第3章

Java Web开发环境搭建

本章学习目标

- 熟悉 Java Web 的工作原理。
- 熟悉 Java Web 开发环境搭建。
- 掌握 JSP Web 程序的开发过程。

扫一扫

视频讲解

3.1 Java Web 工作原理

Java Web 是用 Java 技术来解决互联网领域相关 Web 的技术总和，包括服务器端和客户端两部分。Java 在客户端的应用有 Java Applet，不过现在使用得很少；Java 在服务器端的应用非常丰富，如 Servlet、JSP 和第三方框架等。Java 技术对 Web 领域的发展注入了强大的动力。

Web 的工作原理是：用户使用浏览器通过 HTTP 请求服务器上的 Web 资源，服务器接收到用户发送的请求后，读取请求 URL 所标识的资源，加上消息报头发送给客户端的浏览器，浏览器解析响应中的 HTML 数据，向用户呈现多姿多彩的 HTML 页面。整个过程如图 3-1 所示。

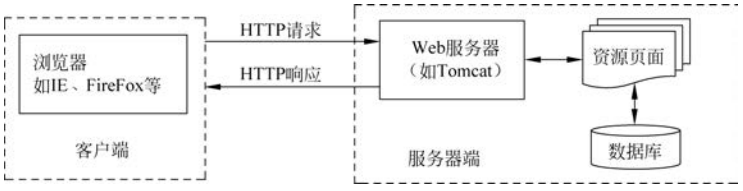


图 3-1 Java Web 工作原理

搭建 Java Web 开发运行环境所要用到的软件如表 3-1 所示。

表 3-1 搭建 Java Web 开发运行环境所要用到的软件

软　　件	作　　用
JDK	整个 Java 的核心，包括了 Java 运行环境、Java 工具和 Java 基础类库
MyEclipse	集成开发工具。用于 Java EE 的开发、发布以及应用程序服务器的整合
Tomcat	Tomcat 服务器是一个免费的开放源代码的轻量级 Web 应用服务器。在中小型系统和并发访问用户不是很多的场合下被普遍使用，是开发和调试 JSP 程序的首选
MySQL	MySQL 是一个开放源码的小型关联式数据库管理系统。由于其体积小、速度快、成本低，尤其是开放源码这一特点，被广泛地应用在 Internet 上的中小型网站中
Navicat for MySQL	Navicat 是一套快速、可靠的数据库管理工具，用来对本机或远程的 MySQL、SQL Server、SQLite、Oracle 及 PostgreSQL 数据库进行管理及开发

3.2 Tomcat 的安装配置

Tomcat 服务器是一个免费的开放源代码的轻量级 Web 应用服务器。Tomcat 是 Apache 软件基金会的 Jakarta 项目中的一个核心项目，由 Apache、Sun 和其他一些公司及个人共同开发。由于有了 Sun 公司的参与和支持，最新的 Servlet 和 JSP 规范总是能在 Tomcat 中得到体现。因为 Tomcat 技术先进、性能稳定，而且免费，因而深受 Java 爱好者的喜爱，成为目前流行的 Web 应用服务器。Tomcat 服务器在中小型系统和并发访问用户不是很多的场合下被普遍使用，是开发和调试 JSP 程序的首选。

扫一扫

视频讲解

在 Sun 公司的 Java Servlet 规范中，对 Java Web 应用做了这样的定义：Java Web 应用由一组 Servlet、HTML 页、类，以及其他可以被绑定的资源构成，可以在各种供应商提供的实现 Servlet 规范的 Web 应用容器中运行。Tomcat 就是这样一个实现了 Servlet 规范的 Servlet/JSP 容器。

在开始安装 Tomcat 之前，先准备 JDK 和 Tomcat 两个软件，如果已经安装了 JDK，只需 Tomcat 即可。

安装 JDK 后，必须配置系统的 JDK 环境变量，设置环境变量的方法如下：

(1) 右击“我的电脑”，然后依次选择“属性”→“高级”→“环境变量”→“新建”。

(2) 分别新建如下系统环境变量 JAVA_HOME 和 CLASSPATH。

① 系统环境变量：JAVA_HOME=JDK 的安装路径。

② 系统环境变量：CLASSPATH= .;%JAVA_HOME%\lib\tools.jar;%JAVA_HOME%\LIB\dt.jar。

在定义 CLASSPATH 变量时，此变量的值必须以“.;”开头，“.”代表当前目录。以上准备工作完成以后可以运行 Tomcat 的安装程序，在安装过程中按照提示选取默认值即可，当问到 JDK 时只要给出正确的 JDK 安装路径。

Tomcat 的安装步骤如下：

(1) 从 http://tomcat.apache.org/网站下载 Tomcat，如图 3-2 所示。

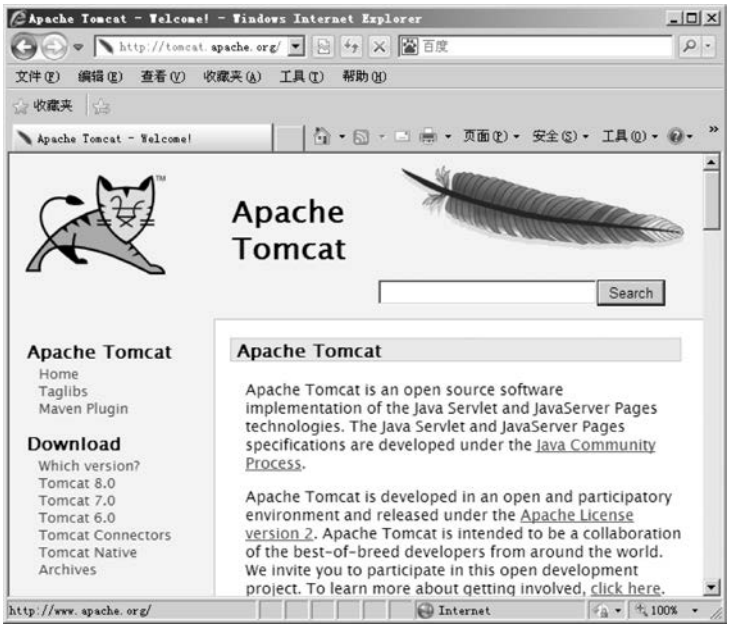


图 3-2 下载 Tomcat

（2）注意：可以下载 zip 格式或 exe 格式的 Tomcat，其中 zip 格式的只要解压缩再配置环境变量就可以使用了。这里使用的是 exe 文件，exe 文件对于新手来说比较方便。

（3）下载后的文件为 apache-tomcat-7.0.21.exe。

（4）双击安装文件，根据安装向导提示完成安装。

（5）测试，在浏览器地址栏输入 http://localhost:8080 或 http://127.0.1.1:8080，出现如图 3-3 所示的结果，则表示 Tomcat 安装成功。

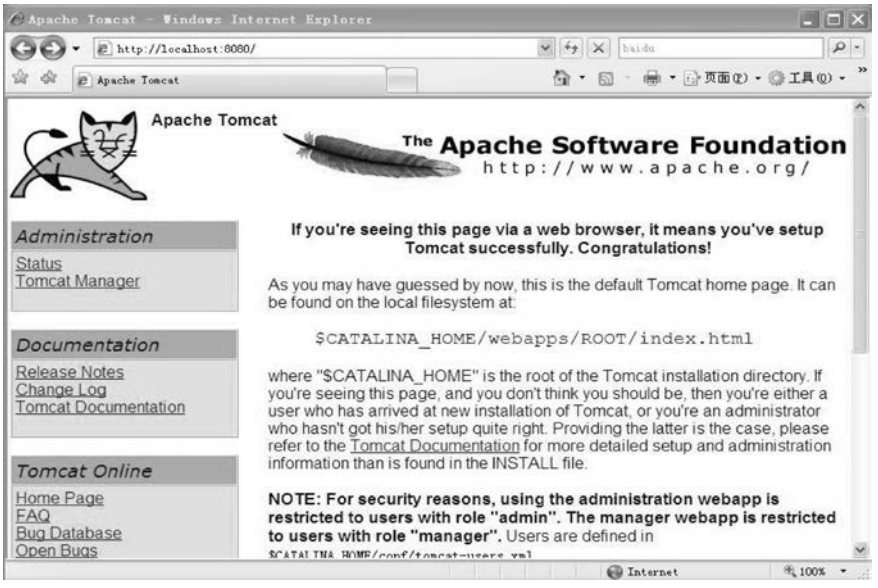


图 3-3　测试 Tomcat

扫一扫

视频讲解

3.3　在 MyEclipse 中配置 Tomcat

在 MyEclipse 6.0 以上的版本中都自带了一个 Tomcat，如果使用用户安装的 Tomcat 服务器，就需要把用户安装的 Tomcat 集成到 MyEclipse 中，以便给开发带来方便。

在 MyEclipse 中配置用户安装的 Tomcat 的方法如下：

（1）从 MyEclipse 菜单栏上的“窗口（Window）”菜单中选择“首选项（Preferences）”，打开如图 3-4 所示的窗口。

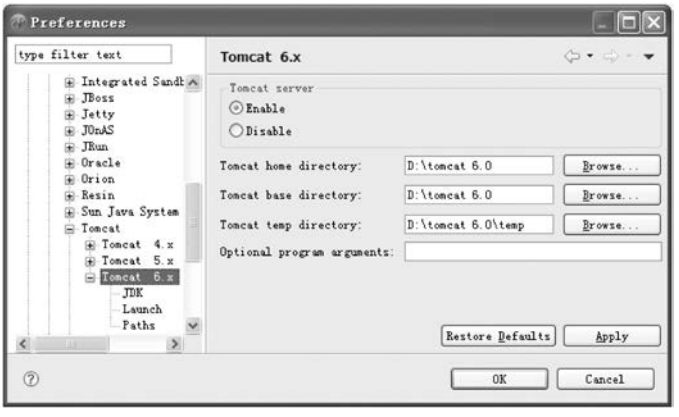


图 3-4　配置 Tomcat 服务器

（2）在左边窗口栏中选择 MyEclipse Enterprise Workbench→Servers→Tomcat→Tomcat 6.x。

（3）在右边窗口栏的 Tomcat home directory 对应的文本框中通过浏览方式填入已安装的 Tomcat 的主目录。下面两个选项的内容会自动添加进去。

（4）选择 Enable 单选按钮。

（5）在左边窗口栏中选择 JDK，在右边窗口栏中单击 Add 按钮，选择 JRE 所在的目录，如 C:\Program Files\Java\jre6。单击 OK 按钮，完成 JDK 配置。到此就可以在 MyEclipse 中启动自己安装的 Tomcat 了。

在 MyEclipse 开发环境工具栏上有三个与 Web 发布有关的图标，如表 3-2 所示。

表 3-2　工具栏中与 Web 发布有关的图标

图　标	功　能
	项目发布
	启动/停止 Web 服务器
	启动 MyEclipse 内置浏览器

- 选择“项目发布”按钮，可将指定的项目发布到选定的服务器上。
- 选取“启动/停止 Web 服务器”按钮，会打开一个下拉列表，在下拉列表中选择响应服务器中的 Start 选项，即可启动服务器。
- 选择内置浏览器，可在 IDE 环境下开启浏览窗口，浏览发布的 Web 站点。当然，也可以通过外部浏览器访问服务器。

3.4　使用 MyEclipse 创建 Web 工程

在 MyEclipse 中有很多工程模板，可以根据需要选择相应的模板。

创建 Web 工程的步骤如下：

（1）从菜单中选择“文件(File)”→“新建(New)”→“Web 项目(Web Project)”，弹出如图 3-5 所示的窗口。

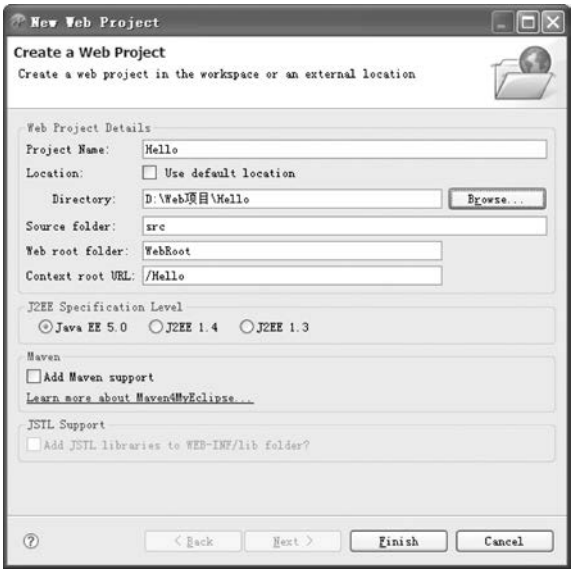


图 3-5　创建 Web 工程

（2）在窗口中填写工程的名字为 Hello，指定工程项目的保存路径，其他可选默认值，选择 Java EE 5.0 单选按钮，单击 Finish 按钮即可。这时在左边的资源管理器中可以看到如图 3-6 所示的目录结构。

从图中可以看到，src 是存放类源文件的目录；WebRoot 为虚拟路径，用于存放静态网页和动态网页的目录；WEB-INF 是受 Web 容器保护的目录，在这个目录下有一个 lib 目录，用来存放工程用到的 jar 包；web.xml 是描述符文件，是 Java Web 服务必须有的配置文件，用于配置和 Web 服务相关的一些参数，包括 Servlet、拦截器等。

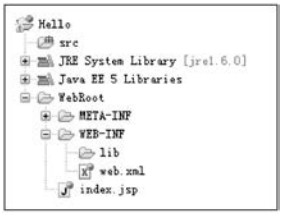


图 3-6　Web 工程的目录结构

3.5　使用 MyEclipse 发布 Web 工程

Web 工程创建完成以后，可以在 MyEclipse 中编程、调试和发布。

将刚才建立的 Hello 工程发布到服务器中的步骤如下。

（1）在工具栏中单击“项目发布”按钮，弹出如图 3-7 所示的对话框，在 Project（工程）下拉列表框中选中要发布的工程，单击 Add 按钮，弹出如图 3-8 所示的窗口。

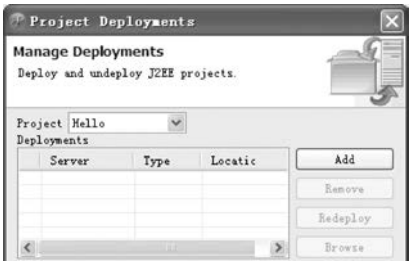


图 3-7　选择要发布的 Web 工程

图 3-8　选择要发布的 Web 服务器

（2）在 New Deployment 对话框中的 Server 下拉列表框中选择用于发布工程的 Web 服务器，如果没有安装自定义的 Web 服务器，则只有一个 MyEclipse 自带的服务器。单击 Finish 按钮，出现如图 3-9 所示的对话框。单击 OK 按钮，即完成 Web 工程的发布。

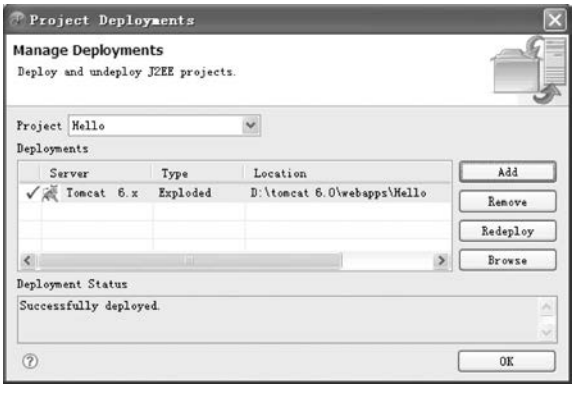


图 3-9　Web 工程的发布

（3）完成发布后，可以从如图 3-10 所示的工具栏中启动 Web 服务器，也可以从“开始”菜单中选择 Monitor Tomcat，在任务栏上启动 Tomcat。

服务器启动成功后，打开浏览器，在地址栏输入 http://localhost:8080/Hello，出现如

图 3-11 所示的画面，说明工程发布成功。

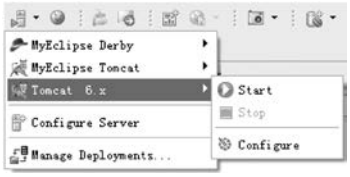


图 3-10　启动 Web 服务器

图 3-11　访问服务器中的 Hello 应用程序

3.6　Tomcat 的其他常用设置

以下介绍 Tomcat 的其他常用设置，仅供提高服务器安全性能做参考。有些服务器在部署时未考虑这些因素，学习时切不可对正常运行的服务器进行“试验”，以免产生不良后果。

1. Tomcat 管理员用户名和密码的设置

进入 Tomcat 安装目录的 conf 文件夹中，在 tomcat-users.xml 中设置用户 admin 的密码，密码要复杂。

原来的 tomcat-user.xml 是：

```
<?xml version = '1.0' encoding = 'utf - 8'?>
< tomcat - users >
  < role rolename = "manager"/>
  < role rolename = "admin"/>
  < user username = "admin" password = "" roles = "admin, manager"/>
</tomcat - users >
```

语句< user username＝"admin" password＝"" roles＝"admin，manager"/>表示用户 admin 的密码为空，拥有 admin 和 manager 的权限。manager 和 admin 是有特权的角色，如果想登录 tomcat manager，则必须添加 manager 角色；如果要进入 status，则必须添加 admin 角色。

2. 处理 webapps 目录下的管理文件夹

为了安全起见，建议删除 webapps 目录下的 ROOT、manager、host-manager 和 docs 四个目录。也可以不删除，而将其改名。

3. 禁用目录访问设置

默认安装 Tomcat 时，如果目录没有 index.jsp，会默认把目录文件列出来，如图 3-12 所示，此时存在安全隐患。

为了禁止对 Tomcat 目录的访问，可以在 Tomcat 的服务中进行如下设置。

(1) 打开 Tomcat 安装目录\conf\web.xml，找到 init-param 节点：

```
< init - param >
    < param - name > listings </param - name >
    < param - value > false </param - value >
</init - param >
```

(2) 将 listings 参数设置为 false。修改后，禁用目录访问有效，如图 3-13 所示。

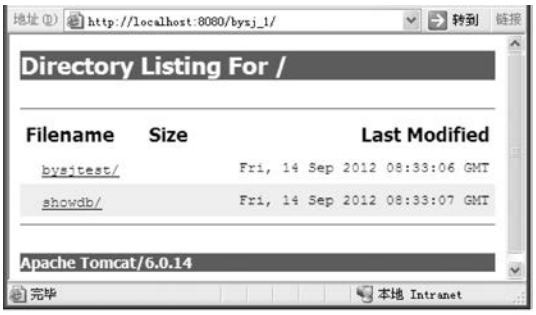


图 3-12　Tomcat 默认安装时的目录，存在安全隐患

图 3-13　Tomcat 禁用目录访问效果

4. 禁用管理控制台设置

默认情况下，可以登录 http://localhost:8080/，单击主页中的 Tomcat Manager 按钮，输入 admin，默认密码为空，即可进入 Tomcat 管理控制台，出现如图 3-14 所示的目录管理界面。这将造成严重安全问题，而解决这一问题的办法是修改{ $tomcat_home}/conf/目录下 tomcat-users.xml 文件中的用户密码。

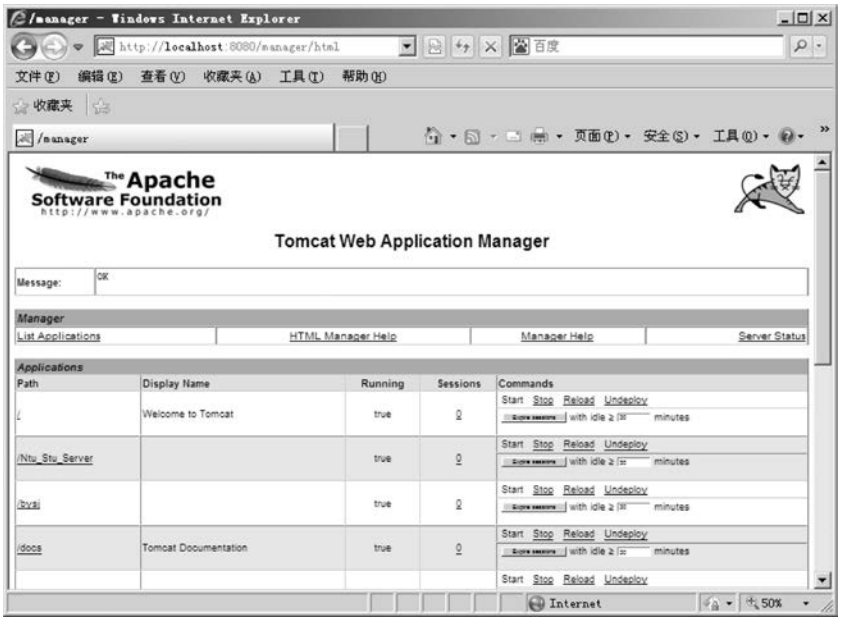


图 3-14　用户进入 Tomcat 目录管理界面

5. 禁用远程 shutdown 命令

默认情况下，{ $tomcat_home}/conf/目录下的 server.xml 文件中有下面一行配置：

```
<Server port="8005" shutdown="SHUTDOWN">
```

允许任何人只要使用 telnet 命令查看服务器的 8005 端口，输入 SHUTDOWN，按 Enter 键，服务器立即被关掉。从安全的角度上考虑，需要把这个 SHUTDOWN 指令改成一个别人不容易猜测的其他字符串，也可禁用 8005 端口。

6. 修改 Tomcat 默认服务器端口 8080

在实际工程中，通常将 Tomcat Web 服务的默认端口 8080 改成 80，这样用户在访问站点时就不必在地址栏中加入端口号 8080 了，因为 HTTP 请求地址中默认的端口号即为 80 端

口。设置 Tomcat Web 服务的端口的方法是修改{ $ tomcat_home}/conf/目录下 server. xml 文件,将下列语句:

```
< Connector port = "8080" protocol = "HTTP/1.1"
```

中的端口号 8080 改为 80,修改后的语句为:

```
< Connector port = "80" protocol = "HTTP/1.1"
```

这样,在访问 Tomcat Web 站点时,就不必加 8080 端口号了,如输入 http://localhost/即可。

7. 修改 Tomcat 默认可以使用的内存大小

Tomcat 默认可以使用的内存为 128MB,在较大型的应用项目中,这些内存是不够的,需要调大,修改方法如下。

对于 Windows 系统,在文件{tomcat_home}/bin/catalina. bat 的前面,增加如下设置:

JAVA_OPTS='-Xms[初始化内存大小] -Xmx[可以使用的最大内存]',可把这两个参数值调大。

例如:

```
JAVA_OPTS = ' - Xms256m  - Xmx512m'
```

表示初始化内存为 256MB,可以使用的最大内存为 512MB。

对于 UNIX 系统,在文件{tomcat_home}/bin/catalina. sh 的相应位置增加上述设置。

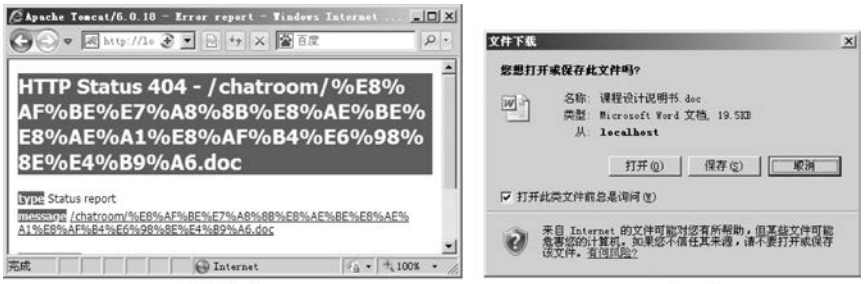
8. 让 Tomcat 支持中文文件名

一般情况下 Tomcat 不支持中文文件名,如超链接中下载含有中文名字的文件时则会出现文件名乱码。解决方法是,修改 Tomcat 的 server. xml 文件,在< Connector port="8080"…>标记中添加语句 URIEncoding="UTF-8",即可识别中文文件名。而且不只在 JSP 页面中显示中文文件名,还可以提示是打开还是下载。

例如:

```
< a href = 'http://localhost:8080/ipnet/课程设计说明书. doc'>课程设计说明书. doc'</a >
```

图 3-15(a)所示为没有添加 URIEncoding="UTF-8"语句时,单击下载出现的中文文件名乱码现象。图 3-15(b)所示为添加了 URIEncoding="UTF-8"语句后的正常情况。



(a) 错误造成乱码　　(b) 正常下载

图 3-15　Tomcat 支持中文文件名效果对比

其他的可配置文件为{tomcat_home}/bin/路径下的 web. xml、server. xml 等,如需配置,请参考有关说明。

3.7 Servlet 容器介绍

Servlet 是 Java Web 技术的核心基础。Servlet 是一种运行在支持 Java 语言的服务器上的组件，与普通 Java 类的区别是必须运行在服务器中。使用 Servlet 可以实现很多网络服务功能，为网络客户提供安全可靠的易于移植的动态网页。由于 Java 语言的平台无关性，加之 Servlet 运行在服务器端，所以对于网络用户，Servlet 的运行是完全透明的。

Servlet 容器的作用是处理客户端的请求，并将处理结果返回给客户端。在 Servlet 容器中，当用户请求到来时，Servlet 容器获取请求，然后调用某个 Servlet，并把 Servlet 的执行结果返回给用户。Tomcat 就是这样的一个 Servlet 容器。

在没有 JSP 之前，就已经出现了 Servlet 技术，JSP 是 Servlet 的扩展。Servlet 是利用输出流动态生成 HTML 页面，包括每一个 HTML 标记和每个在 HTML 页面中出现的内容。

在 JSP 出现之前，由于包括大量的 HTML 标记、静态文本及格式等，导致 Servlet 的开发效率极低。所有的表现逻辑，包括布局、色彩及图像等，都必须耦合在 Java 代码中。JSP 的出现弥补了这种不足，JSP 通过在标准的 HTML 页面中插入 Java 代码，其静态的部分无须 Java 程序控制，只有那些需要从数据库读取并根据程序动态生成信息时，才使用 Java 脚本控制。

从表面上看，JSP 页面已经不再需要 Java 类，似乎完全脱离了 Java 面向对象的特征。事实上，JSP 是 Servlet 的一种特殊形式，每个 JSP 页面就是一个 Servlet 实例，JSP 页面由系统编译成 Servlet，Servlet 再负责响应用户请求。JSP 其实也是 Servlet 的一种简化，使用 JSP 时，其实还是使用 Servlet，因为 Web 应用中的每个 JSP 页面都会由 Servlet 容器生成对应的 Servlet。

对于 Tomcat 而言，JSP 页面生成的 Servlet 放在 work 路径对应的 Web 应用下。

在项目路径 jspweb 项目\WebRoot\ch03 中创建一个简单的 JSP 页面 test.jsp 如下。

程序(\jspweb 项目\WebRoot\ch03\test.jsp)的清单：

```
<!-- 表明此为一个 JSP 页面 -->
    <%@ page contentType = "text/html; charset = gb2312" language = "java" %>
    <!DOCTYPE HTML PUBLIC " - //W3C//DTD HTML 4.0 Transitional//EN">
    <html>
    <head>
        <title>第一个 JSP 页面</title>
    </head>
    <body>
        <!-- 下面是 Java 脚本 -->
           <%
              for(int i = 0 ; i < 10; i++)
                {
                    out.print(i);
                 }
             %>
    </body>
</html>
```

程序运行结果显示：

```
0 1 2 3 4 5 6 7 8 9
```

可以在 Tomcat 的\work\Catalina\localhost\jsptest\org\apache\jsp 目录下找到文件 test_jsp.java 和 test_jsp.class。这两个文件都是 Tomcat 根据 JSP 页面自动生成的 Java

Servlet 文件及 class 文件。

下面对 test_jsp.java 文件的源代码进行分析。这是一个特殊的 Java 类，继承自 HttpJspBase 类，而 HttpJspBase 类正是 HttpServlet 的子类。

```
//JSP 页面经过 Tomcat 编译后默认的包
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
//继承 HttpJspBase 类,该类其实是 HttpServlet 的子类
public final class test_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {
  private static final JspFactory _jspxFactory =
  JspFactory.getDefaultFactory();
  private static java.util.List _jspx_dependants;
  private javax.el.ExpressionFactory _el_expressionfactory;
  private org.apache.AnnotationProcessor _jsp_annotationprocessor;
  public Object getDependants() {
    return _jspx_dependants;
  }
  public void _jspInit() {
  _el_expressionfactory = _jspxFactory.getJspApplicationContext(getServletConfig().
  getServletContext()).getExpressionFactory();
  _jsp_annotationprocessor = (org.apache.AnnotationProcessor)
  getServletConfig().getServletContext().getAttribute(org.apache.AnnotationProcessor.
  class.getName());
  }
  public void _jspDestroy() {
  }
//用于响应用户的方法
public void _jspService(HttpServletRequest request, HttpServletResponse response)
        throws java.io.IOException, ServletException {
    PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    //获得页面输出流 out
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;
    PageContext _jspx_page_context = null;
    //开始生成响应
try {
      //设置输出的页面格式
      response.setContentType("text/html; charset=gb2312");
      pageContext = _jspxFactory.getPageContext(this, request, response,
                  null, true, 8192, true);
      _jspx_page_context = pageContext;
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      //页面输出流
      out = pageContext.getOut();
      _jspx_out = out;
      //输出流,开始输出页面文档
      out.write("<!-- 表明此为一个 JSP 页面 -->\r\n");
      out.write(" \r\n");
      //下面输出 HTML 标签
```

```
        out.write("<!DOCTYPE HTML PUBLIC \" - //W3C//DTD HTML 4.0 Transitional//EN\">\r\n");
        out.write(" <HTML>\r\n");
        out.write(" <HEAD>\r\n");
        out.write(" <TITLE>第一个 JSP 页面</TITLE>\r\n");
        out.write(" </HEAD>\r\n");
        out.write(" <BODY>\r\n");
        out.write(" <!-- 下面是 Java 脚本 -->\r\n");
        out.write(" ");
        //页面中的循环,在此处循环输出
        for(int i = 0 ; i < 10; i++)
         { out.print(i); }
           out.write("\r\n");
           out.write(" </BODY>\r\n");
           out.write(" </HTML>\r\n");
    } catch (Throwable t) {
          if (!(t instanceof SkipPageException)){
             out = _jspx_out;
             if (out != null && out.getBufferSize() != 0)
               try { out.clearBuffer();
                     } catch (java.io.IOException e) {}
             if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
          }
      } finally { _jspxFactory.releasePageContext(_jspx_page_context);
     }
     }
    }
```

当用户请求某个资源时，Servlet 容器使用 ServletRequest 对象把用户的请求信息封装起来，然后调用 Servlet 生命周期中的一些方法，完成客户端的请求任务。对于每一个 JSP 程序，当第一次被用户请求时，服务器都会自动将其转换为对应的 Servlet 程序。JSP 页面中的每个字符都由对应的 Servlet 程序中的输出流生成。容器将 Servlet 执行的结果封装在 ServletResponse 对象中，以此返回客户端，完成服务过程。Servlet 容器的作用如图 3-16 所示。

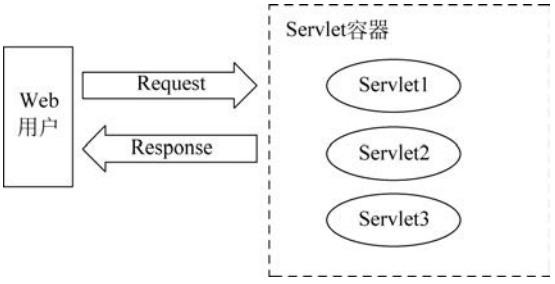


图 3-16　Servlet 容器的作用

由此可以得到如下结论：JSP 文件必须在 JSP 服务器内运行；JSP 文件必须生成 Servlet 才能执行；每个 JSP 页面的第一个访问者速度很慢，因为必须等待 JSP 编译成 Servlet；JSP 页面的访问者除了浏览器，无须安装任何客户端，甚至不需要可以运行 Java 的运行环境，因为 JSP 页面输送到客户端的是标准 HTML 页面。

关于 Servlet 程序的设计将在第 7 章中详细介绍。

3.8　HTTP 分析

HTTP 是 Web 浏览器与 Web 服务器之间交互的一种规则。目前使用的 HTTP 版本是 HTTP 1.1。HTTP 遵循请求(request)/应答(response)模型。Web 浏览器向 Web 服务器发

送请求，Web 服务器处理请求并返回适当的应答。所有 HTTP 连接都被构造成一套请求和应答。

HTTP 是一种无状态的协议，无状态是指 Web 浏览器和 Web 服务器之间不需要建立持久的连接，这意味着当一个客户端向服务器端发出请求，Web 服务器返回响应，然后连接就被关闭了，在服务器端不保留连接的有关信息。

1. HTTP 通信机制

当在浏览器地址栏中输入 http://www.baidu.com，然后按 Enter 键后，可以看到已经打开了对应的网页，下面分析客户端和服务器端是如何通信的。

HTTP 通信机制是在一次完整的 HTTP 通信过程中，Web 浏览器与 Web 服务器之间将完成下列步骤。

1）服务器自动解析 URL

HTTP URL 包含了用于查找某个资源的足够信息，基本格式如下：

```
http://host[":"port][abs_path]
```

其中，http 表示采用 HTTP 来定位网络资源；host 表示合法的主机域名或 IP 地址；port 指定一个端口号，默认值为 80；abs_path 指定请求资源的 URL。如果 URL 中没有给出 abs_path，那么当它作为请求 URL 时，必须以“/”的形式给出，通常浏览器会自动完成这个工作。例如，输入 www.163.com，浏览器会自动转换成 http://www.163.com/。

2）获取 IP 地址，建立 TCP 连接

在浏览器地址栏中输入 http://www.xxx.com/并提交之后，首先它会在 DNS 本地缓存表中查找，如果有则直接告诉 IP 地址；如果没有则要求网关 DNS 进行查找，找到对应的 IP 地址后，则返回给浏览器。当获取 IP 地址之后，就通过 TCP 与 Web 服务器建立连接，默认的 TCP 连接端口号是 80。TCP 与 IP 共同构建 Internet，即著名的 TCP/IP 协议族，因此 Internet 又被称作 TCP/IP 网络。

3）Web 浏览器向 Web 服务器发送请求命令和请求头信息

一旦建立了 TCP 连接，Web 浏览器就向 Web 服务器发出 HTTP 请求。HTTP 是比 TCP 更高层次的应用层协议，根据规则，只有低层协议建立之后才能进行更高层协议的连接。

浏览器发送其请求命令之后，还要以头信息的形式向 Web 服务器发送一些别的信息，之后浏览器发送一个空白行来通知服务器，已经结束了该头信息的发送。

4）Web 服务器应答

浏览器向服务器发出请求后，服务器会向浏览器回送如下应答：

```
HTTP/1.1 200 OK
```

应答的第一部分是协议的版本号和应答状态码，如客户端会随同请求发送关于自身的信息一样，服务器也会随同应答向用户发送关于它自己的数据及被请求的文档。

5）Web 服务器向浏览器发送数据

Web 服务器向浏览器发送头信息后，会发送一个空白行来表示头信息的发送到此结束，接着以 Content-Type 应答头信息所描述的格式发送用户所请求的实际数据。

6）Web 服务器关闭 TCP 连接

一般情况下，一旦 Web 服务器向浏览器发送了请求数据，它就要关闭 TCP 连接。但是，如果浏览器或服务器在其头信息加入了代码 Connection:keep-alive，则 TCP 连接在发送后将

仍然保持打开状态，浏览器可以继续通过相同的连接发送请求。保持连接节省了为每个请求建立新连接所需的时间，还节约了网络带宽。

HTTP 使用的默认端口号是 80。在访问 Web 服务器时，如果这个服务器的端口号是 80，则可不需指定端口号也能进行访问。例如，访问新浪网可以输入网址 http://www.sina.com.cn/，也可以输入 http://www.sina.com.cn:80。但如果服务器的端口号不是 80，则必须指定端口号。例如，访问 Tomcat 服务器就必须指定端口号 8080。

HTTP 向服务器提交请求通常有两种方式，一种是 GET 方法，另一种是 POST 方法。

下面分析 HTTP 请求与响应信息格式。

2. HTTP 请求信息格式

当浏览器向 Web 服务器发出请求时，向服务器传递了一个数据块，也就是 HTTP 请求信息，HTTP 请求信息由三部分组成，分别是请求行（一个）、消息报头（N 个）、请求正文。

其中，消息报头和请求正文是可选的，消息报头和请求正文直接用空行隔开，空行代表消息报头结束。

请求行以一个方法符号开头，以空格分开，后面跟着请求的 URL 和协议的版本，格式如下：

```
请求方式　统一资源定位符　HTTP 版本号<CRLF>
```

例如：

```
GET /web/index.html HTTP/1.1 <CRLF>
```

CRLF 表示回车和换行，除了作为结尾的 CRLF 外，不允许出现单独的 CR 或 LF 字符。

在 MyEclipse 中可以通过 TCP/IP Monitor 窗口查看 HTTP 的请求和响应过程。

要使用监听器，首先要配置监听器，打开“首选项”窗口，如图 3-17 所示。单击右侧的 Add 按钮，弹出如图 3-18 所示的对话框。

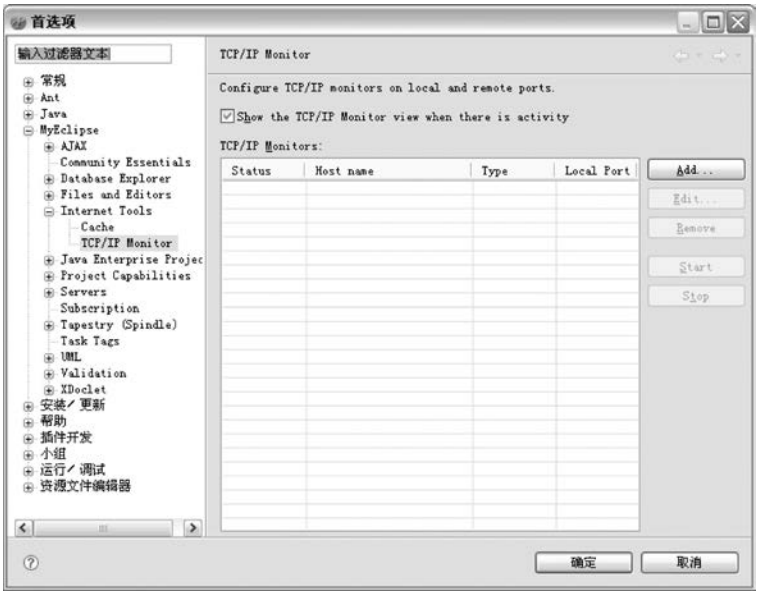


图 3-17 “首选项”窗口

在图 3-18 中单击 OK 按钮建立了一个监听器。在首选项对话框中选中刚建的监听器，同时单击右侧的 Start 按钮即可启动监听器，可通过对本地监听端口（本例中监听端口号设为 8088）对用户的请求与响应进行监听。

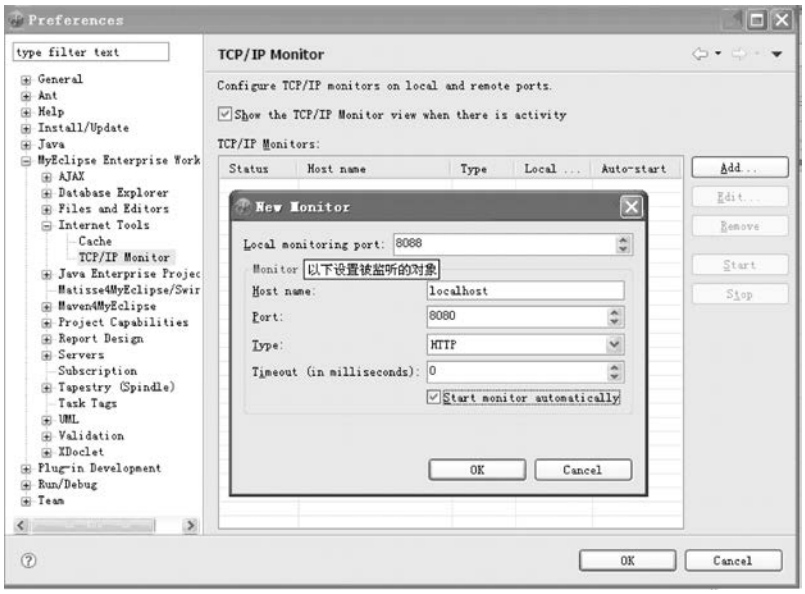


图 3-18　New Monitor 对话框

注意：访问服务器时必须先把请求(request)发到监听器设定的本地监听端口(local monitoring port)，这样，监听器就能知道请求是什么，然后监听器会把请求转到目标主机(这里是 localhost)的 8080 端口。配置的监听器端口 8088，相当于在客户端(浏览器)与应用程序服务 8080 端口之间设置了一个代理端口 8088(对应于图 3-18 配置)，所有请求服务返回的数据都会被 8088 端口获取，监听器代理关系如图 3-19 所示。

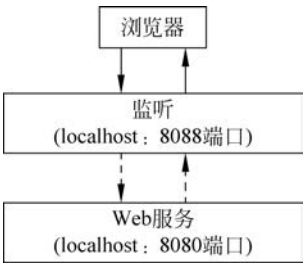


图 3-19　监听器代理关系图

下面是一个 HTTP 请求实例，新建 Web 项目，在 Web 项目中新建 index.html 文件。index.html 代码如下：

```
<html>
   <body>
   <form action="check.jsp" method="get">
      用户名:<input type="text" name="name"/>
      密码:  <input type="text" name="password"/>
             <input type="submit" value="提交"/>
   </form>
  </body>
</html>
```

要查看 TCP/IP 消息头，必须勾选 TCP/IP Monitor 窗口右侧 View Menu 中的 Show Header 选项。

在浏览器地址栏中输入 http://localhost:8088/web/index.html 后，从 TCP/IP Monitor 窗口可以看到请求和响应的信息(注意，请求的端口应输入监听端口 8088)，如图 3-20 所示。

下面是获取到的浏览器生成的 HTTP 请求信息数据包，内容如下：

```
1 GET /web/index.html HTTP/1.1 <CR>
2 Accept: */*<CR>
3 Accept-Language: zh-cn<CR>
4 Accept-Encoding: gzip, deflate<CR>
```

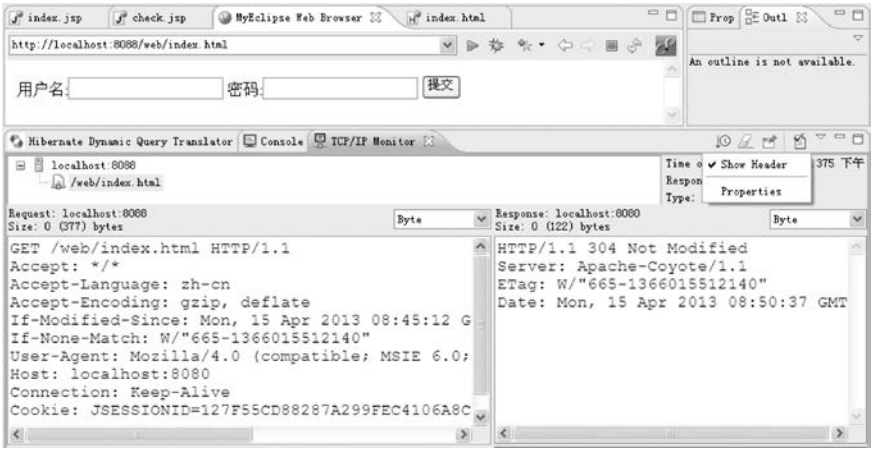


图 3-20　TCP/IP Monitor 窗口显示结果

```
5 If - Modified - Since: Mon, 15 Apr 2013 08:45:12 GMT < CR >
6 If - None - Match: W/"665 - 1366015512140"< CR >
7 User - Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727) < CR >
8 Host: localhost:8080 < CR >
9 Connection: Keep - Alive < CR >
10 Cookie: JSESSIONID = 127F55CD88287A299FEC4106A8C3AB38 < CR >
< CR >
```

为了显示清楚，把所有的回车处都加上了< CR >，注意：最后还有一个空行加一个回车，这个空行正是 HTTP 规定的消息报头和请求正文的分界线，第一个空行以下的内容就是消息体，这个请求数据包是没有消息体的。

HTTP 请求头标由关键字/值对组成，每行一对，关键字和值用冒号(:)分隔，请求头标通知服务器有关于客户端的功能和标识。

请求头标中的 GET 标识表示所使用的 HTTP 请求方式，其他常用的请求方式还有 POST。GET 方式请求的消息没有消息体，而 POST 方式请求的消息是有消息体的，请求正文的内容就是要 POST 的数据。后面/web/index.html 就是要请求的资源，HTTP 1.1 表示使用的是 HTTP 1.1。HTTP 规范定义了 8 种可能的请求方法，如表 3-3 所示。

表 3-3　HTTP 规范定义了 8 种可能的请求方法

请 求 方 法	意　　义
GET	检索 URL 中标识资源的一个简单请求
POST	服务器接受被写入客户端输出流中的数据的请求
HEAD	与 GET 方法相同，服务器只返回状态行和头标，并不返回请求文档
PUT	服务器保存请求数据作为指定 URL 新内容的请求
DELETE	服务器删除 URL 中命名的资源的请求
OPTIONS	关于服务器支持的请求方法信息的请求
TRACE	Web 服务器反馈 HTTP 请求和其头标的请求
CONNECT	已文档化但当前未实现的一个方法，预留做隧道处理

其他典型的请求头标如下。

- User-Agent：客户端厂家和版本。
- Accept：客户端可识别的内容类型列表。
- Content-Length：附加到请求的数据字节数。

最后一个请求头标之后是一个空行，发送回车符和退行，通知服务器以下不再有头标。

使用 POST 传送数据，最常使用的是 Content-Type 和 Content-Length 头标。

上述代码中的第 2 行表示所用的浏览器能接受的 Content-type；第 3～4 行则是语言和编码信息；第 5～7 行显示本机的相关信息，包括浏览器类型、操作系统信息等，很多网站可以显示使用的浏览器和操作系统版本，就是因为可以从中获取到这些信息；第 8 行表示所请求的主机和端口；第 9 行表示使用 Keep-Alive 方式，即数据传递完并不立即关闭连接；第 10 行表示请求信息中带有保存于 Cookie 中的 JSESSIONID，第一次的请求中没有该项信息。

Web 服务器端解析请求，定位指定资源，将资源副本写至套接字，返回 HTTP 响应，由客户端读取。

3. HTTP 响应消息格式

服务器 HTTP 响应消息由三部分组成，分别是状态行（一个）、响应消息报头（N 个）、响应正文数据。

（1）状态行格式如下：

```
HTTP 版本号　状态码　原因叙述< CRLF >
```

例如：

```
HTTP/1.1 200 OK
```

状态码由三位数字组成，第一位数字定义了响应的类别，且有如下 5 种可能取值。

- 1xx：指示信息——请求已接收，继续处理。
- 2xx：成功——请求已被成功接收、理解、接受。
- 3xx：重定向——要完成请求必须进行更进一步的操作。
- 4xx：客户端错误——请求有语法错误或请求无法实现。
- 5xx：服务器端错误——服务器未能实现合法的请求。

常见状态码、状态描述、说明如下：

```
200 OK                  //客户端请求成功
400 Bad Request         //客户端请求有语法错误,不能被服务器所理解
401 Unauthorized        //请求未经授权,这个状态码必须和 WWW - Authenticate 报头域一起使用
403 Forbidden           //服务器收到请求,但是拒绝提供服务
404 Not Found           //请求资源不存在,如输入了错误的 URL
500 Internal Server Error    //服务器发生不可预期的错误
503 Server Unavailable       //服务器当前不能处理客户端的请求,一段时间后可能恢复正常
```

（2）响应消息头标：像请求头标一样，指出服务器的功能，标识响应正文数据的细节。最后一个响应消息头标之后是一个空行，发送回车符和退行，表明服务器以下不再有头标。

（3）响应正文数据：HTML 文档和图像等，也就是 HTML 本身。

（4）HTTP 服务器关闭无状态的连接，浏览器解析响应。

无状态连接模型是表明在处理一个请求时，Web 服务器并不记住来自同一客户端的请求。

① 浏览器首先解析状态行，查看表明请求是否成功的状态码。

② 解析每一个响应消息头标，响应消息头标告知以下为若干字节的 HTML。

③ 读取响应正文数据 HTML，根据 HTML 的语法和语义对其进行格式化，并在浏览器窗口中显示它。

④ 一个 HTML 文档可能包含其他需要被载入的资源引用，浏览器识别这些引用，对其他

的资源再进行额外的请求，此过程循环多次。

（5）HTTP 响应消息实例分析。

当浏览器发出请求后，服务器返回响应，服务器在直接返回响应信息之前，还需要加上 HTTP 消息头。

```
HTTP /1.1 200 OK
Date: Apr 11 2006 15:32:08 GMT
Server: Apache/2.0.46(win32)
Content - Length: 119
Content - Type: text/html

< html >
   < body >
   < form action = "check. jsp" method = "get">
    用户名:< input type = "text" name = "name"/>
    密码:< input type = "text" name = "password"/>
   < input type = "submit" value = "提交"/>
   </form >
  </body >
</html >
```

可以看到，这条响应消息也是用空行划分成消息头和消息体两部分，消息体的部分正是前面写好的 HTML 代码。

消息头中 HTTP 1.1 也表示所使用的协议，200 OK 是 HTTP 返回代码，200 表示操作成功，还有其他常见的状态码，如 404 表示对象未找到，500 表示服务器错误，403 表示不能浏览目录，304 表示客户端原来缓冲的文档还可以继续使用等。

如果将 index. html 页面中的请求方式改为 POST，当再次提交 index. html 页面时，在 TCP/IP Monitor 窗口中会看到不同的请求信息。

POST 提交时在请求信息头的第一行看不到提交的数据，只有服务器的地址。如果不希望在地址栏看到客户端提交的数据就要采用 POST 提交，如一些比较敏感的网站、网上银行、电子商务网站，都采用这种方式。GET 和 POST 提交还有一个不同的地方：GET 对数据长度有限制，而 POST 对提交数据长度无限制。

习题 3

1. 怎样理解 HTTP 是无状态协议？HTTP 默认端口号是多少？
2. 通过 HTTP 向服务器端提交请求有哪两种方式？它们有什么区别？
3. 在服务器端返回的信息头中，状态码 200 和 404 分别表示什么含义？
4. Tomcat 由哪几个组件组成？它们的关系如何？
5. 在 MyEclipse 中如何配置和启动 Tomcat？
6. 简述 Java Web 的目录结构。