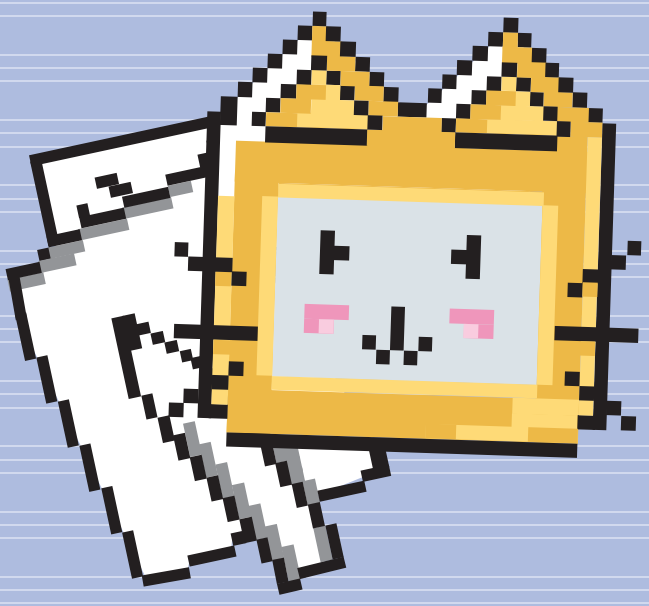
# JSP & Servlet 学习笔记（第3版）

## ——从Servlet到Spring Boot

林信良　著



清华大学出版社
北　京

## 内 容 简 介

本书是作者多年来教学实践经验的总结，汇集了学员在学习 JSP & Servlet 或认证考试时遇到的概念、操作、应用等各种问题及解决方案。

本书基于 Servlet 4.0/Java SE 8 重新改版，无论章节架构还是范例程序代码，都做了全面更新。书中详细介绍了 OWASP TOP10、CWE、CVE，讨论了会话安全、密码管理、Java EE 安全机制、CSRF 等 Web 安全基本概念，增加了对 Spring、Spring MVC、Spring Boot 的入门介绍，认识 Web MVC 框架与快速开发工具的使用。本书还涵盖了文本处理、图片验证、自动登录、验证过滤器、压缩处理、线上文件管理、邮件发送等实用范例。

本书在讲解过程中，以“微博”项目贯穿全书，将 JSP & Servlet 技术应用于实际项目开发之中，并使用重构方式来改进应用程序架构。

本书适合 JSP & Servlet 初学者以及广大 JSP & Servlet 技术应用人员。

北京市版权局著作权合同登记号　图字：01-2018-3785



图书在版编目(CIP)数据

JSP & Servlet 学习笔记：从 Servlet 到 Spring Boot：第 3 版 / 林信良 著. —北京：清华大学出版社，2019

ISBN 978-7-302-52245-4

Ⅰ. ①J…　Ⅱ. ①林…　Ⅲ. ①JAVA 语言－程序设计　Ⅳ. ①TP312.8

中国版本图书馆 CIP 数据核字(2019)第 020117 号

责任编辑：王　定
封面设计：孔祥峰
版式设计：思创景点
责任校对：牛艳敏
责任印制：沈　露

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市君旺印务有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　印　　张：30.25　　字　　数：755 千字
版　　次：2010 年 4 月第 1 版　2019 年 4 月第 3 版　　印　　次：2019 年 4 月第 1 次印刷
定　　价：98.00 元

产品编号：080994-01

# 序

时间可以是最大的敌人，也可以是最强的盟友，然而多数人选择了前者。

2009/2/21

人生在世，要赚到钱也要赚到时间。

2010/7/21

金钱投资得宜可以赢得更多金钱，时间投资得当可以赢得更多时间，后者远比前者重要，但前者往往比后者得到更多的关注。

2012/2/18

只要每日认真投资(时间)，有时需要的只是等待。

2012/3/17

存起来的钱用掉就没了，存起来的时间却可反复使用。

2012/3/18

人啊！多半用时间换取金钱，再用金钱换取时间，过程遗留的残烬叫等待；有些人的时间越来越难换取金钱；有些人的钱再也换不到时间，而是无奈、空虚、怨念、固执……或者是无法再复燃的灰烬……

2013/4/8

人们多关心工作可以带来多少金钱，却鲜少关心工作可以带来多少时间！

2013/4/28

人的一生中两个最大的财富其实是：你的健康和你的时间。

2013/9/24

花费的时间成本是用来获得时间回馈，不是金钱回馈，不能混为一谈。

2013/10/18

时间不是金钱，时间就是时间，时间要值回更多的时间。

2014/6/16

有些事，现在不付出时间，将来需要付出更多时间。

2015/11/29

其实时间并不是一个定量，除了变少还可能变多。

2016/10/9

致投资时间在此的你！

2018/3/18

# 导　　读

这份导读可以让你更了解如何使用本书。

## 字型

本书内文中与程序代码相关的文字，都用固定宽度字体来加以呈现，以与一般名词作区别。例如，JSP 是一般名词，而 `HttpServlet` 为程序代码相关文字，使用了固定宽度字体。

## 新旧版差异

本书是从《JSP & Servlet 学习笔记(第 2 版)》改版而来的，因此这里说明一下与《JSP & Servlet 学习笔记(第 2 版)》之间的差异。

就目录上可以看出的主要差异是，删除了《JSP & Servlet 学习笔记(第 2 版)》第 12 章“从模式到框架”，并由新撰写的 3 个 Spring 相关章节取代，这是为了从实际的框架中学习，而不是空谈概念；然而，Spring 那些章节并不是作为全面探讨 Spring 之用，而是作为一个衔接，希望从实际的应用程序重构中筛选出对应用程序有益的框架特性，以便逐步掌握框架的本质。


当然，照例要谈一些 Java EE 8 的功能，相关讨论会放在各章节中适当的地方。由于《JSP & Servlet 学习笔记(第 2 版)》是基于 Java EE 6，为了便于查找 Java EE 7/8 的功能介绍，如果发现页左侧有如  图示，就表示提及 Java EE 7 或 Java EE 8 功能，本书还提供了 Java EE 7/8 功能快速查询目录。

各章节的范例都做了全面改写，由于 Java EE 8 是基于 Java SE 8，范例程序代码会适当使用 Java SE 8 的特性，例如 Lambda 与 Stream API 等。

时至今日，撰写应用程序时必须有相关的安全防护概念，作为一本谈论 Web 应用程序的书，适时地提及安全概念是必要的，书中谈到了 OWASP TOP 10，讨论了 Session 防护、注入攻击、Cookie 安全、密码加盐哈希、跨域伪造请求(Cross-Site Request Forgery，CSRF)等安全基本观念，并在适当的地方介绍了 OWASP Java Encoder、Java HTML Sanitizer 等项目的使用。

## 程序范例


本书大多数范例使用完整的程序实作来展现，如果是用以下方式示范程序代码：

FirstServlet Hello.java

```
package cc.openhome;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/hello")
public class Hello extends HttpServlet {          ❶继承 HttpServlet
    @Override
    protected void doGet(                         ❷重新定义 doGet()
            HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
                                                              ❸设置响应内容类型
        response.setContentType("text/html;charset=UTF-8");

        String name = request.getParameter("name");           ❹取得请求参数

        PrintWriter out = response.getWriter();               ❺取得响应输出对象
        out.print("<!DOCTYPE html>");
        out.print("<html>");
        out.print("<head>");
        out.print("<title>Hello</title>");
        out.print("</head>");
        out.print("<body>");
        out.printf("<h1> Hello! %s!%n</h1>", name);           ❻跟用户说 Hello!
        out.print("</body>");
        out.print("</html>");
    }
}
```

范例开始的左边名称为 FirstServlet，表示可以在范例文件的 samples 文件夹中查找相应章节目录，即可找到对应的 FirstServlet 项目，而右边名称为 Hello.java，表示可以在项目中找到 Hello.java 文件。如果程序代码中出现标号与提示文字，表示后续的内文中会有对应于标号及提示的更详细说明。

原则上，建议每个项目范例都亲自动手撰写，如果由于教学时间或实现时间上的限制，本书有建议进行的练习。在范例开始前有  图示的，表示建议动手实践，而且在范例文件的 labs 文件夹中有练习项目的基础内容，可以在导入项目后，完成项目中遗漏或必须补齐的程序代码或设置。

如果文中使用以下程序代码，则表示它是一个完整的程序内容，但不是项目的一部分，主要用来展现如何撰写一个完整的文件。

```
<%@page import="java.time.LocalDateTime"%>
<%@page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JSP 范例文件</title>
```

```
</head>
<body>
<!-- 这里会依 Web 网站的时间而产生不同的响应 -->
<%= LocalDateTime.now() %>
</body>
</html
```

如果使用以下程序代码，则表示它是一个代码段，主要展现程序撰写时需要特别注意的片段。

```
// 略 ...
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
      throws java.io.IOException, ServletException {
  // 略...
  try {
    response.setContentType("text/html;charset=UTF-8");
    //略...
    out = pageContext.getOut();
    // 略...
  } catch (Throwable t) {
    // 略 ...
  } finally {
    // 略 ...
```

# 操作步骤

本书将 IDE 设定的相关操作步骤，也作为练习的一部分，你会看到如下的操作步骤说明：

(1) 运行 eclipse 文件夹中的 eclipse.exe。

(2) 出现 Eclipse Launcher 对话框时，将 Workspace 设定为 C:\workspace，单击 Launch 按钮。

(3) 执行菜单 Window|Preferences 命令，在弹出的 Preferences 对话框中，展开左边的 Server 节点，选择其中的 Runtime Environment 节点。

(4) 单击右边 Server Runtime Environments 中的 Add 按钮，在弹出的 New Server Runtime Environment 对话框中选择 Apache Tomcat v9.0，单击 Next 按钮。

(5) 单击 Tomcat installation directory 旁的 Browse 按钮，选取 C:\workspace 中解压缩的 Tomcat 文件夹，单击“确定”按钮。

# 提示框

在本书中会出现以下提示框：

**提示》》** 针对课程中提到的观点，提供一些额外的资源或思考方向，暂时忽略这些提示对课程进行的影响，但有时间的话，针对这些提示多作阅读、思考或讨论是有帮助的。

**注意》》** 针对课程中提到的观点，以提示框方式特别呈现出必须注意的一些使用方式、陷阱或避开问题的方法，看到这个提示框时请集中精力阅读。

## 综合练习

本书以“微博”项目的实现过程贯穿全书，随着每一章的进行，会在适当的时候将新介绍的技术应用至“微博”程序之中并作适当的修改，以了解完整的应用程序基本上是如何建构出来。

## 附录

本书配套资源中的范例文件包括本书全部范例，提供 Eclipse 范例项目，部分范例是 Gradle 项目，附录 A 说明如何使用这些范例项目。本书也说明如何在 Web 应用程序中整合数据库，实现范例时使用的数据库为 H2，使用方式可见 9.1 节的内容，范例若包含 H2 数据库文件*.mv.db 的话，联机时的名称与密码都是 caterpillar 与 12345678。

## 联系作者

若有与本书相关的勘误反馈等问题，可通过网站与作者联系：
http://openhome.cc

## 资源下载

本书配套资源下载：

# 目　录

# Java EE 7/8 新功能索引

# Web 应用程序简介

Chapter 1

学习目标：

- 认识 HTTP 基本特性
- 了解 GET、POST 使用时机
- 了解何为 URL/URI 编码
- 认识 Web 容器角色
- 了解 Servlet 与 JSP 的关系
- 认识 MVC/Model 2

# 1.1 Web 应用程序基础

在正式学习 Servlet/JSP 相关技术之前，要先花点时间了解一些 Web 应用程序基础知识，虽然是基础知识，却很重要。在我这几年的教学中，发现有些学员并不具备这些基础知识，或者忽略了这些基础知识中的一些细节，如 HTML(HyperText Markup Language)、HTTP (HyperText Transfer Protocol)、URI(Uniform Resource Identifier)甚至文字编码的问题等。

当然，谈这些内容并不是要你成为这几个名词的专家，而是在以后学习 Servlet/JSP 相关技术时，若有这些基础知识，就能真正理解相关技术背后的原理，而不会沦落到死背 API(Application Programming Interface)的窘境。

## 1.1.1 关于 HTML

本书介绍的 Web 应用程序，是由客户端(Client)与服务器端(Server)两个部分组成的，客户端基本是浏览器，服务器端是指 HTTP 服务器及运行在其上的相关资源，后面会使用浏览器来作为客户端代表，以 Web 网站来代称 HTTP 服务器及运行在其上的相关资源。

浏览器会请求 Web 网站，就本书来说，Web 网站必须产生 HTML，HTML 是以标签的方式来定义文件结构的。下面是一个简单的 HTML 范例：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>HTML 范例文件</title>
    </head>
    <body>
        <img src="images/caterpillar.jpg">哈喽！请输入...<br><br>
        <form method="get" action="echo" name="message">
            名称：<input type="text" name="name"><br><br>
            <button>发送</button>
        </form>
    </body>
</html>
```

整份 HTML 文件的定义编写在<html>与</html>标签之间。在文件开始之前，浏览器必须先解析所有标签，<head>与</head>标签间用来定义文件主体前，提供给浏览器的一般信息，像是该范例的文件编码与标题信息等，就分别定义在<meta>以及<title>与</title>标签之间的内容。

接着浏览器针对文件主体内容解析，以便进行画面呈现与定义行为，文件主体内容定义在<body>标签中，例如<br>告诉浏览器换行，范例文件中有个代表图片的<img>标签，这告知浏览器下载指定图片并显示。HTML 标签可以拥有属性，用来定义标签的额外信息，像是图片的来源(src 属性)。<form>标签定义了窗体，可让用户填写一些将要发送至 Web 网站的信息，其中还使用了<input>标签，分别定义输入字段及发送按钮。

简而言之，浏览器从 Web 网站取得这份 HTML 文件之后，就可以按照 HTML 定义的结构等信息进行画面的绘制。图 1.1 所示为大致的 HTML 标签与对应的画面呈现。

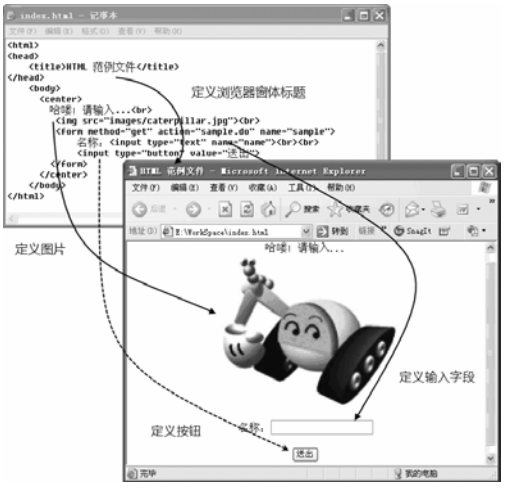



图 1.1　浏览器按 HTML 的结构等信息进行画面绘制

以上描述是一种测试，如果连以上的基本 HTML 都不甚了解，建议先寻找 HTML 相关的文件或书籍进行大致的了解。w3schools 的 HTML5 Tutorial(参考网址 www.w3schools.com/html/)是不错的快速入门文件，足以应付阅读本书所需的 HTML 基础。

## 1.1.2　URL、URN 与 URI

既然 Web 应用程序的文件等资源是放在 Web 网站上的，而 Web 网站栖身于广大网络之中，就必须要有个方式，告诉浏览器到哪里取得文件等资源。通常会听到有人这么说：“你要指定 URL”，偶尔会听到有人说：“你要指定 URI”。那么到底什么是 URL、URI？甚至你可能还听过 URN。首先，三个名词都是缩写，其全名分别如下。

- URL：Uniform Resource Locator
- URN：Uniform Resource Name
- URI：Uniform Resource Identifier

从历史的角度来看，URL 标准最先出现，早期 U 代表 Universal(万用)，标准化之后代表着 Uniform(统一)。正如名称所指出，URL 的主要目的是以文字方式来说明互联网上的资源如何取得。就早期的 RFC 1738(参考网址 tools.ietf.org/html/rfc1738)规范来看，URL 的主要语法格式为：

<scheme>:<scheme-specific-part>

协议(scheme)指定了以何种方式取得资源。下面是一些协议名的例子：

- FTP(File Transfer Protocol，文件传输协议)
- HTTP(Hypertext Transfer Protocol，超文本传输协议)
- Mailto(电子邮件)
- File(特定主机文件名)

协议之后跟随冒号，协议特定部分(scheme-specific-part)的格式依协议而定，通常会是：

//<用户>:<密码>@<主机>:<端口号>/<路径>

举例来说，若资源放置在 Web 网站上，如图 1.2 所示。

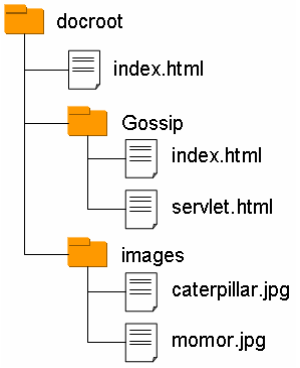



图 1.2　HTTP 服务器上的资源

假设主机名为 openhome.cc，要以 HTTP 协议取得 Gossip 目录中的 index.html 文件，端口号 8080，则必须使用以下 URL(见图 1.3)：

```
http://openhome.cc:8080/Gossip/index.html
```

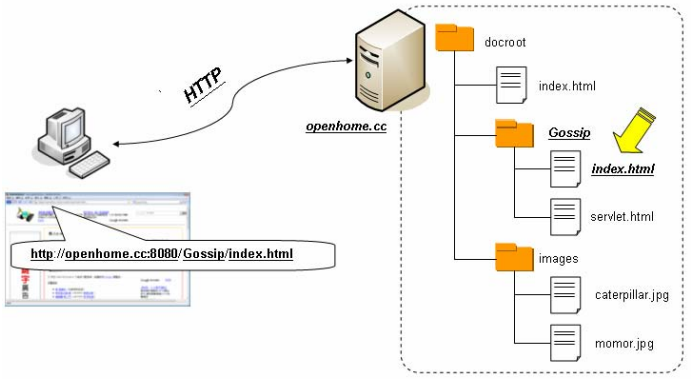



图 1.3　以 URL 指定资源位置等信息

如果想取得计算机文件系统中 C:\workspace 下的 jdbc.pdf 文件，则可以指定如下 URL 格式：

```
file://C:/workspace/jdbc.pdf
```

URN 代表某个资源独一无二的名称，就早期规范 RFC 2141(参考网址 www.ietf.org/rfc/rfc2141.txt)来看，URN 的主要语法格式为：

```
<URN> ::= "urn:" <NID> ":" <NSS>
```

举例来说，《Java JDK 9 学习笔记》的国际标准书号(International Standard Book Number，ISBN)若用 URN 来表示的话，应为 urn:isbn:978-7-302-50118-3，这就是 URN 的一个例子。

由于 URL 或 URN 的目的都是用来标识某个资源，后来制定了 URI 标准，而 URL 与 URN 成为 URI 的子集。就 RFC 3986(参考网址 www.ietf.org/rfc/rfc3986.txt)的规范来看，URI 的主要语法格式主要为：

```
URI  = scheme ":" hier-part [ "?" query ] [ "#" fragment ]
hier-part  = "//" authority path-abempty
          / path-absolute
          / path-rootless
          / path-empty
```

在规范中有个语法的实例对照：

```
foo://example.com:8042/over/there?name=ferret#nose
 \_/   \______________/\_________/ \_________/ \__/
  |          |            |           |        |
scheme     authority      path       query   fragment
  |   _____________________|__
 / \ /                     \
 urn:example:animal:ferret:nose
```

在制定 URI 规则之后，一些标准机构如 W3C(World Wide Web Consortium)文件中，就算指的是 Web 网站上的资源，多半也会使用 URI 这个名称，不过许多人已经习惯使用 URL 名称来表示 Web 网站上的资源，因而 URL 这个名称仍广为使用。不少既有的技术，如 API 或者相关设定中，也会出现 URL 字样，然而为了符合规范，本书将统一采用 URI 来表示。

如果想对 URL、URI 与 URN 的历史演变与标准发布有更多的了解，可以参考 Wikipedia (http://www.wikipedia.org/)的 Uniform Resource Identifier(参考网址 http://en.wikipedia.org/wiki/Uniform_Resource_Identifier)。

## 1.1.3 关于 HTTP

前面一直提到 HTTP，这是一种通信协议，指架构在 TCP/IP 之上的应用层协议。通信协议基本上就是两台计算机间对谈沟通的方式，例如客户端要跟服务器请求联机，假设就是跟服务器说声 CONNECT，服务器响应 PASSWORD 表示要求密码，客户端再进一步跟服务器说声 PASSWORD 1234，表示这是所需的密码，诸如此类，如图 1.4 所示。


图 1.4　通信协议是计算机间沟通的一种方式

按不同的联机方式与所使用的网络服务而定，会有不同的通信协议。例如，发送信件时会使用 SMTP(Simple Mail Transfer Protocol)，传输文件时会使用 FTP，下载信件时会使用 POP3(Post Office Protocol 3)等，而浏览器跟 Web 网站之间使用的沟通方式是 HTTP，它有两个基本但极为重要的特性：

- 基于请求(Request)/响应(Response)模型
- 无状态(Stateless)协议

### 1. 基于请求/响应模型

HTTP 是基于请求/响应的通信协议，浏览器对 Web 网站发出取得资源的请求，Web 网站将该请求要求的资源响应给浏览器，是一种很简单的通信协议，没有请求就不会有响应。

HTTP 1.1 支持 Pipelining，浏览器可以在同一个联机中对 Web 网站发出多次请求，然

而 Web 网站必须依请求顺序来进行响应。

HTTP 2.0 支持 Server Push，允许 Web 网站在收到请求后，主动推送必要的 CSS、JavaScript、图片等资源到浏览器，不用浏览器后续再对资源发出请求。

HTML5 支持 Server Sent Event，在请求发送至 Web 网站后，Web 网站的响应可一直持续(始终处于“下载”状态)，支持 Server Sent Event 的浏览器能知道响应中有哪些个别数据。

然而，无论是哪种情况，浏览器没有发出请求，Web 网站就不会有响应的基本模型仍然没有改变。

#### 2. 无状态协议

在 HTTP 协议之下，Web 网站是个健忘的家伙，Web 网站响应客户端之后，就不会记得客户端的信息，因此 HTTP 又称为无状态的通信协议，如图 1.5 所示。

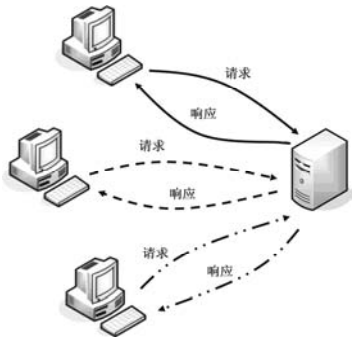



图 1.5　HTTP 是基于请求/响应的无状态通信协议

明白 HTTP 这两个基本特性很重要，这样才知道 Web 应用程序可以做到什么，又有哪些做不到，才能知道之后要介绍的 MVC 模式(Model-View-Controller Pattern)为何要变化为 Model 2 模式，之后谈到会话管理(Session management)时，才能知道会话管理的基本原理，并针对需求采取适当的会话管理机制。

### 1.1.4　HTTP 请求方法

HTTP 定义了 GET、POST、PUT、DELETE、HEAD、OPTIONS、TRACE 等请求方式，在 JavaScript 尚未使用的年代，撰写 Servlet 或 JSP 时最常接触的就是 GET 与 POST，这是因为过去发送请求信息时，主要以 HTML 窗体发送为主，而 HTML 的<form>标签在 method 属性上只支持 GET 与 POST。不过，在 JavaScript 兴起及前端工程当道之后，因为可以通过 JavaScript 来发出各种请求方法，也就不再局促于 GET 与 POST 了。

由于本书主要是谈 Servlet 与 JSP，因此还是就 GET 与 POST 来进行说明。

#### 1. GET 请求

GET 请求，顾名思义，就是向 Web 网站取得指定的资源，在发出 GET 请求时，必须一并告诉 Web 网站所请求资源的 URL，以及一些标头(Header)信息。图 1.6 所示是一个 GET 请求的发送范例。


图 1.6 GET 请求范例

在图 1.6 中，请求标头提供了 Web 网站一些浏览器相关的信息，Web 网站可以使用这些信息来进行响应处理。例如，Web 网站可以从 User-Agent 中得知使用者的浏览器种类与版本，从 Accept-Language 了解浏览器接受哪些语系的内容响应等。

请求参数通常是使用者发送给 Web 网站的信息，这个信息可利用窗体或 JavaScript 来进行发送，Web 网站有了这些信息，可以进一步针对使用者请求进行正确的响应。请求参数是路径之后跟随一个问号(?)，然后是请求参数名称与请求参数值，中间以等号(=)表示成对关系。若有多个请求参数，则以&字符连接。使用 GET 方式发送请求，浏览器的地址栏上也会出现请求参数信息，如图 1.7 所示。


图 1.7 GET 的请求参数会出现在地址栏上

GET 请求可以发送的请求参数长度有限，这根据浏览器而有所不同。Web 网站也会设定长度上的限制，对于太大量的数据并不适合用 GET 方式来进行请求，这时可以改用 POST。

### 2. POST 请求

对于大量或复杂的信息发送(如文件上传)，通常会采用 POST 来进行发送。图 1.8 所示是一个 POST 发送的范例。


图 1.8 POST 请求范例

POST 将请求参数移至最后的信息体(Message body)之中，由于信息体的内容长度不受限制，大量数据的发送都会使用 POST 方法，而由于请求参数移至信息体，地址栏上也就不会出现请求参数。对于一些较敏感的信息，例如密码，即使长度不长，通常也会改用 POST 的方式发送，以避免因为出现在地址栏上而被直接窥视。

**注意**

虽然在 POST 请求时，请求参数不会出现在地址栏上，然而在非加密联机的情况下，若请求被第三方获取了，请求参数仍然是一目了然，机密信息请务必在加密联机下传送。

在考虑使用 GET 或 POST 时，实际上并不是只考虑数据长度，以及地址栏是否会出现请求参数，想知道应该选用哪个 HTTP 方法，最好的方式就是对 HTTP 的各个方法规范有进一步的认识。

### 3. 敏感信息

如刚才谈到的，像密码之类的敏感信息，不适合使用 GET 发送，除了可能被邻近之人偷窥，或者是被浏览器过于方便的网址自动补齐记录下来之外，另一个问题还出现在 HTTP 的 Referer 标头上，这是用来告知 Web 网站，浏览器是从哪一个页面链接到目前网页。如果地址栏出现了敏感信息，之后又链接到另一个网站，该网站就有可能通过 Referer 标签得到敏感信息。

### 4. 书签设置考虑

由于浏览器书签功能是针对地址栏，因此想让用户针对查询结果设定书签的话，可以使用 GET。POST 后新增的资源不一定会有个 URI 作为识别，基本上无法让用户设定书签。

### 5. 浏览器快取

GET 的响应是可以被快取的，最基本的就是指定的 URI 没有变化时，许多浏览器就会从快取中取得数据。不过，服务器端可以指定适当的 Cache-Control 标头来避免 GET 响应被快取的问题。

至于 POST 的响应，许多浏览器(但不是全部)并不会快取，不过 HTTP 1.1 规范中指出，如果服务器端指定适当的 Cache-Control 或 Expires 标头，仍可以建议浏览器 POST 响应进行快取。

### 6. 安全与等幂

由于传统上发送敏感信息时，并不会通过 GET，因而有人会认为 GET 不安全，这其实是个误会，或者说对安全的定义不同。在 HTTP 1.1 对 HTTP 方法的定义中，区分了安全方法(Safe methods)与等幂方法(Idempotent methods)。

安全方法是指在实际应用程序时，必须避免有使用者非预期中的结果。惯例上，GET 与 HEAD(与 GET 同为取得信息，不过仅取得响应标头)对使用者来说就是“取得”信息，不应该被用来“修改”与用户相关的信息，如进行转账或删除数据之类的动作，GET 是安全方法，这与传统印象中 GET 比较不安全相反。

相对之下，POST、PUT 与 DELETE 等其他方法就语义上来说，代表着对使用者来说可能会产生不安全的操作，如删除用户的数据等。

安全与否并不是指方法对服务器端是否产生副作用(Side effect)，而是指对使用者来说该动作是否安全，GET 也有可能在服务器端产生副作用。

对于副作用的进一步规范是方法的等幂特性，GET、HEAD、PUT、DELETE 是等幂方法，也就是单一请求产生的副作用，与同样请求进行多次的副作用必须是相同的(而不是无副作用)。举例来说，若使用 DELETE 的副作用是某笔数据被删除，相同请求再执行多次的结果就是该笔数据不存在，而不是造成更多的数据被删除。OPTIONS 与 TRACE 本身就不该有副作用，所以也是等幂方法。

HTTP 1.1 中的方法去除上述的等幂方法之后，只有 **POST** 不具有等幂特性，HTTP 1.1 对 POST 的规范，是要求指定的 URI“接受”请求中附上的实体(Entity)，例如存储为文档、新增为数据库中的一笔数据等，要求服务器接受的信息是附在请求本体(Body)而不是在 URI。也就是说，POST 时指定的 URI 并不代表能取得 POST 时的资源(如文件等)，每次 POST 的副作用可以不同。

**提示**

这是使得 POST 与 PUT 有所区别的特性之一，在 HTTP 1.1 规范中，PUT 方法要求将附加的实体存储于指定的 URI，如果指定的 URI 下已存在资源，附加的实体用来进行资源的更新，如果资源不存在，则将实体存储下来并使用指定的 URI 来代表它，这也符合等幂特性。

例如，用 PUT 来更新用户基本数据，只要附加于请求的信息相同，一次或多次请求的副作用都会是相同的，也就是用户信息保持为指定的最新状态。

前面讲过，就窗体发送而言，可以借助<form>的 method 属性来设定使用 GET 或 POST 方式来发送数据，不设定 method 属性的话，默认会使用 GET：

```
...
   <form method="get" action="download " name="filename">
      名称：<input type="text" name="name"><br><br>
      <input type="button" value="送出">
   </form>
...
```

**提示**

现在很多 Web 服务或框架支持 REST 风格的架构，REST 全称为 REpresentational State Transfer，REST 架构由客户端/服务器端组成，两者间的通信机制是无状态的(Stateless)，在许多概念上，与 HTTP 规范不谋而合(REST 架构基于 HTTP 1.0，与 HTTP 1.1 平行发展，但不限于 HTTP)。

符合 REST 架构原则的系统称为 RESTful，以基于 HTTP 的基本书签程序来说，可搭配 JavaScript 来发出 GET、POST 外的其他请求，POST/bookmarks 用来新增一笔资料，GET /bookmarks/1 用来取得 ID 为 1 的书签，PUT/bookmarks/1 用来更新 ID 为 1 的书签数据，而 DELETE/bookmarks/1 用来删除 ID 为 1 的书签数据。

## 1.1.5 有关 URI 编码

HTTP 请求参数必须使用请求参数名称与请求参数值，中间以等号(=)表示成对关系。现在问题来了，如果请求参数值本身包括“=”符号怎么办？又或者你想发送的请求参数值是 https://openhome.cc 这个值呢？假设是 GET 请求，直接这么发送是不行的：

```
GET/Gossip/download?url=https://openhome.cc HTTP/1.1
```

### 1. 保留字符

在 URI 规范中定义了保留字符(Reserved character)，例如“:”“/”“?”“&”“=”“@”“%”等字符，在 URI 中有它们各自的作用。如果要在请求参数上表达 URI 中的保留字符，

必须在“%”字符之后以十六进制数值表示方式，来表示该字符的八个位数值。

例如，“:”字符真正存储时的八个位为 00111010，用十六进制数值来表示则为 3A，所以必须使用“%3A”来表示“:”；“/”字符存储时的八个位为 00101111，用十六进制表示则为 2F，所以必须使用“%2F”来表示“/”字符，所以若发送的请求参数值是 https://openhome.cc，则必须使用以下格式：

```
GET/Gossip/download?url=https%3A%2F%2Fopenhome.cc HTTP/1.1
```

这是 URI 规范中的百分比编码(Percent-Encoding)，也就是俗称的 URI 编码或 URL 编码。如果想知道某个字符的 URI 编码是什么，在 Java 中可以使用 `java.net.URLEncoder` 类的静态 `encode()` 方法来进行编码的动作(相对地，要译码则使用 `java.net.URLDecoder` 的静态 `decode()` 方法)。例如：

```
String text = URLEncoder.encode("http://openhome.cc ", "ISO-8859-1");
```

知道这些有什么用？例如，你想给某人一段 URI，让他直接单击就可以连接到你想要让他看到的网页，你给他的 URI 在请求参数部分就要注意 URI 编码。

不过在 URI 之前，HTTP 在 `GET`、`POST` 时也对保留字作了规范，这与 URI 规范的保留字有所差别。其中一个差别就是在 URI 规范中，空格符的编码为%20，而在 HTTP 规范中空白的编码为“+”，`java.net.URLEncoder` 类的静态方法 `encode()` 产生的字符串，空格符的编码就为“+”。

### 2. 中文字符

URI 规范的 URI 编码针对的是字符 UTF-8 编码的八位数值，如果请求参数都是 ASCII 字符，那没什么问题，因为 UTF-8 编码与在 ASCII 字符的编码部分是兼容的，也就是使用一个字节，编码方式就如先前所述。

但在非 ASCII 字符方面，如中文，在 UTF-8 编码下，会使用三个字节来表示。例如，“林”这个字在 UTF-8 编码下的三个字节，对应至十六进制数值表示就是 E6、9E、97，所以在 URI 规范下，请求参数中要包括“林”这个中文，表示方式就是“%E6%9E%97”。例如：

```
https://openhome.cc/addBookmar.do?lastName=%E6%9E%97
```

有些初学者会直接打开浏览器输入如图 1.9 所示内容，告诉我：“URI 也可以直接打中文啊！”

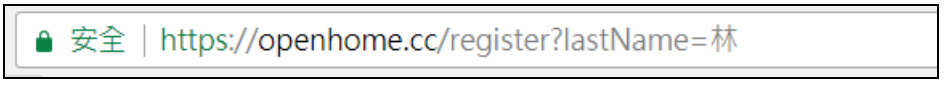



图 1.9　浏览器地址栏真的可以输入中文

不过你可以将地址栏复制，粘贴到纯文本文件中，就会看到 URI 编码的结果，这其实是因为现在的浏览器很聪明，会自动将 URI 编码显示为中文。无论如何，在 URI 规范上若以上面方式发送请求参数，Web 网站处理请求参数时，必须使用 UTF-8 编码来取得正确的“林”字符。

然而在 HTTP 规范下的 URI 编码，并不限使用 UTF-8，例如在一个 MS950 网页中，若窗体使用 `GET` 发送“林”这个中文字，则地址栏中会出现：

```
https://openhome.cc/register?lastName=%AA%4C
```

**提示>>>** 若是%AA%4C，由于单独看%4C 的话，代表着字符 L，浏览器也可以发送%AAL。

这是因为“林”这个中文字的 MS950 编码为两个字节，若以十六进制表示，则分别为 AA、4C。如果通过窗体发送，由于网页是 MS950 编码，则浏览器会自动将“林”编码为“%AA%4C”，Web 网站处理请求参数时，就必须指定 MS950 编码，以取得正确的“林”汉字字符。

若使用 java.net.URLEncoder 类的静态 encode()方法来做这个编码的动作，则可以像下面这样得到“%AA%4C”的结果：

```
String text = URLEncoder.encode("林 ", " MS950");
```

同理可推，如果网页是 UTF-8 编码，而你通过窗体发送，则浏览器会自动将“林”编码为“%E6%9E%97”。若使用 java.net.URLEncoder 类的静态 encode()方法来做编码的动作，则可像下面这样得到“%E6%9E%97”的结果：

```
String text = URLEncoder.encode("林", "UTF-8");
```

知道这些要做什么吗？你应该隐约感觉到了：“我们会发送中文”。中文是如何编码的？到服务器端后又是如何译码的？这些问题必须先搞清楚。随便问个“为什么我收到的是乱码？”“为什么数据库中是乱码？”，往往解决不了问题。如果具备这些基础，之后在说明 Servlet/JSP 中如何接收包括中文字符的请求参数时，你才能理解如何使用某些 API 进行正确的编码转换动作。

**提示>>>** 由于一些历史性的原因，编码问题错综复杂，如果有兴趣进一步探究，可以参考《乱码 1/2》(参考网址：http://openhome.cc/Gossip/Encoding/)。

## 1.1.6 后端与前端

现在这个世界通常不会只使用单一技术来完成 Web 应用程序，就今天来说，若粗略区分，Web 应用程序技术可分为前端(Frontend)与后端(Backend)，而就本书的范畴来说，主要是在谈论 Servlet/JSP、Spring MVC 等技术，而这些是属于后端的技术。

举例来说，下面是个 JSP 的例子(见图 1-10)，当浏览器请求这个 JSP 时，会根据 Web 网站上的时间产生响应内容：

```
<%@page import="java.time.LocalDateTime"%>
<%@page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>JSP 范例文件</title>
    </head>
    <body>
        <!-- 这里会依 Web 网站的时间而产生不同的响应 -->
        <%= LocalDateTime.now() %>
    </body>
</html>
```

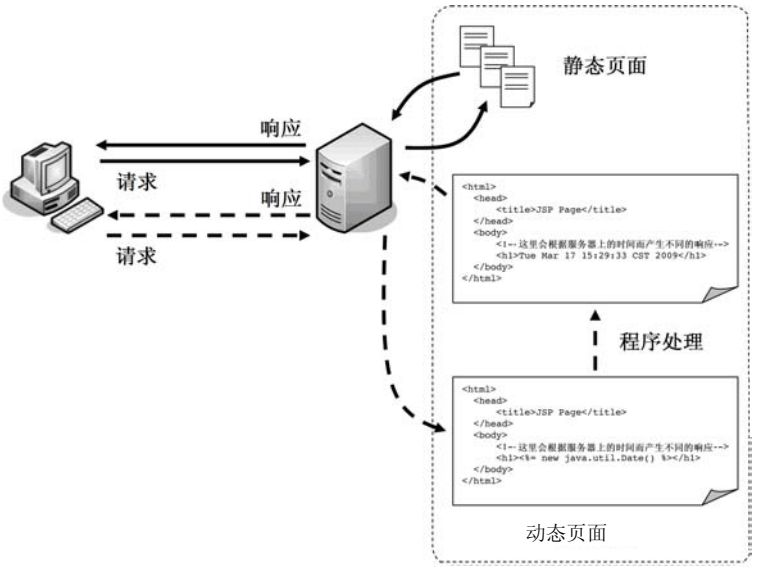



图 1.10　JSP 会自动产生响应内容

JSP 会在 Web 网站上执行程序代码，产生响应内容后传回，例如 PHP(Hypertext Preprocessor)、ASP(Active Server Page)等，都是属于这类技术。

在响应内容传至浏览器之后，若其中包含了 JavaScript 程序代码，浏览器会执行 JavaScript，由于浏览器直接面对使用者，以 JavaScript 为出发点，在浏览器上执行的相关技术称为前端技术。而相对来说，执行于 Web 网站上的相关技术，就被称为后端技术。

有些初学者常分不清楚 JavaScript 与 Servlet/JSP 的关系，由于 JSP 中可以编写 Java 程序代码，而 JSP 中又可以编写 JavaScript，所以 JavaScript 当初命名时，又套上了个 Java 的名字在前头，让许多学习 JavaScript 或 JSP 的人，误以为 JavaScript 与 JSP(或 Java)有直接的关系，事实上并没有这回事。

如前面所讲的，Servlet/JSP 是后端技术，执行于 Web 网站的内存空间，而 JavaScript 属于前端技术，执行于浏览器，也就是用户计算机上的内存空间，两个内存空间实体位置并不同，无法做直接的互动(例如以 Servlet/JSP 直接取得 JavaScript 执行时期的变量值)，必须通过网络经由 HTTP 来进行互动、数据交换等动作，以完成应用程序的功能，如图 1.11 所示。

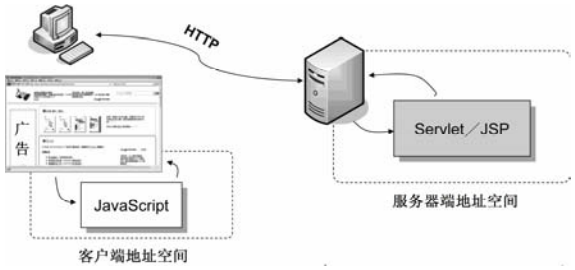



图 1.11　Servlet/JSP 与 JavaScript 执行于不同的地址空间

在今后学习或应用 JSP(或其他后端技术)的过程中，也会在 JSP 网页中写一些内嵌的

JavaScript，这些 JavaScript 并不是在 Web 网站上执行的，Web 网站如同 HTML 标签一样，将 JavaScript 传给浏览器，浏览器接收到内容后再处理 HTML 标签与执行 JavaScript。

对处理 JSP 内容的服务器端而言，内嵌的 JavaScript 跟 HTML 标签没有两样，如果了解两者的差别，就不会有所谓“可以直接让 JavaScript 取得 `request` 中的属性吗？”“为什么 JSP 没有执行 JavaScript？”或“可以直接用 JavaScript 取得 JSTL 中`<c:if>`标签的 `test` 属性吗？”这样的问题。

## 1.1.7 Web 安全概念

在这个各式系统入侵事件频传的年代，不用太多强调，每个人都知道系统安全的重要性。不注重安全而带来的损失不单是经济层面，也会面临法律问题；不注重安全不单是危及商誉的问题，也有可能阻碍政府政策的推动，甚至牵动国家安全等问题。

安全是一个复杂的议题，最好的方式是有专责部门、专职人员、专门流程、专业工具，以及时时实施安全教育训练等。虽说如此，在学习程序设计，特别是 Web 网站相关技术时，若能一并留意基本的安全概念，在实际撰写应用程序时避免显而易见的安全弱点，对于应用程序的整体安全来说，也是不无小补。

就 Web 安全这块来说，想要认识基本的安全弱点从何产生，可以从 OWASP Top 10 (www.owasp.org/index.php/Category:OWASP_Top_Ten_Project)出发，这是由 OWASP(Open Web Application Security Project，开放式 Web 应用程序安全项目)发起的计划之一，于 2002 年发起，针对 Web 应用程序最重大的十个弱点进行排行，首次 OWASP Top 10 于 2003 年发布，2004 年做了更新，之后每三年改版一次，就撰写这段文字的时间点来说，最新版的 OWASP Top 10 于 2017 年 11 月正式发布。图 1.12 所示为 OWASP Top 10 中 2013 年与 2017 年十大弱点比较。

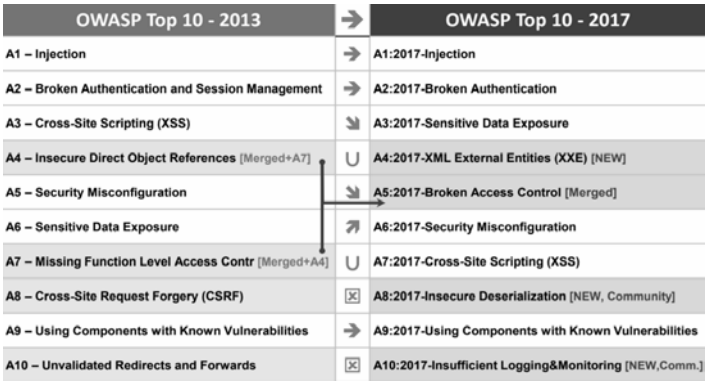



图 1.12　OWASP Top 10 中 2013 年与 2017 年十大弱点比较

如果对 Web 应用程序设计有基本认识，查看 2013 年与 2017 年的十大弱点内容时就会发现，在产生弱点的原因中，有些异常简单，如未经验证的输入就能导致各式的注入攻击，未经过滤的输出就可能引发 XSS 攻击等，若能在撰写程序时多一份留意，至少能让 Web 网站不至于赤裸裸地暴露出这些弱点。

以 OWASP Top 10 作为起点，进一步可以留意 CWE(Common Weakness Enumeration，

通用弱点列表) (cwe.mitre.org)。这个列表始于 2005 年，收集了近千个通用的软件弱点。另外，针对特定软件漏洞，可以查看 CVE(Common Vulnerabilities and Exposures，公共漏洞和暴露)(cve.mitre.org)数据库，CVE 会就特定软件发生的安全问题给予 CVE-YYYY-NNNN 形式的编号，以便于通报、查询、交流等，如 2017 年底的 CPU“推测执行”(Speculative execution)安全漏洞，就发出了 CVE-2017-5754、CVE-2017-5753 与 CVE-2017-5715 变种漏洞的 CVE 通报。

在本书中，会适当地提示一些 Web 安全问题，在相关的操作中，在可能的情况下，会提示可能形成弱点的原因。然而，本书毕竟不是谈安全的专著，范例终究是以介绍 Servlet/JSP 相关技术为主体，必然会有所简化以彰显技术上的重点，无法全面涵盖相关安全设计。

实际上，任何非谈论安全为主体的技术相关书籍都是如此，范例都是经过简化的，无论如何，绝对不要把范例的做法或概念直接用于实际应用程序，在安全议题面前，在有心破坏的使用者面前，这些范例都是脆弱而不堪一击的。

## 1.2 Servlet/JSP 简介

在学习 Java 程序语言时，有个重要的概念：“JVM(Java Virtual Machine)是 Java 程序唯一认识的操作系统，其可执行文件为.class 文件。”基于这一概念，在编写 Java 程序时，必须了解 Java 程序如何与 JVM 这个虚拟操作系统进行通信，JVM 如何管理 Java 程序中的对象等问题。

在学习 Servlet/JSP 时，也有个重要概念：“Web 容器(Container)是 Servlet/JSP 唯一认得的 HTTP 服务器。”如果希望用 Servlet/JSP 编写的 Web 应用程序可以正常运作，就必须知道 Servlet/JSP 如何与 Web 容器沟通，Web 容器如何管理 Servlet/JSP 的各种对象等问题。

### 1.2.1 何谓 Web 容器

对于 Java 程序而言，JVM 是其操作系统，.java 文件会编译为.class 文件，.class 对于 JVM 而言，就是其可执行文件。Java 程序基本上只认得一种操作系统，那就是 JVM。

在开始编写 Servlet/JSP 程序时，必须接触容器的概念，容器这个名词也用在如 `List`、`Set` 这类的 `Collection` 上，也就是用来持有、保存对象的集合(Collection)对象。对于编写 Servlet/JSP 来说，容器的概念更广，它不仅持有对象，还负责对象的生命周期与相关服务的连接。

在具体层面，容器就是一个用 Java 写的程序，运行于 JVM 之上，不同类型的容器负责不同的工作，若以运行 Servlet/JSP 的 Web 容器(Web Container)来说，也是一个 Java 写的程序。编写 Servlet 时，会接触 `HttpServletRequest`、`HttpServletResponse` 等对象，想想看，HTTP 那些文字性的通信协议，如何变成 Servlet/JSP 中可用的 Java 对象，其实就是容器中的剖析与转换。

在抽象层面，可以将 Web 容器视为运行 Servlet/JSP 的 HTTP 服务器。就如同 Java 程序仅认得 JVM 这个操作系统，Servlet/JSP 程序在抽象层面上，也仅认得 Web 容器这个

被抽象化的 HTTP 服务器，只要 Servlet/JSP 撰写时符合 Web 容器的标准规范，Servlet/JSP 就可以在各种不同厂商实现的 Web 容器上运行，而不用理会底层真正的 HTTP 服务器是什么。

本书将会使用 Apache Tomcat(tomcat.apache.org)作为范例运行的 Web 容器。若以 Tomcat 为例，容器的角色位置可以用图 1.13 来表示。

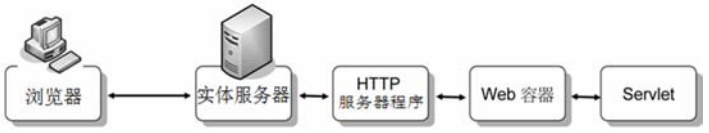



图 1.13　从请求到 Servlet 处理的线性关系

就如同 JVM 介于 Java 程序与实体操作系统之间，Web 容器介于实体 HTTP 服务器与 Servlet 之间，也正如编写 Java 程序时必须了解 JVM 与 Java 应用程序之间如何互动，编写 Servlet/JSP 时也必须知道 Web 容器如何与 Servlet/JSP 互动，以及如何管理 Servlet 等事实(JSP 最后也是转译、编译、加载为 Servlet，在容器的世界中，真正负责请求、响应的是 Servlet)。

下面是一个请求/响应的基本例子：

(1) 浏览器对 HTML 服务器发出请求。

(2) HTTP 服务器收到请求，将请求转由 Web 容器处理，Web 容器会剖析 HTTP 请求内容，创建各种对象(如 `HttpServletRequest`、`HttpServletResponse`、`HttpSession` 等)。

(3) Web 容器由请求的 URI 决定要使用哪个 Servlet 来处理请求(开发人员要定义 URI 与 Servlet 的对应关系)。

(4) Servlet 根据请求对象(`HttpServletRequest`)的信息决定如何处理，通过响应对象(`HttpServletResponse`)来创建响应。

(5) Web 容器与 HTTP 服务器沟通，HTTP 服务器将相关响应对象转换为 HTTP 响应并传回给浏览器。

以上是了解 Web 容器如何管理 Servlet/JSP 的一个例子。不了解 Web 容器行为容易产生问题，举例来说，Servlet 执行在 Web 容器之中，Web 容器由服务器上的 JVM 启动，JVM 本身就是服务器上的一个可执行程序，当一个请求来到时，Web 容器会为每个请求分配一个线程(Thread)，如图 1.14 所示。

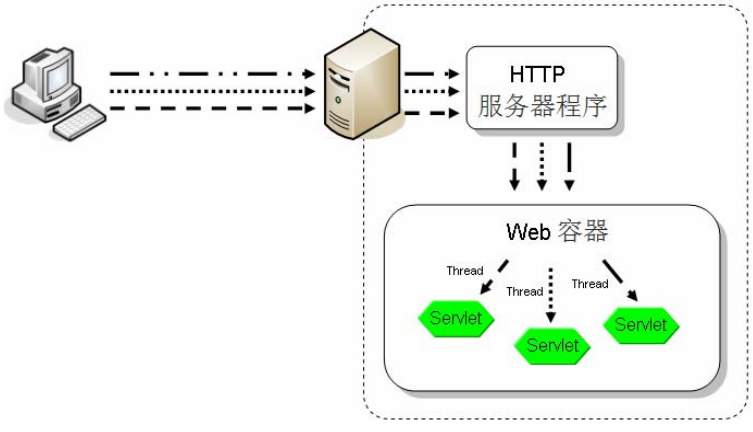



图 1.14　Web 容器为每个请求分配一个线程

如果有多个请求，就会有多个线程来各自处理，而不是重复启动多次 JVM。线程就像是进程中的轻量级流程，由于不用重复启动多个进程，可以大幅减轻性能负担。

要注意的是，Web 容器可能会使用同一个 Servlet 实例来服务多个请求。也就是说，多个请求下，就相当于多个线程在共享存取一个对象，因此得注意线程安全的问题，避免引发数据错乱，造成如 A 用户登录后看到 B 用户的数据这类问题。

其实不仅是使用 Servlet/JSP 要了解 Web 容器，Java 各平台的解决方案都对底层作了抽象化，在各平台上都会有对应的容器解决方案。编写 EJB 就要理解 EJB 容器(EJB Container)的行为，编写应用程序客户端就要理解应用程序客户端容器(Application client container)的行为，不理解容器的行为就容易引发程序执行上的各种问题，甚至造成安全弱点。图 1.15 所示是摘自 Java EE 8 Tutorial(javaee.github.io/tutorial/toc.html)中 Java EE Containers(javaee.github.io/tutorial/overview005.html)文件的容器示意。

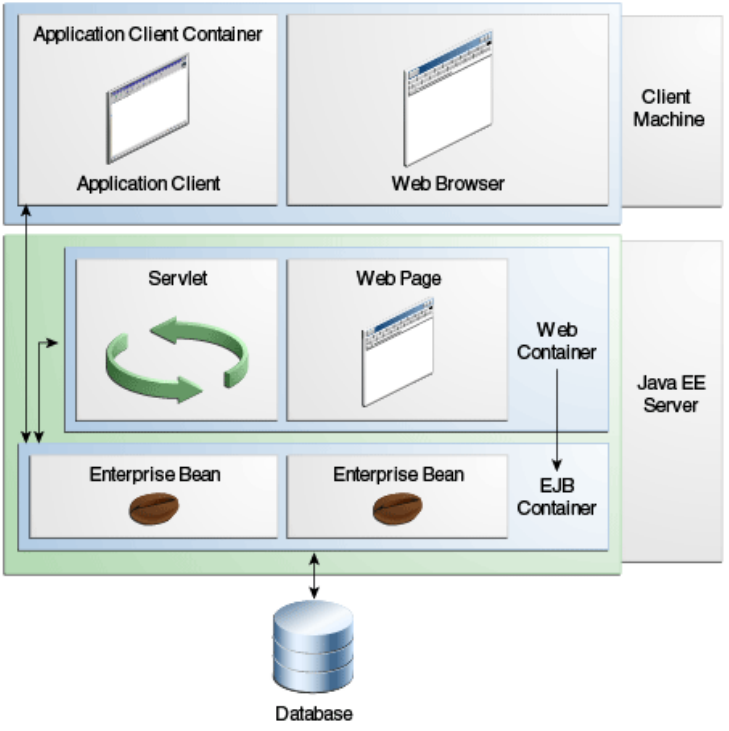



图 1.15　Java EE 服务器与容器

## 1.2.2　Servlet 与 JSP 的关系

本书从开始到现在一直在谈 Servlet，这是因为 Servlet 与 JSP 是一体两面，JSP 会被 Web 容器转译为 Servlet 的.java 源文件、编译为.class 文件，然后加载容器，因此最后提供服务的还是 Servlet 实例。这也是为什么始终在谈 Servlet 的原因，要想完全掌握 JSP，也必须先对 Servlet 有相当程度的了解，才不会一知半解，遇到错误无法解决。

也许有人会说，有必要掌握 JSP 吗？毕竟自 Java EE 6 中规范的 JSP 2.2 之后，JSP 本身就没有显著的改进了，虽然 Java EE 7 规范中是 JSP 2.3，但只是做些规范维护，主要是

因为 Expression Language、JSF 技术做了一些调整，而在 Java EE 8 之中，JSP 规范仍维持在 2.3。

这一方面是由于有些商业性考虑，另一方面则是因为前端技术的兴起。就今天来说，若能与前端技术相关开发者适当配合，JSP 已经不是撰写的主要选择，不过，既有的应用程序，不少是基于 JSP 撰写，若有 JSP 的基础，将来转换使用其他的页面模板技术就容易上手。

至于 Servlet 规范，仍持续在演变，特别是在 Java EE 8 中，Servlet 从 3.1 版号跳到了 4.0，用以突显其规范上有着重大不同；在 Java 的 Web 开发这块，一些重大 Web 框架，例如 Spring MVC，仍是基于 Servlet，如果能掌握 Servlet，在使用这类框架时，对理解底层细节或者进行框架细节控制会有很大的帮助。

因而，无论是从掌握 JSP 的角度来看，或者是能灵活运用基于 Servlet 的 Web 框架来看，掌握 Servlet 都是必要的！

先来看看一个基本的 Servlet 长什么样子。

```
package cc.openhome;

import java.io.IOException;
import java.io.PrintWriter;
import java.time.LocalDateTime;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/time")
public class Time extends HttpServlet {
    @Override
    protected void doGet(
            HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
        PrintWriter out = resp.getWriter();

        out.println("<!Doctype html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<meta charset='UTF-8'>");
        out.println("<title>Servlet 范例文件</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("</body>");
        out.println(LocalDateTime.now());
        out.println("</html>");
    }
}
```

先别管这个程序中有太多细节还是没见过的，目前只要注意两件事。第一件事是 Servlet 类必须继承 `HttpServlet`，第二件事就是要输出 HTML 时，必须通过 Java 的输入输出功能(在这里是从 `HttpServletResponse` 取得 `PrintWriter`)，并使用 Java 程序取得 Web 网站上的时间。

就输出结果来说，这个 Servlet 与 1.1.6 节中看到的 JSP 页面，基本上是相同的，如果是从网页设计师角度来看待这个功能的撰写，相信会选择 JSP 而不是 Servlet。

事实上，Servlet 主要是用来定义 Java 程序逻辑的，应该避免直接在 Servlet 中产生画面输出(如直接编写 HTML)。如何适当地分配 JSP 与 Servlet 的职责，需要一些经验与设计，这些在本书之后章节会有所介绍。

前面说过，JSP 网页最后还是成为 Servlet。以 1.1.6 节中的 JSP 为例，若使用 Tomcat 作为 Web 容器，最后由容器转译后的 Servlet 类别如下所示：

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import java.time.LocalDateTime;

public final class time_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent,
               org.apache.jasper.runtime.JspSourceImports {

    // 略...

  public void _jspInit() {
  }

  public void _jspDestroy() {
  }

  public void _jspService(final javax.servlet.http.HttpServletRequest request,
final javax.servlet.http.HttpServletResponse response)
      throws java.io.IOException, javax.servlet.ServletException {

    final java.lang.String _jspx_method = request.getMethod();
    if (!"GET".equals(_jspx_method) && !"POST".equals(_jspx_method) && !"HEAD".
    equals (_jspx_method) && !javax.servlet.DispatcherType.ERROR.equals(request.
    getDispatcherType())) {
      response.sendError(HttpServletResponse.SC_METHOD_NOT_ALLOWED, "JSPs only
      permit GET POST or HEAD");
      return;
    }
    // 略...
    try {
      response.setContentType("text/html; charset=UTF-8");
      pageContext = _jspxFactory.getPageContext(this, request, response,
                null, true, 8192, true);
      // 略...
      out = pageContext.getOut();
      _jspx_out = out;

      out.write("\r\n");
      out.write("\r\n");
      out.write("<!Doctype html>\r\n");
      out.write("<html>\r\n");
      out.write("    <head>\r\n");
      out.write("        <meta charset=\"UTF-8\">\r\n");
      out.write("        <title>JSP 范例文件</title>\r\n");
      out.write("    </head>\r\n");
      out.write("    <body>\r\n");
      out.write("        ");
      out.print( LocalDateTime.now() );
```

```
            out.write("\r\n");
            out.write("    </body>\r\n");
            out.write("</html>");
          } catch (java.lang.Throwable t) {
              // 略...
          } finally {
            _jspxFactory.releasePageContext(_jspx_page_context);
          }
        }
      }
```

由于篇幅限制，上例的程序代码省略了一些目前还不需要注意的细节，重点在观察这个类继承了 HttpJspBase，而 HttpJspBase 继承自 HttpServlet，HTML 的输出方式与先前所编写的 SimpleServlet 类是类似的。这个由容器转译的 Servlet 类会再进行编译并加载容器以提供服务。

许多初学 JSP 的人会遇到很多转译、编译或执行的问题，而问题通常在于不了解 JSP 转译为 Servlet 之后，对应到哪个程序段，更有人完全不知道 JSP 与 Servlet 其实是一体两面的事实，因而遇到问题就无法解决。了解 JSP 与 Servlet 的对应关系，必要时查看一下 JSP 转译为 Servlet 后的源代码，都是 JSP 网页执行遇到错误时解决问题的重要方法之一。

## 1.2.3 关于 MVC/Model 2

在 Servlet 程序中夹杂 HTML 的画面输出绝对不是什么好主意，而之后介绍到 JSP 时，你可以了解到在 JSP 中也可以编写 Java 程序代码，然而 JSP 网页中的 HTML 间夹杂 Java 程序代码，也是不建议的。Java 程序代码与呈现画面的 HTML 等混杂在一起，非但撰写不易、日后维护麻烦，对大型项目的分工合作也是一大困扰，将来若需转换为其他页面模板技术，也会遇上许多的问题。

谈及 Web 应用程序架构上的设计时，总会谈到 MVC 和 Model 2 这两个名词。MVC 是 Model、View、Controller 的缩写，这里译为模型、视图、控制器，分别代表应用程序中三种职责各不相同的对象。最原始的 MVC 模式其实是指桌面应用程序的一种设计方式，为了让同一份数据能有不同的画面呈现方式，并且当数据被变更时，画面可获得通知并根据资料更新画面呈现。通常 MVC 模式的互动示意，会使用如图 1.16 所示的方式来表现。

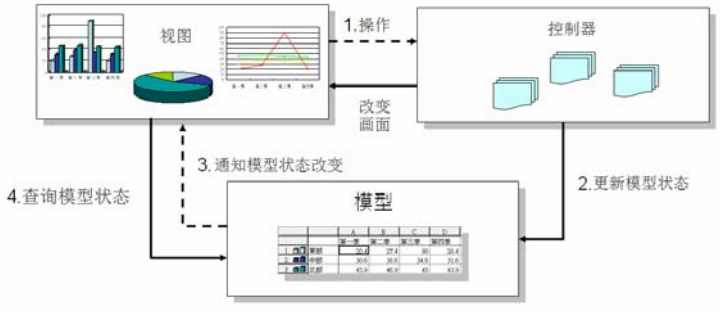



图 1.16　MVC 互动示意图

本书不是在教桌面应用程序，对于图 1.16 的 MVC 模型，你需要了解：

- 模型不会有画面相关的程序代码。
- 视图负责画面相关逻辑。
- 控制器知道某个操作必须调用哪些模型。

后来有人认为，MVC 这样的职责分配，可以套用在 Web 应用程序的设计上：

- 视图部分可由网页来实现。
- 服务器上的数据访问或业务逻辑(Business logic)由模型负责。
- 控制器接受浏览器的请求，决定调用哪些模型来处理。

然而，桌面应用程序上的 MVC 设计方式，有个与 Web 应用程序决定性的不同。先前介绍过，Web 应用程序是基于 HTTP，必须基于请求/响应模型，没有请求就不会有响应，也就是 HTTP 服务器不可能主动对浏览器发出响应，如图 1.16 所示第 3 点，基于 HTTP 是做不到的。因此，对 MVC 的行为作了变化，因而形成所谓的 Model 2 架构，如图 1.17 所示。

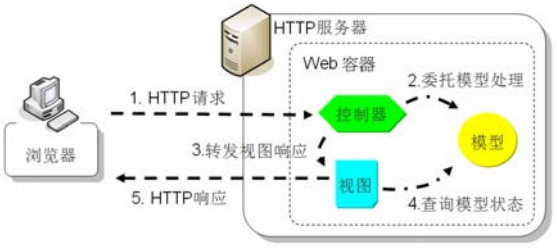



图 1.17　Model 2 架构

在 Model 2 的架构上，仍将程序职责分为模型(Model)、视图(View)、控制器(Controller)，这就是为什么有些人也称这个架构为 MVC，或并称为 MVC/Model 2，也有人直接称之为 Web MVC。在 Model 2 的架构上，控制器、模型、视图各负的职责如下。

- 控制器：取得请求参数、验证请求参数、转发请求给模型、转发请求给画面，这些都使用程序代码来实现。
- 模型：接受控制器的请求调用，负责处理业务逻辑、负责数据存取逻辑等，这部分还可依应用程序功能，产生各种不同职责的模型对象，模型使用程序代码来实现。
- 视图：接受控制器的请求调用，会从模型提取运算后的结果，根据页面逻辑呈现所需的画面，在职责分配良好的情况下，可做到不出现 Java 程序代码，因此不会发生程序代码与 HTML 混杂在一起的情况。如图 1.18 所示的 JSP 就完全没有出现 Java 程序代码。

这样的 JSP 页面，将来要转换至其他模板技术时相对来说比较容易。例如，上面的 JSP 转换为基于 Java 的 Thymeleaf 模板页面的话，如图 1.19 所示。

Model 2 在 Web 应用程序中是非常重要的模式，因为职责分配清楚，有助于团队合作，许多 Web 框架都实现了 Model 2，其应用也不仅在 Java 技术实现的 Web 应用程序。要以文字方式描述 Model 2 会比较抽象，本书后面的章节会以实际程序逐步实现 Model 2 架构，你也可通过这些内容逐步了解各个角色如何分配职责。

```
<html>
    <head>
        <meta content='text/html;charset=UTF-8' http-equiv='content-type'>
        <title>Gossip  微博 </title>
        <link rel='stylesheet' href='css/member.css' type='text/css'>
    </head>
    <body>
        <div class='leftPanel'>
            <img src='images/caterpillar.jpg' alt='Gossip 微博 />
            <br><br>
            <a href='logout?username="${ sessionScope.login }'>退出 ${ sessionScope.login }</a>
        </div>
        <form method='post' action='message'>
        分享新鲜事...<br>
       <c:if test="${requestScope.blabla != null}">信息要140字以内<br></c:if>
            <textarea cols='60' rows='4' name='blabla'>${requestScope.blabla}</textarea><br>
            <button type='submit'>发送</button>
        </form>
        <table style='text-align: left; width: 510px; height: 88px;'
                border='0' cellpadding='2' cellspacing='2'>
            <thead>
                <tr>
                    <th><hr></th>
                </tr>
            </thead>
            <tbody>
            <c:forEach var="blah" items="${requestScope.blahs}">
                <tr>
                    <td style='vertical-align: top;'>${blah.username}<br>
                        <c:out value="${blah.txt}"/><br>
                        <fmt:formatDate value="${blah.date}" type="both"
                                        dateStyle="full" timeStyle="full"/>
                        <a href='delete?message=${blah.date.time}'>删除</a>
                        <hr>
                    </td>
                </tr>
            </c:forEach>
            </tbody>
        </table>
        <hr style='width: 100%; height: 1px;'>
    </body>
</html>
```

图 1.18　没有混杂 Java 程序代码的 JSP 网页

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
    <head>
        <meta content='text/html;charset=UTF-8' http-equiv='content-type'>
        <title>Gossip  微博 </title>
        <link rel='stylesheet' href='css/member.css' type='text/css'>
    </head>
    <body>
        <div class='leftPanel'>
            <img src='images/caterpillar.jpg' alt='Gossip 微博 />
            <br><br>
            <a th:href="@{logout?username={username}(username=${session.login})}">退出</a>
        </div>
        <form method='post' action='message'>分享新鲜事...<br>
        <span th:if="${blabla != null}">信息要140字以内</span><br>
            <textarea cols='60' rows='4' name='blabla' th:text="${blabla}">Blabla...</textarea><br>
            <button type='submit'>发送</button>
        </form>
        <table style='text-align: left; width: 510px; height: 88px;'
                border='0' cellpadding='2' cellspacing='2'>
            <thead>
                <tr>
                    <th><hr></th>
                </tr>
            </thead>
            <tbody>
            <tr th:each="blah : ${blahs}">
                <td style='vertical-align: top;'>
                    <span th:text="${blah.username}">user name</span><br>
                    <span th:text="${blah.txt}">blabla</span><br>
                    <span th:text="${#dates.format(blah.date, 'dd/MMM/yyyy HH:mm')}">time here</span>
                    <a th:href="@{delete?message={time}(time=${blah.date.time})}">删除 </a>
                    <hr>
                </td>
            </tr>
            </tbody>
        </table>
        <hr style='width: 100%; height: 1px;'>
    </body>
</html>
```

图 1.19　使用 Thymeleaf 模板的页面

如果 Web 应用程序的设计符合 Model 2 模式，那么在使用支持 Model 2 的 Web 框架时，也能感受到 Web 框架的益处。本书之后会谈到 Spring MVC 框架，会将基于 Model 2 的范例程序改写为套用 Spring MVC，从实际的例子中，了解 Model 2 设计带来的优点。

## 1.2.4 Java EE 简介

时至今日，Java 这个名词不仅代表一个程序语言的名称，更代表了一个开发平台。Java 平台可以解决的领域非常庞大，主要分为三大平台：Java SE(Java Platform, Standard Edition)、Java EE(Java Platform, Enterprise Edition)与 Java ME(Java Platform, Micro Edition)。

Java SE 是初学 Java 必要的标准版本，可解决标准桌面应用程序需求，并为 Java EE 的基础，Java ME 的部分集合。Java ME 的目标则为微型装置，如手机、平板电脑等的解决方案，为 Java SE 的部分子集加上一些装置的特性集合，目前有很大部分被 Android 方案取代了。Java EE 目标是全面性解决企业可能遇到的各个领域问题的方案，Servlet/JSP 就在 Java EE 的范畴中。

无论是 Java SE、Java EE，还是 Java ME，都是业界共同订制的标准，业界代表可加入 JCP(Java Community Process)共同参与、审核、投票决定平台应有的组件、特性、应用程序编程接口等，制订出来的标准会以 JSR(Java Specification Requests)作为正式标准规范文件，不同的技术解决方案标准规范会给予一个编号。在 JSR 规范的标准之下，各厂商可以各自实现成品。所以，同样是 Web 容器，会有不同厂商的实现产品，而 JSR 通常也会提供参考实现(Reference Implementation, RI)。

在撰写本书时，Java EE 的版本为 Java EE 8，主要规范是在 JSR 366 文件之中，而 Java EE 平台中的特定技术，则在规范于特定的 JSR 文件之中，若对这些文件有兴趣，可以参考 Java EE 8 Technologies(www.oracle.com/technetwork/java/javaee/tech/)。

本书主要介绍 Servlet 4.0 规范在 JSR 369 文件，JSP 2.3 规范在 JSR 245 文件，Expression Language 3.0 规范在 JSR 341 文件，JSTL 1.2 规范在 JSR 52 文件。

JSR 文件规范了相关技术应用的功能，在阅读完本书内容之后，建议可以试着自行阅读 JSR，内容虽然有点生硬，但可以了解更多 Servlet/JSP 的相关细节。

**提示»»** 想要查询 JSR 文件，只要在“http://jcp.org/en/jsr/detail?id=”之后加上文件编号就可以了。例如，查询 JSR 369 文件，网址就是：jcp.org/en/jsr/detail?id=369。

也可以看到，本书将探讨的 Servlet/JSP，其实只是 Java EE 中 Web 容器的一个技术规范，可见整个 Java EE 体系之庞大。Servlet/JSP 在 Java EE 中，主要在接受客户端(浏览器)的请求，收集请求信息并转发后端服务对象进行处理，而处理完的信息又交由 Servlet/JSP 来对客户端进行响应。

Java EE 8 基于 Java SE 8，而 Java SE 8 是 Java 演变过程中非常重要的版本，包含了 Lambda、Stream API、新日期时间 API 等重大特性，这意味着在撰写基于 Java EE 8 的 Web 应用程序时，好处不只是有了 Servlet 4.0 的新功能，还可以运用 Java SE 8 的重大特性。例如，在运用 Lambda、Stream API 等撰写应用程序时，程序代码风格也可以有非常大的不同，例如运用函数的风格。

在 2017 年 9 月，Oracle 正式宣布 Java EE 开放原始码，后来选定将 Java EE 相关技术授权给 Eclipse 基金会，基金会也决定将 Java EE 更名为 Jakarta EE。

## 1.3 重点复习

URL 的主要目的，是以文字方式来说明互联网上的资源如何取得。URN 则代表某个资源独一无二的名称。URL 或 URN 的目的都是标识某个资源，后来制定了 URI 标准，而 URL 与 URN 成为 URI 的子集。

HTTP 是基于请求/响应的通信协议，浏览器对 Web 网站发出一个取得资源的请求，Web 网站将要求的资源响应给浏览器，没有请求就不会有响应。在 HTTP 协议之下，Web 网站是个健忘的家伙，Web 网站响应浏览器之后，就不会记得浏览器的信息，更不会去维护与浏览器有关的状态，因此 HTTP 又称为无状态的通信协议。

请求参数是在 URI 之后跟随一个问号(?)，请求参数名称与请求参数值中间以等号(=)表示成对关系。若有多个请求参数，则以&字符连接。

GET 与 POST 在使用时除了 URI 的数据长度限制、是否在地址栏上出现请求参数等表面上的功能差异之外，事实上在 HTTP 最初的设计中，该选择使用 GET 或 POST，可根据其是否为安全或等幂操作来决定。GET 应用于安全、等幂操作的请求，而 POST 应用于非等幂操作的请求。

在 URI 的规范中定义了一些保留字符，如“:”“/”“?”“&”“=”“@”“%”等字符，在 URI 中都有它们各自的作用。如果要在请求参数上表达 URI 中的保留字符，必须在%字符之后以十六进制数值表示方式，来表示该字符的八个位数值，这是 URI 规范中的百分比编码(Percent-Encoding)，也就是俗称的 URI 编码或 URL 编码。

在 URI 规范中，空格符的编码为%20，而在 HTTP 规范中空格符的编码为“+”。URI 规范的 URI 编码，针对的是字符 UTF-8 编码的八个位数值，在 HTTP 规范下的 URI 编码，并不限使用 UTF-8。

对于 Web 安全而言，想要认识基本的安全弱点从何产生，可以从 OWASP TOP 10 出发，这是由 OWASP 发起的计划之一，于 2002 年发起，针对 Web 应用程序最重大的十个弱点进行排行，首次 OWASP Top 10 于 2003 年发布，2004 年做了更新，之后每三年改版一次，就本书撰写的这个时间点，最新版的 OWASP Top 10 于 2017 年 11 月正式发布。

在学习 Servlet/JSP 时，有个重要的概念：“Web 容器是 Servlet/JSP 唯一认得的 HTTP 服务器。”如果希望用 Servlet/JSP 编写的 Web 应用程序可以正常运作，就必须知道 Servlet/JSP 如何与 Web 容器沟通，Web 容器如何管理 Servlet/JSP 的各种对象等问题。

Servlet 的执行依赖于 Web 容器提供的服务，没有容器，Servlet 只是单纯的一个 Java 类，不能称为可提供服务的 Servlet。对每个请求，容器是创建一个线程并转发给适当的 Servlet 来处理，因而可以大幅减轻性能上的负担，但也因此要注意线程安全问题。

JSP 最后终究会被容器转译为 Servlet 并加载执行，了解 JSP 与 Servlet 中各对象的对应关系是必要的，必要时可配合适当的工具，查看 JSP 转译为 Servlet 之后的源代码内容。

Java EE 是一个由厂商共同制订的标准，厂商再遵守标准来实现自己的产品。Java EE 的中心是由容器提供服务，了解容器的特性为学习 Java EE 的不二法门。Servlet/JSP 为 Java EE 中接受、转发、响应客户端请求的技术，是基于 Web 容器所提供的服务。

# 编写与设置 Servlet

Chapter 2

学习目标：

- 开发环境的准备与使用
- 了解 Web 应用程序架构
- Servlet 编写与部署设置
- 了解 URI 模式对应
- 使用 web-fragment.xml