发人员阅读，也是高校相关课程理想的教材。

图灵程序设计丛书

JSP与JSF基础教程

◆ 著 [意]Giulio Zambon [澳]Michael Sekler
译 石晓辉 苑永凯
责任编辑 傅志红
执行编辑 武 嘉

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京顺义振华印刷厂印刷

◆ 开本：800×1000 1/16
印张：22.5
字数：560千字 2009年9月第1版
印数：1－3 000册 2009年9月北京第1次印刷

著作权合同登记号 图字：01-2009-2906号
ISBN 978-7-115-21096-8/TP

定价：59.00元

读者服务热线：(010)51095186 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版权声明

译 者 序

在本书中文版付梓之际，作为译者，我不得不承认我在开始时低估了这本书的价值。

由于在初学编程时几本所谓的入门指南没有给我留下好的印象，所以一直以来我都是靠阅读官方参考文档、网络文章来学习技术的。诚然，这样可以很容易跟进最新的技术，但如果想把这些孤立的知识点顺利连接成面，却需要在实践中折腾许久。

那么现在你不需要再重复我的老路了。本书书名虽然普普通通，但却是一本极好的入门教程。它并没有罗列孤立的知识点去填充页数，也没有极尽所能地讲述技术的细枝末节。相反，它在用尽量多的章节向你展示Web开发所需的总体知识面。我想，作者和我一样都认为：高深繁琐的技术细节只会让初学者感到恐惧和枯燥，而总揽全局的知识面才是今后入门者进一步驾驭知识点的基石。

一本好书会激起无限共鸣和感慨。作为工作多年的开发人员，我对本书也充满了感激之情。因为它让我系统全面地回顾了Web开发的基础，织补了我一直有所缺失的知识体系；而且在为初学者讲解Web开发知识的同时，本书还时时提醒读者要卓越编程，即在达到预定功能的前提下，保持良好的编码习惯，使代码干净整洁可维护。我想这也是本书最大的亮点之一。

自然，作为一本Web开发的入门指南，本书的主要内容还是对相关技术的讲解——HTML、JSP、JSF、CSS、XML、Tomcat、Eclipse，等等。显然，本书中保留的都是这些技术最基础、最重要的内容，因为其中的每项技术都可以单独写出一部专业的图书。你可以把本书看做是一本Web开发参考目录，沿着这本书的条目就可以更细致深入地学习，探究其中提及的技术。而且，有了本书给予的基础技术、知识面和编程实践，相信你深入研究哪一项技术都不会感到吃力和迷茫。

翻译不是件轻松的事。除去教学、编码和处理日常事务外，我的大部分时间都投入到了对作者原意的斟酌和再次阐述上。能够将一本优秀的入门教程带给国内的朋友们，这些付出也是值得的。但由于受译者的专业水平和时间的双重限制，译稿中难免有错误和不妥之处，在此诚恳地希望读者批评指正。

在此我要谢谢合作者苑永凯，他的敬业精神和丰富的实践经验让我受益匪浅。还要感谢我的父亲，是他给了我全心全意的支持，在此说一句：爸爸，女儿爱你。

最后，感谢你花费时间来倾听我对这本书的看法。相信这本书也一定不会让你感到失望！

石晓辉

2009年春

前　言

欢迎阅读本书。本书旨在教你如何使用JSP（JavaServer Pages）2.1 和JSF（JavaServer Faces）1.2开发动态网页。

当然，这个目标说起来容易，做起来难，因为在使用JSP和JSF进行Web开发的过程中，你至少需要了解Java、HTML、XML和SQL相关的知识。此外，掌握JavaScript和CSS（层叠样式表）的一些知识也是有帮助的，而不懂得JSP表达式语言（EL）与XPath，就很难充分发挥JSP/JSF的威力。

为了涵盖以上所有知识面，同时避免书中内容的混乱，我们决定在本书正文中介绍关键的概念和组件，而将大部分的技术细节放在附录中。

第1章介绍JSP，描述JSP应用的一般结构，并逐字逐句地分析第一个JSP应用程序的代码。在章末，我们还会告诉你如何将这个应用部署到Tomcat中——我们知道你迫切渴望看到它运行起来。

第2章讲述了JSP的所有方面和组件。在简要介绍Java语法后，我们引入网上书店应用程序作为例子，这个例子将贯穿本书的多个章节。然后，我们会说明如何使用JSP变量（variable）、指令（directive）和标准动作（action），如何创建自定义动作，以及如何使用JSTL和EL。最后，将简单介绍如何使用XML语法编写JSP文档。

在满足了你尽快了解JSP的渴望后，我们在第3章后退一步，介绍HTML相关的内容。这一点很重要，因为HTML是执行JSP页面的结果，你必须熟悉它。这一章开始会先介绍HTTP请求–响应机制，之后介绍HTML组件（文本、对象、链接、表格和表单）、CSS，并给出一些JavaScript的例子。

第4章阐述如何从JSP访问数据库。因为撇开持久数据存储来谈论动态网页没有多大意义。

第5章讨论了JSF，介绍如何让它和JSP协作。

第6章专门讲述XML。读完本章你就会明白，为什么说开发Web应用程序而不使用XML是不可想象的。

第7章介绍Tomcat。

第8章是本书的最后一章，专门讲解网上书店应用程序。我们将前几章中提到的程序片段整理成型，并补全缺失的部分。

附录A介绍如何下载安装所需的程序包Java、JSP、Tomcat、SQL等。

附录B、C、D和E分别介绍HTML字符集、HTML、JSP和SQL。在这里，你可以看到前面章节中没有涉及的技术细节。

附录F是JSF的快速参考，附录G介绍Eclipse集成开发环境（IDE），而附录H是缩写词对照表。

读过本书之后，你就会得到一个开发高质量Web应用程序的工具箱。想出好点子，开发出使你成为百万富翁或者亿万富翁的下一个杀手级应用吧，一切尽在你的掌握中！

目　录

第1章 JSP和Tomcat简介

正是交互性使得Web变得真正有用起来。通过与一些远程服务器的交互，你可以找到所需的信息，办理银行业务或者实现网上购物。每次当你在Web表单中输入信息时，远程服务器上的应用程序会解析你的请求并生成一个Web页面作为响应，而JSP（JavaServer Pages）正是帮助你创建这种动态生成页面的技术。

Sun公司在1997年6月推出了Java servlet应用程序编程接口（API），目的是为动态Web页面的开发提供轻便高效的机制。简而言之，servlet通过包中定义的Java类来表示远程Web浏览器发送到服务器端的请求，以及服务器端到远程Web浏览器的响应。servlet就是放置在服务器上的Java对象，它能通过互联网接收请求，访问资源（例如数据库），执行逻辑以准备响应，最后将响应发送回网络。

Apache软件基金会（ASF）[①]开发的Apache Tomcat应用服务器，为servlet的执行提供了环境。Tomcat也能将JSP文档转换成servlet。

本章我们将向你介绍Java servlet和JSP，并展示它们如何在Tomcat中共同运行以生成动态Web页面。我们对JSP和Tomcat的介绍仅仅是点到为止，甚至都不会提到JSF。我们将教你如何使用基本工具开发应用程序，而不是在需要考虑极其枯燥细微的操作，并提供了复杂验证和调试能力的环境中进行开发。这会使你更好地了解现代开发工具能为你做些什么。

我们知道，你急于一头扎进有深度的技术中。所以，在简要地介绍完如何构建基于JSP的Web应用程序之后，我们会立刻向你展示一个复杂的例子，而不会事先进行面面俱到的技术讲解。

我们建议你首先参照附录A安装软件包。这样，你就不是只能浏览本书中印刷的代码，而是可以执行其中的例子，得到直观的感觉。

1.1 什么是JSP

正如我们所说的，JSP是支持你在Web页面中添加动态内容的技术。没有JSP，你总是得手动更新纯静态HTML页面的显示和内容。即使你仅想改变一个日期或者一个图片，也必须编辑HTML文件，键入修改的内容。没有人会为你去做这些事，而使用JSP，可以使页面内容基于许多因素而动态改变，包括时间、用户提供的信息、用户与你网站交互的历史，甚至用户的浏览器类型。

---

① Apache是一个非营利组织，其最早是从久负盛名的Apache HTTP Server发展起来的，为开源软件项目开发提供基础。Apache的成员通过投票产生，一般是对项目有贡献的活跃成员，提交到Apache的项目更是要经过严格的审核或是从内部项目中发展起来。——译者注

对于提供在线服务，这种能力是必不可少的——需要根据喜好和需求区别对待每一位用户。提供有意义的在线服务的一个关键方面，就是系统要能够记住与服务及其用户相关的数据。因此，数据库在动态Web页面中扮演着一个非常重要的角色。下面让我们循序渐进地学习这些技术。

历　史

Sun公司在1999年提出了JSP技术。开发人员很快就意识到添加更多的标签（tag）非常有用，于是JSP标准标签库（JSTL）诞生了。JSTL是自定义标签库的集合，自定义标签库封装了许多JSP标准应用程序的功能，从而消除重复使得应用程序更简洁。和JSTL一同出现的还有JSP表达式语言（EL）。

2003年，随着JSP 2.0的诞生，新的JSP规范中整合了EL，使其可用于自定义组件和模板文本，而不再像以往版本那样仅用于JSTL中。另外，JSP 2.0使我们能够创建自定义标签文件，从而极大增强了语言的可扩展性。

在JSP发展的同时，几种Web应用的开发框架出现了。2004年，其中一个框架JavaServer Faces（JSF）面世，它致力于构建用户界面（UI），并默认使用JSP作为其基本脚本语言。JSF对外提供API、JSP自定义标签库和表达式语言。

成立于1998年的Java执行委员会（Java Community Process，JCP[①]），在2006年5月发布了名为*JavaServer Pages 2.1*的245号Java规范请求（Java Specification Request，JSR），来有效地协调JSP和JSF技术。尤其是，JSP 2.1引入了统一表达式语言（UEL），将定义在JSP 2.0和JSF 1.2（由JSR 252定义）中的两个EL版本整合在一起。Sun公司将JSP 2.1引入到了Java平台企业版5（Java EE 5，在2006年5月作为JSR 244规范定稿）之中。包含在EE 5中的类依赖于来自Java平台标准版5（Java SE 5）的通用类。Java SE 5可以作为Java运行时环境（JRE）和Java开发工具包（JDK）。

同时，servlet技术也在发展，Sun公司在2005年9月发布了Servlet 2.5。2006年5月，JCP正式指定Servlet 2.5作为JSR 152的升级版本。

总之，Java EE 5包含JSP 2.1（JSP 2.1进而规定与JSF 1.2相一致的UEL），而Java SE 5则提供基础类，最后Servlet 2.5包括处理HTTP请求的类库。

1.1.1　访问Web页面

为了理解JSP，你首先得明白浏览器打开Web页面时都发生了些什么，不管是从浏览器的地址栏输入URL还是单击超链接。图1-1展示了它是如何运作的。

当你要求浏览器访问一个Web页面时，会发生如下几步操作。

(1) 当你在地址栏里输入地址http://www.website.com/path/whatever.html时，浏览器首先将www.website.com（即Web服务器的名称）解析成相应的互联网协议（Internet Protocol，IP）地址，这通常是要求互联网服务提供商（Internet Service Provider，ISP）提供的域名服务器来完成。然后，浏览器发送一个HTTP请求到新找到的IP地址，以接收/path/whatever.html标识的文件内容。

① JCP是由来自全世界的Java开发人员和获得许可的人员组成的开放性组织，负责对Java技术规范、参考实现（RI）和技术兼容性包（TCK）进行开发和修订。——译者注

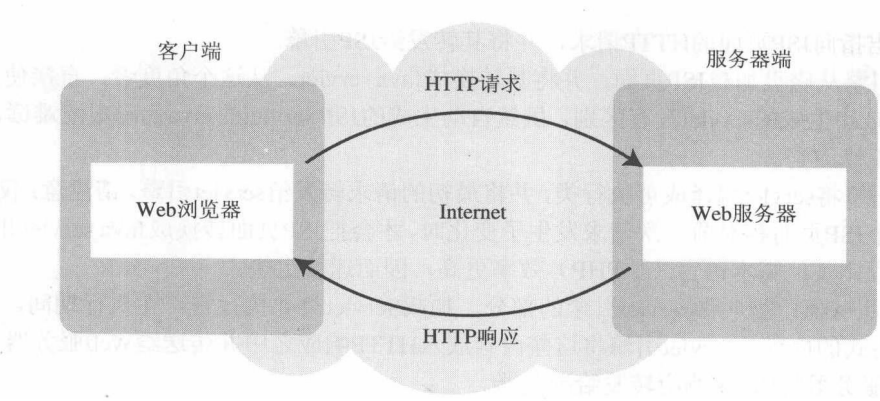


图1-1 访问简单的HTML页面

(2) 作为答复，Web服务器会发送一个包含纯文本HTML页面的HTTP响应。图像和其他非文本组成部分（如applet和声音）在页面中仅以引用的方式出现。

(3) 浏览器收到响应，解析页面中包含的HTML代码，接着向服务器端请求非文本组成部分，并显示这些内容。

1.1.2 访问 JSP 页面

对于JSP来说，Web页面并非真正的存在于服务器之上。正如你在图1-2中看到的，当对每一个请求作出响应时，服务器会创建一个新Web页面。

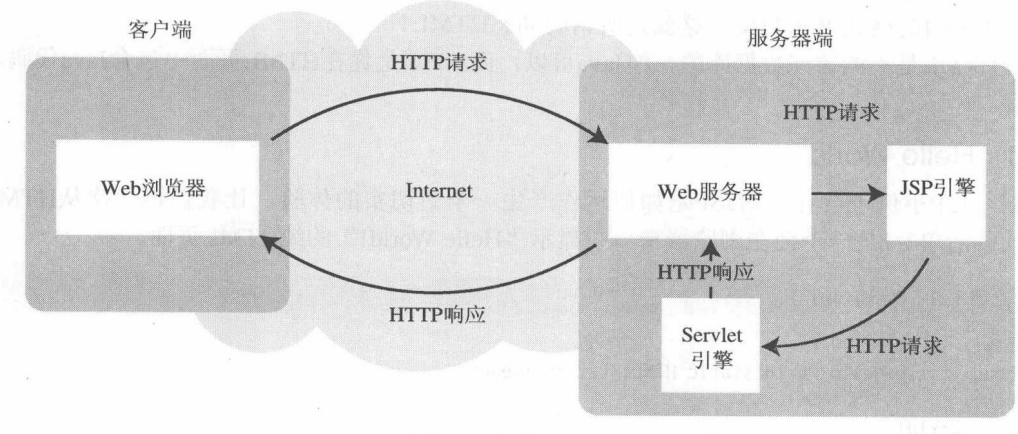


图1-2 访问JSP页面

下面的步骤揭示了Web服务器如何创建Web页面。

(1) 和普通页面一样，浏览器发送一个HTTP请求到Web服务器。有JSP这一步也不会变，虽然URL以.jsp而不是.html结尾。

(2) Web服务器并不是普通的服务器，而是带有识别、处理Java servlet扩展的Java服务器。Web

服务器识别出指向JSP页面的HTTP请求，并将其转发给JSP引擎。

(3) JSP引擎从磁盘加载JSP页面，并将其转换成Java servlet。从这个角度看，直接使用Java编写的servlet与JSP生成的servlet没有区别，虽然自动生成的JSP servlet的Java代码艰涩难读，而且你决不会去手工修改它。

(4) JSP引擎将servlet编译成可执行类，并将最初的请求转发给servlet引擎。请注意：仅仅当JSP引擎认为这个JSP页面相对前一次请求发生了变化时，才会把JSP页面转换成Java servlet并重编译。这使得此过程比其他脚本语言（如PHP）效率更高，因而速度更快。

(5) Web服务器中被叫做*servlet*引擎的部分，加载Servlet类并执行它。在执行期间，servlet会输出HTML格式的内容，servlet引擎将这些内容放入HTTP响应之中并传送给Web服务器。

(6) Web服务器将HTTP响应转发给浏览器。

(7) 浏览器采用与处理静态页面完全一样的方式，来处理HTTP响应里面动态生成的HTML页面。事实上，静态网页和动态Web页面格式是相同的。

你可能要问："对于JSP，如果仅当页面有更新时才会重新编译，那为什么你说，每次请求都会创建新的页面？"

传递到你浏览器上的是（由JSP页面转换、编译所得到的）servlet生成的输出，而不是JSP页面本身。受HTTP请求的参数和其他因素的影响，同样的servlet会产生不同的输出。比如，假设你在浏览网上商店供应的商品。当你单击某个商品的图片时，你的浏览器会生成一个以产品代码作为参数的HTTP请求。因此，servlet生成一个带有该商品描述的HTML页面。服务器并不需要为每个商品代码重新编译servlet。

servlet首先查询包含所有商品细节的数据库，然后得到你所感兴趣的产品的详细描述，最后以HTML页面的格式展示数据。这就是所谓的动态HTML！

纯HTML是不能访问数据库的，但Java可以，而JSP则是你在HTML页面中包含Java代码片段的途径。

1.1.3　Hello World

一个JSP小例子会让你对JSP是如何运作产生一个更切实的体验。让我们再一次从HTML开始。代码清单1-1是一个会在浏览器窗口中显示"Hello World!"的纯HTML页面。

代码清单1-1　hello.html

```
<html>
<head><title>Hello World static HTML</title></head>
<body>
Hello World!
</body>
</html>
```

创建下面所示文件夹以存放hello.html。

```
C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\hello\
```

在浏览器中输入以下URL来查看Web页面。

```
http://localhost:8080/hello/hello.html
```

通常情况下，为了使你的浏览器遵照万维网联盟（W3C）的XHTML标准检查页面语法，必须以下面3行内容来开始页面代码。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

同样你还需要将下面的：

```
<html>
```

替换为下列内容：

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
```

不过，对于这个简单的例子，我们希望保持代码简明扼要。

图1-3是页面在浏览器中的显示效果。

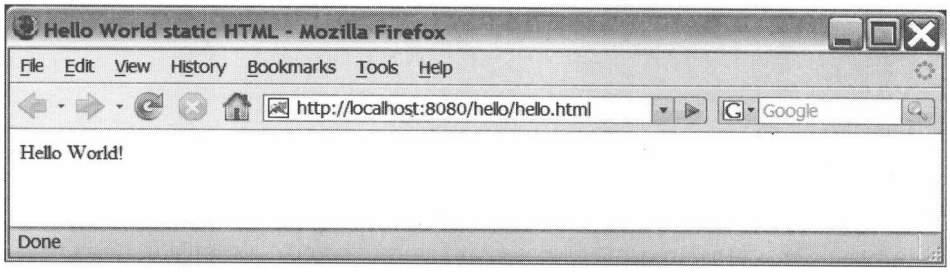


图1-3 纯HTML的“Hello World!”

如果通过浏览器查看页面源文件，毫无疑问，你会看到代码清单1-1中显示的内容。为了在JSP页面中获得完全相同的结果，你只需要像代码清单1-2中那样，在第一行之前插入一个JSP指令，并将文件扩展名从html改为jsp。

代码清单1-2 JSP页面的“Hello World!”

```
<%@page language="java" contentType="text/html"%>
<html>
<head><title>Hello World not-so-dynamic HTML</title></head>
<body>
Hello World!
</body>
</html>
```

说明 IE 7仅解析包含page contentType指令的JSP页面。

显然，在这么简单的页面中使用JSP没有多大用处。仅当页面中包含动态内容时才值得去使用JSP。浏览内容更多的代码清单1-3。

代码清单1-3　hello.jsp

```
<%@page language="java" contentType="text/html"%>
<html>
<head><title>Hello World dynamic HTML</title></head>
<body>
Hello World!
<%
  out.println("<br/>Your IP address is " + request.getRemoteAddr());

  String userAgent = request.getHeader("user-agent");
  String browser = "unknown";

  out.print("<br/>and your browser is ");
  if (userAgent != null) {
    if (userAgent.indexOf("MSIE") > -1) {
      browser = "MS Internet Explorer";
      }
    else if (userAgent.indexOf("Firefox") > -1) {
      browser = "Mozilla Firefox";
      }
    }
  out.println(browser);
  %>
</body>
</html>
```

和hello.html一样，将hello.jsp放到webapps\hello\文件夹中，你就可以访问这个页面了。

写在符号对<%...%>之间的代码是用Java编写的scriptlet。当Tomcat的JSP引擎解析此模块时，它会创建一个包含92行代码的Java Servlet，在这些代码中你可以找到如代码清单1-4所示的代码（缩进并删除了空行）。

代码清单1-4　从“Hello World!”JSP页面产生的Java代码

```
out.write("\r\n");
out.write("<html>\r\n");
out.write("<head><title>Hello World dynamic HTML</title></head>\r\n");
out.write("<body>\r\n");
out.write("Hello World!\r\n");
out.write('\r');
out.write('\n');
out.println("<br/>Your IP address is " + request.getRemoteAddr());
String userAgent = request.getHeader("user-agent");
String browser = "unknown";
out.print("<br/>and your browser is ");
if (userAgent != null) {
  if (userAgent.indexOf("MSIE") > -1) {
    browser = "MS Internet Explorer";
    }
```

```
    else if (userAgent.indexOf("Firefox") > -1) {
      browser = "Mozilla Firefox";
      }
    }
out.println(browser);
out.write("\r\n");
out.write("</body>\r\n");
out.write("</html>\r\n");
```

正如前面提到的，浏览器每次向服务器发送请求都会触发servlet的执行。然而，在代码清单1-4中的代码执行之前，变量out会被绑定到响应内容上。结果就是，所有写入到out变量中的内容，最终将会显示在浏览器中的HTML页面上。粗体显示的scriptlet会被复制到servlet中。其他的所有内容都被写入到输出中。现在我们应该清楚，在JSP页面中是如何混合HTML和Java编码的。

由于每个servlet中都定义了变量out，所以在任何JSP模块里，你都可以利用它将一些内容插入到响应中。其他类似的“全局”JSP变量还有request（HttpServletRequest类型）。请求中包含着请求发起地的IP地址——也就是浏览器所在的远程计算机（记住，代码运行在服务器上）。想要从请求信息中提取这个地址，你只需要执行其方法getRemoteAddr()。请求中还会包含浏览器的相关信息，但当某些浏览器发送请求时，它们会提供一些误导性的信息，而且格式复杂。然而，代码清单1-4会告诉你如何判断浏览器到底是IE还是Mozilla Firefox。图1-4是生成的页面在浏览器中的效果。

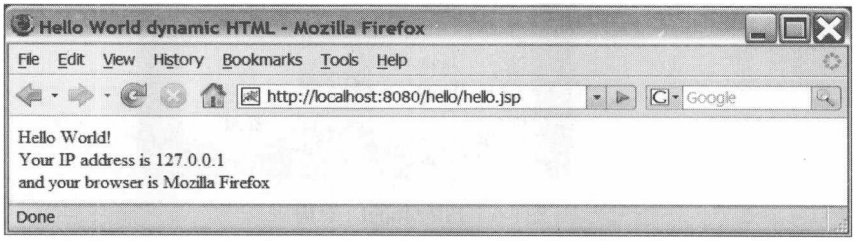


图1-4 “Hello World!” JSP

需要注意，IP地址127.0.0.1与主机localhost是一样的。万一你想确认下这个HTML确实是动态的，请参见图1-5。顺便声明一下，在hello.jsp中基于用户代理识别IE的方法是由Microsoft官方提供的。

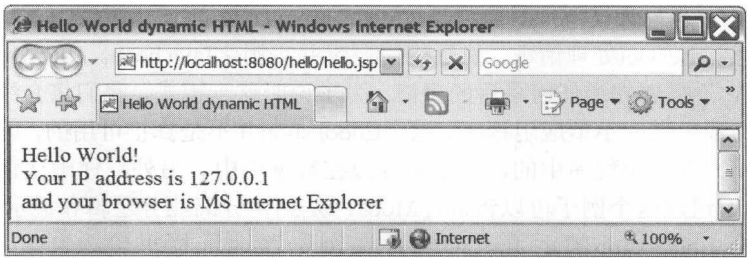


图1-5 IE中的“Hello World!” JSP

1.2　JSP应用架构

在HTML模块里插入Java代码提供了创建动态Web页面的可能，但可能并不意味着你可以做到经济有效。如果你使用包含在<%...%>符号对中的scriptlet开始复杂应用程序的开发，那么很快就会陷入到代码难以维护的地步。Java和HTML混合编码的关键问题——以“Hello World!”为例——是应用逻辑和信息在浏览器中的显示混杂在了一起。一般来说，业务应用设计师和Web页面设计师是不同的人，他们之间的技能互补而且很少重叠。应用设计师谙熟复杂算法和数据库之道，而Web设计师擅长页面布局和制图。基于JSP的应用程序架构应该注重这种区别。你最不想看到的就是开发团队中角色不清晰，最终每个人都在做其他人更能胜任的工作。

1.2.1　Model 1架构

对于这个问题，开发人员找到的第一种解决方案就是定义JSP Model 1架构，Model 1中的应用逻辑由Java类（像Java bean）实现，然后你就可以在JSP中使用这些Java类了（见图1-6）。

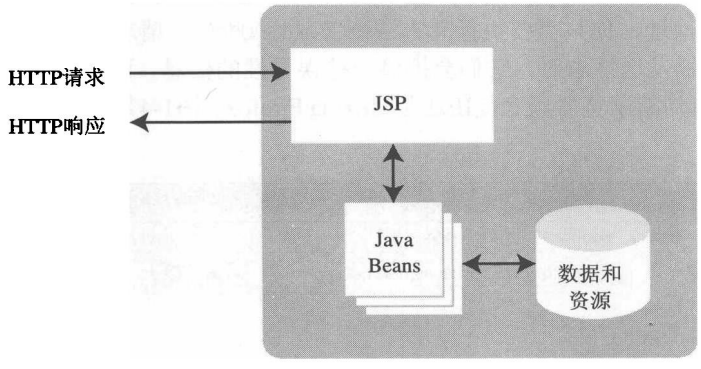


图1-6　JSP Model 1架构

几千行代码量的应用程序，Model 1是可以接受的，对于程序员来说更是小菜一碟，但是JSP页面另外还需要处理HTTP请求，这就会让页面设计师感到头疼了。

1.2.2　Model 2架构

一个更好的解决方案就是分离应用逻辑和页面显示，这也适用于大型的应用。这个解决方案就是JSP Model 2架构，也称为模型视图控制器（model-view-controller，MVC）设计模式（见图1-7）。

使用这种模式，servlet处理请求，执行应用逻辑并实例化Java bean。JSP从bean中获取数据并设置响应格式，它不需要知道任何幕后的事情。为了说明这一模式，我们将介绍一个简单的例子Ebookshop——在线售书的小型应用程序。这个Ebookshop并不是真正可用的，因为图书列表是硬编码（hard-coded）在应用程序中的，而不是存放在数据库中。另外，你确认订单也不会有任何反应。不过，我们通过这个例子可以告诉你Model 2是如何分离业务逻辑和显示的。

图1-8显示了Ebookshop的主页，即当你在浏览器地址栏里输入http://localhost:8080/ebookshop后所看到的页面。