Ajax 开发详解
（试读 20060302）

作者：柯自聪
联系电邮：eamoi@163.com（技术） zcke0728@hotmail.com （版权）
博客：http://www.blogjava.net/eamoi/

另注：本书近期将由电子工业出版社出版。正式书名另定。

第 2 章 B/S 请求响应机制与 Web 开发模式

B/S 请求响应机制是 Web 系统所使用的服务方式，也是如今最为流行的体系结构模式，受到了广大开发人员和客户的认同。B/S 请求响应机制所使用的 HTTP 协议决定其特有的工作机制以及 Web 开发模式，也决定了其自身的缺陷。

本章在简要介绍 HTTP 协议的基础上，着重讲述 B/S 请求响应机制的工作原理、当前几种主流 Web 开发技术的开发模式，并且搭建一个后续章节开发测试需要的开发环境。

Ajax 不是一项新的技术，只是对现有技术的整合和对 B/S 请求响应机制的有效补充和改善。了解 B/S 请求响应机制以及响应的 Web 开发模式将有助于理解 Ajax 在 Web 系统中所处的位置，更好的认识 Ajax 的作用。

2.1 HTTP 请求响应模型

1、HTTP 协议

Internet 的基本协议是 TCP/IP 协议（传输控制协议和网际协议），目前广泛使用的 FTP、HTTP（超文本传输协议，Hypertext Transfer Protocol）、Archie Gopher 都是建立在 TCP/IP 上面的应用层协议，不同的协议对应不同的应用。而 HTTP 协议是 Web 应用所使用的主要协议。

HTTP 协议是基于请求响应模式的。客户端向服务器发送一个请求，请求头包含请求的方法、URI、协议版本、以及包含请求修饰符、客户端信息和内容的类似 MIME 的消息结果。服务器则以一个状态行作为响应，相应的内容包括消息协议的版本、成功或者错误编码加上包含服务器信息、实体元信息以及可能的实体内容。

HTTP 是无状态协议，依赖于瞬间或者近乎瞬间的请求处理。请求信息被立即发送，理想的情况是没有延时的进行处理，不过，延时还是客观存在的。HTTP 有一种内置的机制，在消息的传递时间上有一定的灵活性：超时机制。一个超时就是客户机等待请求消息的返回信息的最长时间。

HTTP 的请求和响应消息如果没有发送并传递成功 的话，不保存任何已传递的信息。比如，单击“ 提交 ”按钮，如果表单没有发送出去，则浏览器将显示错误信息页，并且返回空白表单。虽然没有发送成功，但是 HTTP 不保存表单信息。

由于 HTTP 协议的上述特点，通常，客户端每次需要更新信息都必须重新向服务器发起请求，客户端收到服务器返回的信息后再更新屏幕内容。

基于 HTTP 协议的客户端/服务器请求响应机制的信息交换过程包括四个步骤：

- 建立连接：客户端与服务器建立 TCP 连接；
- 发送请求：打开一个连接后，客户端把请求消息送到服务器的相应端口上，完成请求动作提交；
- 发送响应：服务器在处理完客户端请求之后，要向客户端发送响应消息；
- 关闭连接：客户端和服务器双方都可以通过关闭套接字来结束 TCP/IP 对话。

HTTP 的工作机制是请求消息和响应消息，最简单的情况，一个用户输入一个站点地址，发送一个请求。之后，浏览器返回所请求的页面，这个页面可能是简单的 HTML 页面，也可能是动态编译后的页面。如果这个页面有错误或者不存在，则 Web 服务器将发送错误信

息页面。

Web 服务器发送错误信息页是因为 HTTP 没有内置的处理机制，是无状态的，传输协议不记忆从一条请求消息到另一条请求消息的任何信息。这个特点可以报纸 Web 的一致性。但是，用户常常需要记忆一些设置内容或者浏览过程，这就需要在 Web 页面或者 URL 中携带各种参数及其值。HTTP 请求有多种样式，其中常用 的有 Post、Get、Head。

2、Get 请求

Get 请求返回以 URL 形式表示的资源，当用户输入一个简单的 URL 时，就是使用 Get 请求。Get 请求可以发送 Query String（就是在 URL 后用？附加一个参数列表），代表 URL 编码字符串的实际意义。

3、Post 请求

Post 请求则将表单体置入 Web 服务器中，发送消息到公告板、新闻组、邮件列表或者其他机构中，或者为数据处理机制提供诸如提交表单后的结果等数据。Post 请求的功能由 Web 服务器决定，依赖于 URL 所指向的应用程序。长期以来，在填好的表单提交之后，单击浏览器“ 后退 ”按钮，表单内容是空白的，这避免了用户由于疏忽两次提交表单的可能。不过很多浏览器还是会自动保存已经发送过的表单信息，单击浏览器“ 后退 ”按钮也会看到刚刚提交的部分表单内容。

使用 Get 和 Post 提交表单的主要不同之处是 Get 显示追加了查询字符串的表单参数，Post 连同请求消息体一起发送表单参数。

4、Head 请求

除了服务器禁止在响应中发送消息体外，Head 的发送方式和 Get 一致。Head 请求的 HTTP 头所包含的信息与 Get 请求的响应中的信息相同，可以使用 Head 请求获取没有发送消息体而由该请求暗指的消息的有关元信息。也可以使用请求消息测试超文本链接的有效性、可访问性以及最新变动。

5、状态管理

正如前面所提到的，HTTP 协议是无状态的，不能保存每次提交的信息，即当服务器返回与请求相对应的应答之后，这次事务的所有信息就丢掉了。如果用户发来一个新的请求，服务器无法知道它是否与上次的请求有联系。

对于简单的静态 HTML 文件来说，这种特性很是适用，但是对于那些需要多次提交请求才能完成的 Web 操作比如购物车来说，就成问题了。服务器端 Web 应用程序必须允许用户通过多个步骤才能完整全部的物品采购。这种情况下，应用程序必须跟踪由同一个浏览器发出的多个请求所提供的信息，即记住用户的交易状态。

通常，采取两种方法来解决这个问题。一是在每次应答中都返回完整的状态，让浏览器把它作为下一次请求的一部分再发送会来。二是把状态保存在服务器的某个地方，只发送回一个标识符，浏览器在下次提交中再把这个标识符发送会来；这样就可以定位存储在服务器上的状态信息了。

在这两种方法中，信息可以通过下列三种方法之一发送给浏览器：作为 Cookie、作为隐藏域嵌入 HTML 表单中、附加在主体的 URL 中（通常作为指向其他应用程序页面的链接，即 URL 重写）。

Cookie 是服务器在应答信息中传送给浏览器的名称/值对。浏览器保存这些 Cookie，保存的期限由 Cookie 的有效期属性决定。当浏览器向服务器发送一个请求时，它检查 Cookie 设置，并将它从同一个服务器收到的所有 Cookie 都注入请求信息中。使用 Cookie 是处理状态问题的一个简单的方法，但不是所有的浏览器都支持，用户也可能禁用 Cookie。

如果使用 HTML 表单中隐藏域来向浏览器发送状态信息的话，当表单提交时，浏览器将以常规 HTTP 参数的方式将这些信息返回服务器。当状态信息被注入 URL 时，它将作为

请求 URL 的一部分被传送到服务器。

在浏览器和服务器之间反复的来回传送所有状态信息不是一种高效的方法，所以大部分的服务器还是选择在服务器上保存信息，而只在浏览器和服务器之间传送一个标识符。这个就是所谓的会话（Session）跟踪。来自浏览器的所有包含同一个标识符（这里是会话 ID）的请求同属于一个会话，服务器则对与会话有关的所有信息保持跟踪。会话的有效期直到它被显式的中止，或者当用户在一段时间内没有动作，由服务器自动设置为过期。目前没有办法通知服务器用户已经关闭浏览器，因为在浏览器和服务器之间并不存在持久的连接，并且当浏览器关闭时也不向服务器发送消息。同时，关闭浏览器通常意味着会话 ID 丢失；Cookie 将过期，或者注入了信息的 URL 将不能再使用。所以，当用户再次打开浏览器的时候，服务器无法将新的请求与一千的会话联系起来，而只能创建一个新的会话。然而，所有与前一个会话有关的数据依然存在服务器上，直到会话过期被清除为止。

2.2 B/S 架构的请求响应机制

Web 应用程序采用 B/S（Browser/Server）结构即浏览器和服务器结构。在这种结构下，用户工作界面是通过 Web 浏览器来实现，极少部分事务逻辑在前端（Browser）实现，主要事务逻辑在服务器端（Server）实现，形成所谓三层结构。

相应的，Web 系统的工作机制如下：客户端使用浏览器与服务器建立连接，用户在客户机上所下达的指令通过浏览器分析后，将被发送到服务器端；当服务器收到用户的请求之后，响应客户端的请求，回送应答的数据，把存放在服务器上的消息（以 HTML/XTHML/XML 等方式）传回给用户，然后再由浏览器显示在屏幕上。当客户端发出断开连接的请求后，服务器关闭连接，本次会话结束。这个过程如图 2 - 1 所示。

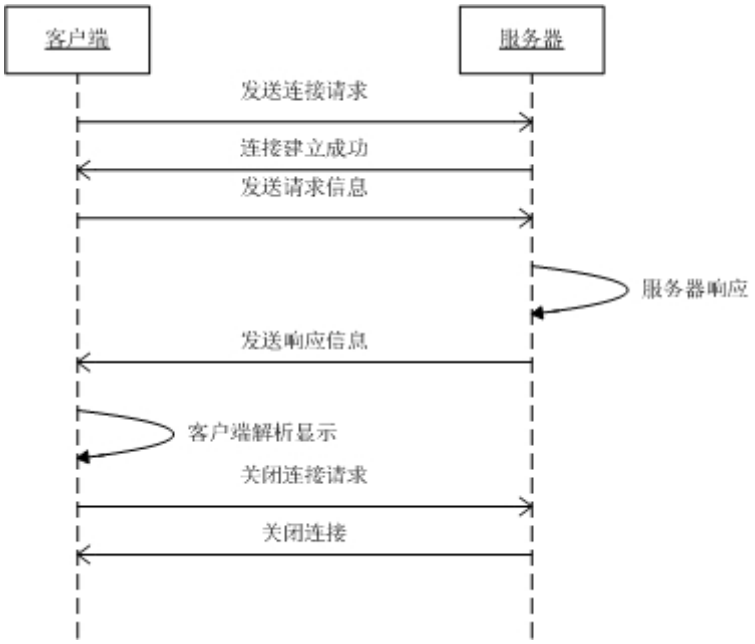

图 2 – 1 B/S 请求响应过程

而部署在服务器的 Web 系统的主要职责是根据服务器转发的客户端发送的请求信息，判断用户的请求行为，调用、综合、编译来自各种数据源的数据信息，加入到服务器要返回给客户端的信息内容中，设置返回信息的相关属性值。服务器最终再将这些响应信息发送给

客户端，完成整个响应过程。这个过程如图 2 - 2 所示。

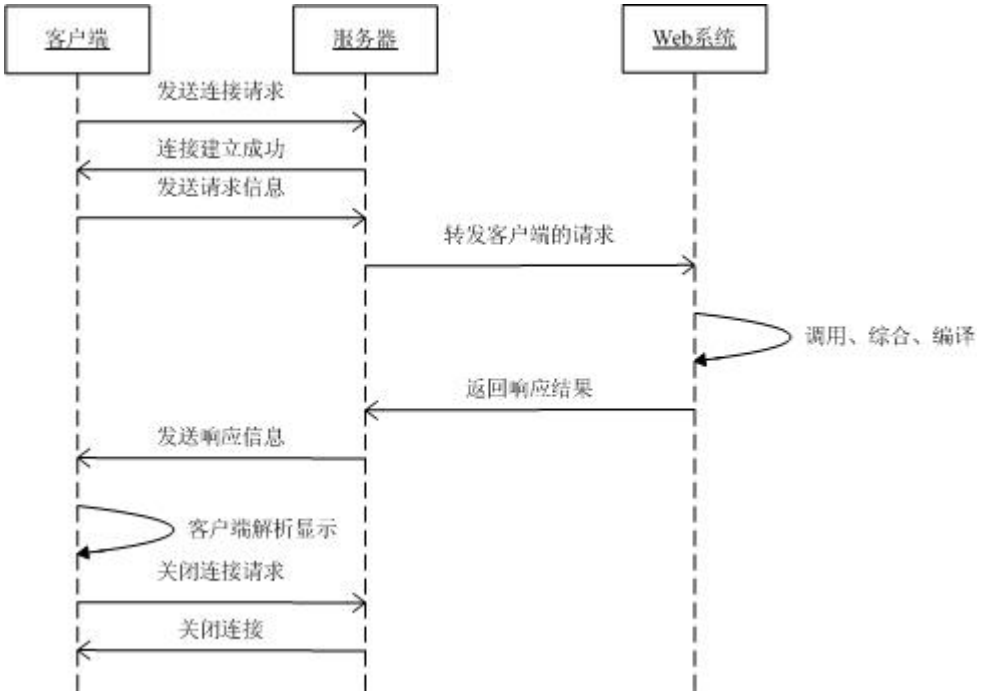

图 2 – 2 Web 系统响应过程

Web 系统可以由 CGI、ASP、Java、CGI、PHP、.NET 等语言编写，Web 服务器也可以根据 Web 系统的要求选择合适的服务器。开发人员可以根据需要选择符合自己需求的技术方案。

2.3 Web 开发模式

从早期的 CGI 到 ASP、PHP，再到现在的 Java 和.NET， 开发技术在不断发展。开发模式也在不断发展，从早期的单层模式（HTML 等静态页面）到二层模式，再到现在广泛流行的 MVC 三层模式和多层模式。随着需求的增加和变更频繁，Web 系统也越来越复杂。本节将着重以二层模式和三层模式探讨 Ajax 在 Web 系统中的位置。

1、单层模式

在单层模式中，浏览器向 Web 服务器提交请求，直接获取保存在 Web 服务器上面的静态信息内容，比如 HTML 文件等。这些信息内容不需要经过 Web 系统的解析编译就可以由 Web 服务器直接发送给浏览器，由浏览器解析呈现。这种模式主要出现在早期的 Web 系统中，也部分包含在二层、三层模式的 Web 系统中。单层模式下的 Web 系统开发，只需要把相应的静态文件放到 Web 服务器就可以了。单层模式的请求响应过程如图 2– 3 所示。

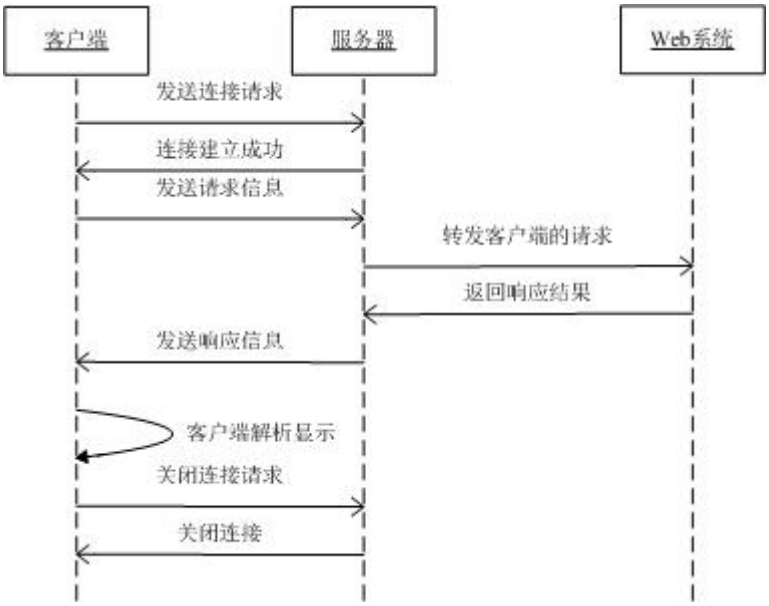

图 2 – 3 单层模式下的请求响应过程

2、二层模式

二层模式下，浏览器向 Web 服务器请求所获得的信息内容往往是经过动态编译完成的。这些信息内容需要包含来自各种数据源的数据，比如数据库、其他数据文件等。各种各样的数据经过 Web 系统的调用、综合、编译之后，才通过 Web 服务器发送给浏览器呈现。

在这种模式下，Web 系统直接跟数据源交互取得所需的数据。在 Web 系统调用、综合、编译之前，这些数据是分散的无组织的。系统的负载主要有 Web 系统完成。这种模式主要以 ASP 技术为代表，对数据源的访问、业务逻辑的操作、用户界面的显示都集中在 Web 系统中一个或者多个文件完成。二层模式下的 Web 系统开发，开发人员要专注于从各种数据源取得并加工数据、业务逻辑的实现以及用户界面显示呈现。随着需求越来越复杂，应用系统越来越大，二层模式构建的系统也会越来越复杂。二层模式的请求响应如图 2 – 4 所示。

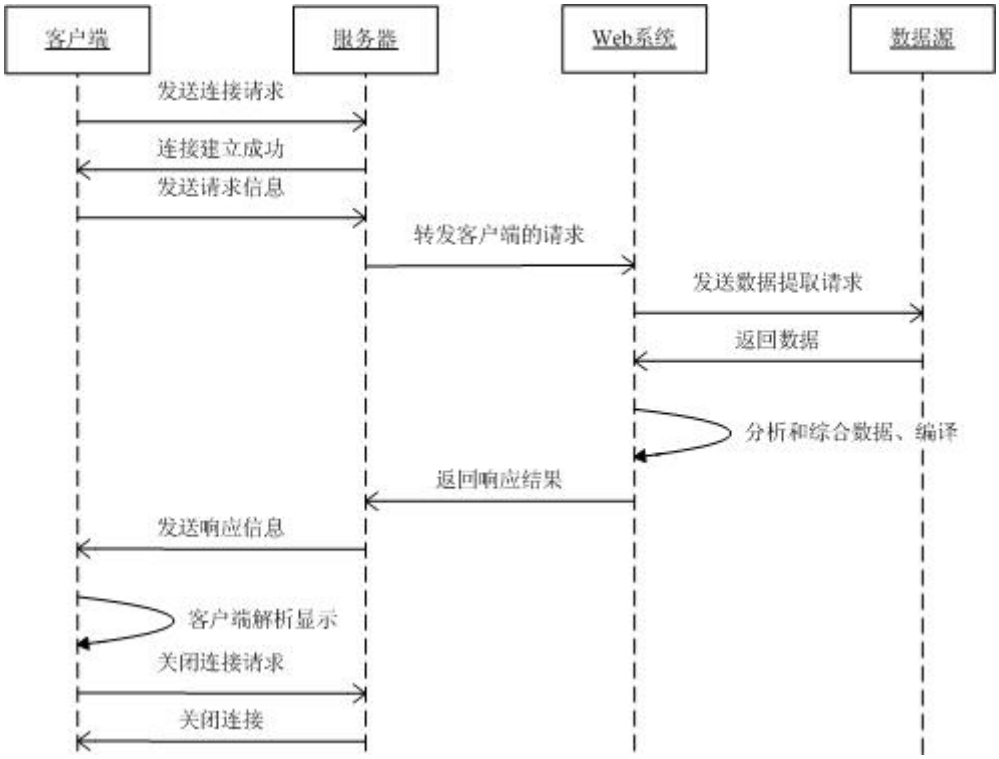

图 2-4 二层模式下的请求响应过程

3、三层模式

三层模式在二层模式的基础上，将 Web 系统分为三层：客户显示层、业务逻辑层、数据层。数据层位于最底层，用来定义、维护、访问和更新数据并管理和满足应用服务对数据的请求；数据层封装对数据源的操作访问，向业务逻辑层暴露访问接口。业务逻辑层处于中间，把客户显示层和数据层分开，用来执行应用策略和封装应用模式，并将封装的模式呈现给客户显示层；其不需要直接跟数据源交互，只需要访问数据层的数据接口。显示层为客户提供应用服务的图形界面。三层模式下，各层逻辑清楚，各司其职，数据层负责数据访问，业务逻辑层专注于业务逻辑的处理，用户显示层则负责用户界面的设计与呈现。

三层模式具有良好的灵活性、安全性和可扩展性，具备比较高的稳定性和执行效率、容错能力和负载均衡能力。客户显示层、业务逻辑层、数据层的关系如图 2-5 所示。

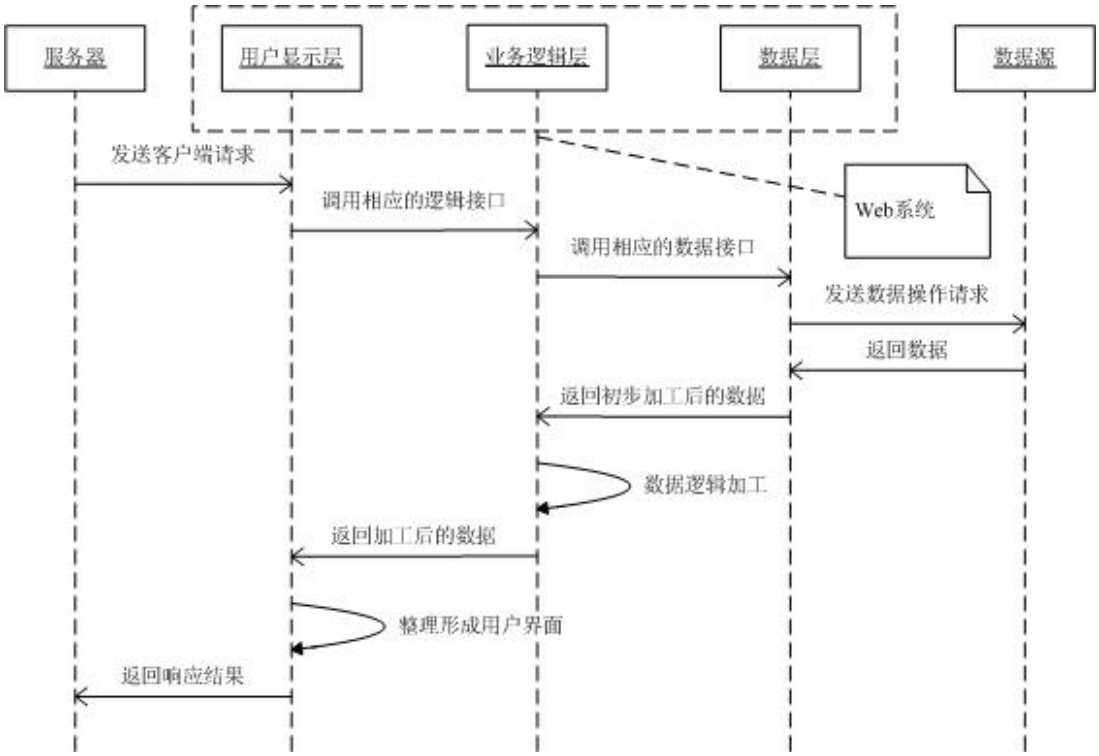

图 2-5 三层模式下各层的关系

4、MVC 模式

进一步的，可以将用户显示层和业务逻辑层继续划分为 *MVC（Model、View、Controller，模型、视图、控制器）*三个层次，把一个应用的输入、处理、输出进行合理的分离。这就是著名的 MVC 模式。

- 模型(Model)

模型继承原来业务逻辑层的功能，包含了应用程序的核心，它封装了应用程序的数据结构和事务逻辑，集中体现了应用程序的状态。有时候仅包含状态信息，因为它并不了解视图或控制器的信息。模型不依赖于视图和控制器。在 Java 技术中，Java Bean / EJB 很适合扮演这个角色，因为其能够处理绝大部分事务逻辑和数据结构，还能与数据库或文件系统进行交互，承担维护应用程序数据的责任。

- 视图(View)

视图继承原来客户显示层的功能，实现模块的外观，它是应用程序的外在表现。它可以访问模型的数据，却不了解模型的情况，同时它也不了解控制器的情况。当模型发生改变时，视图会得到通知，它可以访问模型的数据，但不能改变这些数据。Web 界面的开发人员不需要了解或关心数据库发生什么事情，或者事务逻辑进行了什么操作，他只需要掌握 HTML 的知识，而不需要深入了解 Java。JSP 页面适合完成这个功能，因为它只包含很少的非 HTML 代码。

- 控制器(Controller)

控制器控制整个框架中各个组件的协调工作，对用户的输入做出反应，并且将模型和视图联系在一起，它创建并设置模块。Servlet 能够接受客户端的 HTTP 请求，并且根据需要创建所需的 Java Beans，然后将模块产生的变化通知给视图。

模型、视图、控制器三个层次的责任和关系如图 2-6 所示。

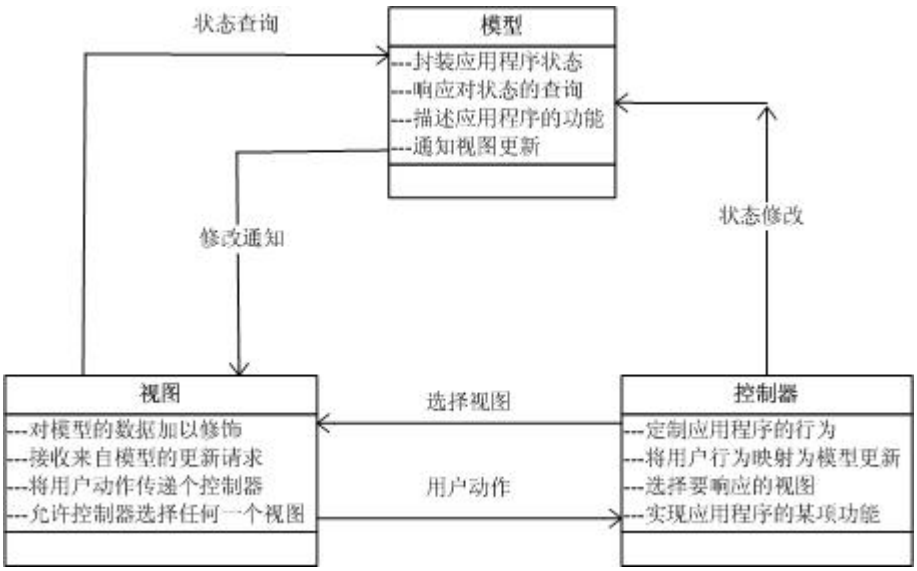

图 2-6 模型、视图、控制器三个层次的责任和关系图

三层模式下的 Web 系统开发，可以对参与人员进行良好的分工。开发人员可以将精力集中于模型层、控制层、数据层的设计以及对复杂的数据封装和业务逻辑处理；网页设计人员可以集中精力设计更加人性化的用户界面。

在 Java 技术领域，Web 开发模式也经历了从二层模式（JSP 结合 JDBC、JSP 结合 Java Bean）到三层模式（JSP、Servlet 结合 Java Bean，MVC）再到成熟开发框架（Struts、WebWork2）的发展。

在早期阶段，JSP 直接操作 JDBC。在这类实现中，数据显示、逻辑处理、数据持久化都集中在视图层即 JSP 页面完成。这种开发模式在项目小、需求简单的时候确实给系统开发带来了好处。但是当项目逐步增大、需求复杂的情况下，会给调试、修改、维护、扩展以及开发效率的提高带来不少问题。

之后，Java Bean 诞生，JSP 结合 Java Bean 的开发模式也逐渐成为主流。在这种开发模式中，Java Bean 封装对持久化数据的访问操作，处理复杂业务逻辑，向 JSP 页面暴露相关接口；JSP 页面负责页面显示以及接收页面请求，并调用相应的 Java Bean 接口来完成逻辑处理。在这样的设计模式中，逻辑是由 Java Bean 封装的，可以对其进行调试，代码重用和有一定的保证了，程序结构也相对清晰了很多。但是页面中还是需要编写代码请求页面的数据，还得根据请求调用相应的 Java Bean 接口，然后根据处理结果再转到相应的页面。

随着 MVC 理论的成熟和应用，Java Servlet 逐渐被应用为 Java MVC 模式的控制器，Java Bean 被抽象为模型层，JSP 独立成为视图层。JSP 页面将控制权交给 Java Servlet 后，Java Servlet 将封装好的 HttpServletRequest、HttpServletResponse 作为参数传给 Java Bean 处理；Java Bean 将处理结果放入 HttpServletResponse；Java Servlet 再次获得控制权，根据处理的结果返回相应的页面。这种处理方式当 JSP、Java Bean 频繁变动的情况下，Java Servlet 这个控制器也要跟着频繁的变动，特别是调用相应的 Java Bean 以及转发相应页面的部分。后来逐步将控制器深化为前台控制器和应用控制器两部分，前台控制器负责接收并封装来自 JSP 页面的请求、根据处理结果转发到相应的页面，应用控制器负责调用相应的 Java Bean。于是，MVC 模式深化成为“ 视图层＋前台控制器＋应用控制器＋模型层 ”。之后为了避免因为变化频繁修改前台控制器，引入了配置文件的解决方法，编写 action 的配置文件，控制根据 action 的返回结果转入相应的视图。

MVC 思想被广泛使用之后，出现了诸如 Struts、WebWork2、Spring 等一批成熟的框架，进一步封装、简化并深化 MVC 模式，同时也带动了异步请求相应技术如 Ajax 的发展。

5、Ajax 与 Web 开发模式

Ajax 不是一项新的技术。其综合运用 Javascript、XHTML 和 CSS、DOM、XML 和 XSTL、XMLHttpRequest 等技术，提供异步的数据交换方式，加快 Web 系统中客户端服务器的数据交换速度，改善用户体验。

对于采用单层或者二层模式的系统，Ajax 引擎集成于 Web 系统，跟 HTML、ASP 等文件绑定在一起，嵌入到单独的页面编码中，可以帮助 Web 系统异步获取服务器的信息，在不刷新浏览器的情况下更新界面内容。Ajax 引擎与 Web 系统的关系如图 2-7 所示。

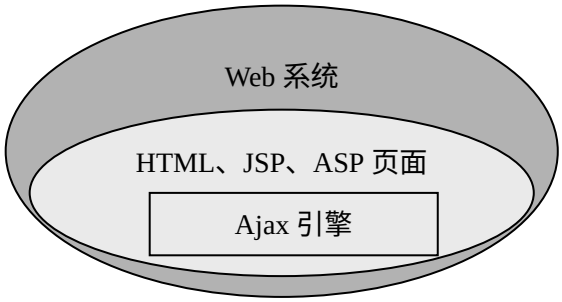

图 2-7 单层和二层模式下 Ajax 在 Web 系统中的位置

在采用三层模式乃至 MVC 层次的 Web 系统中，Ajax 引擎处于客户表示层（视图）层当中，在异步方式下直接或间接的访问业务逻辑层或者控制器的相应接口，获取系统数据，在不刷新浏览器界面的情况下更新用户界面内容，以此方式提高数据传输速度，改善系统用户体验。三层模式 Web 系统中，Ajax 引擎所处的位置如图 2-8 所示。

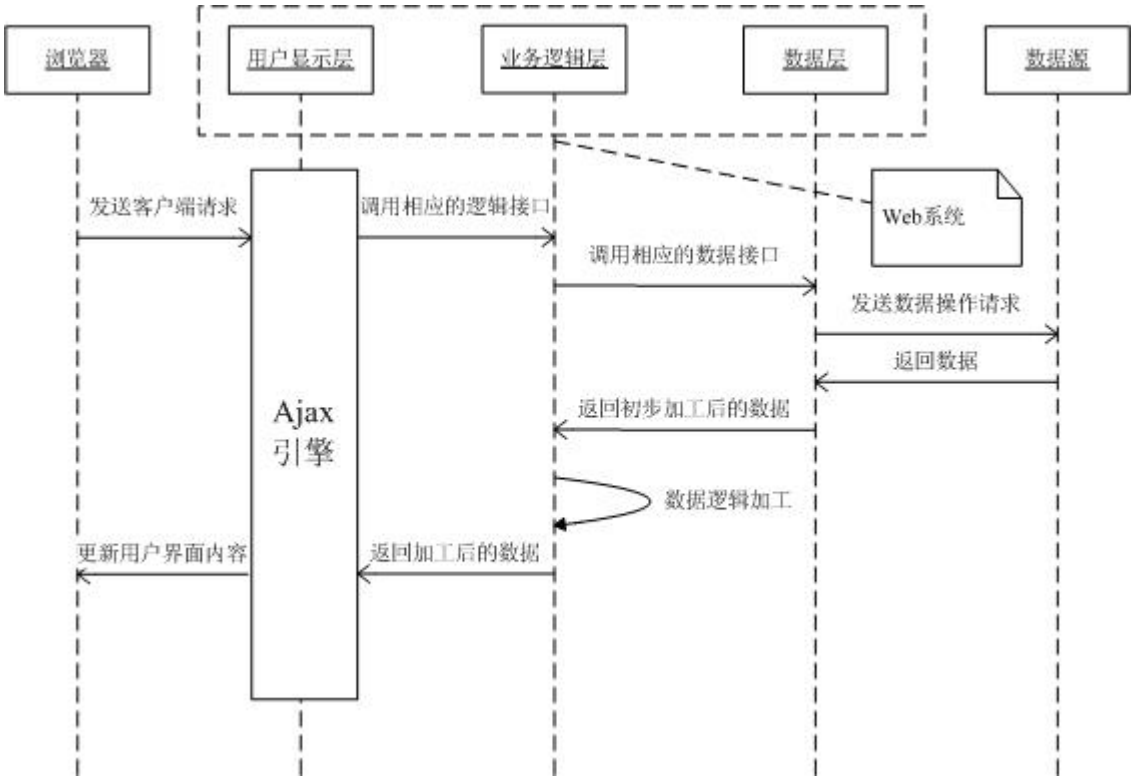

图 2-8 Ajax 在三层模式 Web 系统中的位置

对于应用了诸如 Struts、WebWork2、Spring 等框架的 Web 系统，Ajax 引擎可以跟视图无缝集成，Ajax 将请求异步发送个相应的控制器，在控制器返回处理结果后再更新视图内容。

2.4 开发环境的设置与配置

Ajax 所依赖的 Javascript 等技术不依赖于特定的 Web 系统开发语言比如 ASP、JSP 等。因此，Ajax 技术也同样适用于 ASP、Java（JSP）、PHP、.NET 等技术。

默认的，本书大部分的示例代码用 Java（JSP、Servlet）语言编写。 为测试需要，本节搭建一个开发测试环境，包括一个 Web 服务器、一个 Java 开发工具包和一个 Java 开发 IDE。Java Servlet API 2.3 需要 JDK 1.4；JSP API 1.2 需要 Tomcat 1.4。故本书选用 JDK（J2SE Software Development Kit）1.4.2 作为编译器、Tomcat 4.1.31 作为 Web 服务器，Eclipse3.1.1 作为 Java 开发 IDE。需要其他测试环境和开发语言的读者可以根据自身的情况进行相应的设置与配置，并改写相应的程序代码。

2.4.1 下载 Java 开发工具包（JDK）

JDK 1.4.2 可以从 SUN 公司的网站下载，下载地址为 ：*http://java.sun.com/j2se/1.4.2/download.html* 。本书采用自动安装版本，下载后是一个j2sdk-1_4_2_10-windows-i586-p.exe文件。双击下载后的exe文件开始安装，直接点击**下一步**直到安装结束。默认安装路径为：C:\j2sdk1.4.2_10。将此路径称为JAVA_HOME，本文后续章节提到{JAVA_HOME}即是指此路径。在操作系统中增加设置如下环境变量：

- JAVA_HOME：JDK 安装目录即 *C:\j2sdk1.4.2_10*。
- PATH：*%JAVA_HOME%/bin*。
- CLASS_PATH：*%JAVA_HOME%/lib*。

设置结果如图 2-9 所示。

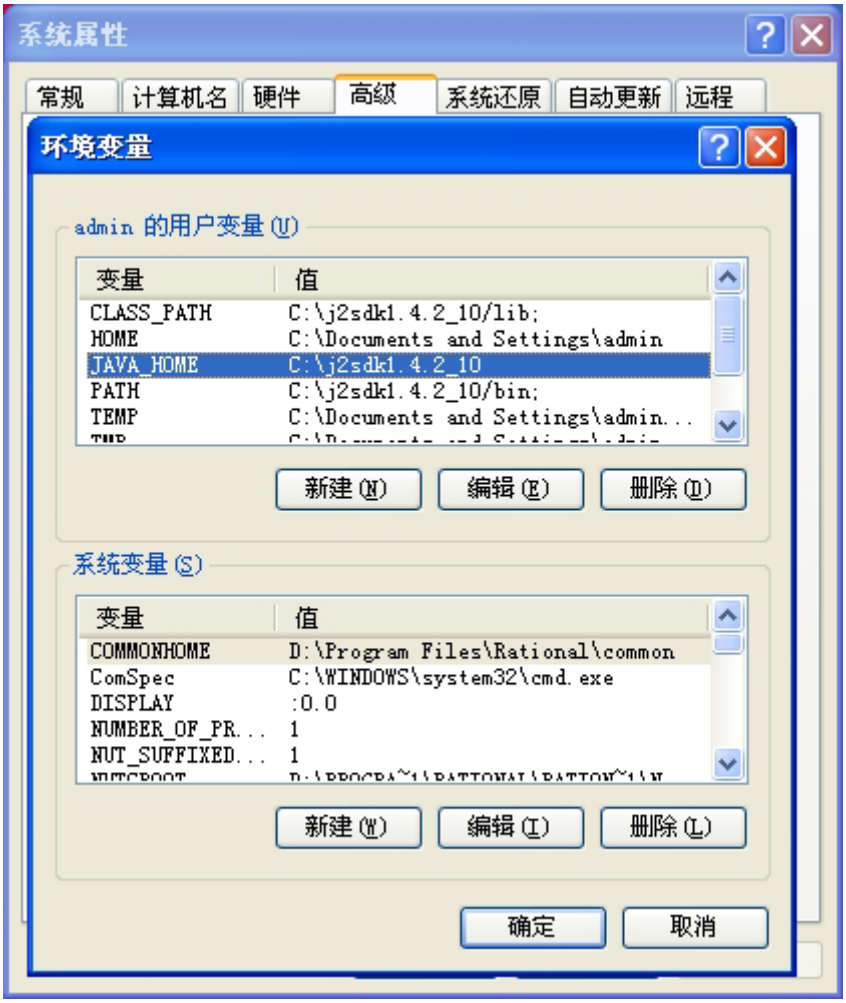

图 2-9 设置操作系统环境变量

2.4.2 下载安装 Tomcat

Tomcat 4.1.31 可以从 apache organization 的官方网站下载，下载地址为：*http://tomcat.apache.org/download-41.cgi* 。这里选择自动安装版本，下载后是一个 jakarta-tomcat-4.1.31.exe文件。双击下载后的exe文件开始安装，将安装目录改为*C:\Tomcat 4.1*，其他则直接点击“下一步”保持默认设置。在安装的最后一步要设置端口号。如果Web服务端口 80 没有被其他服务器占用，则设置端口为 80 更为方便；否则保持 8080 的默认值。安装完毕，在浏览器中输入*http://localhost:8080/*，如果所得的Web页面如图 2-10 所示，则表示Tomcat安装成功。

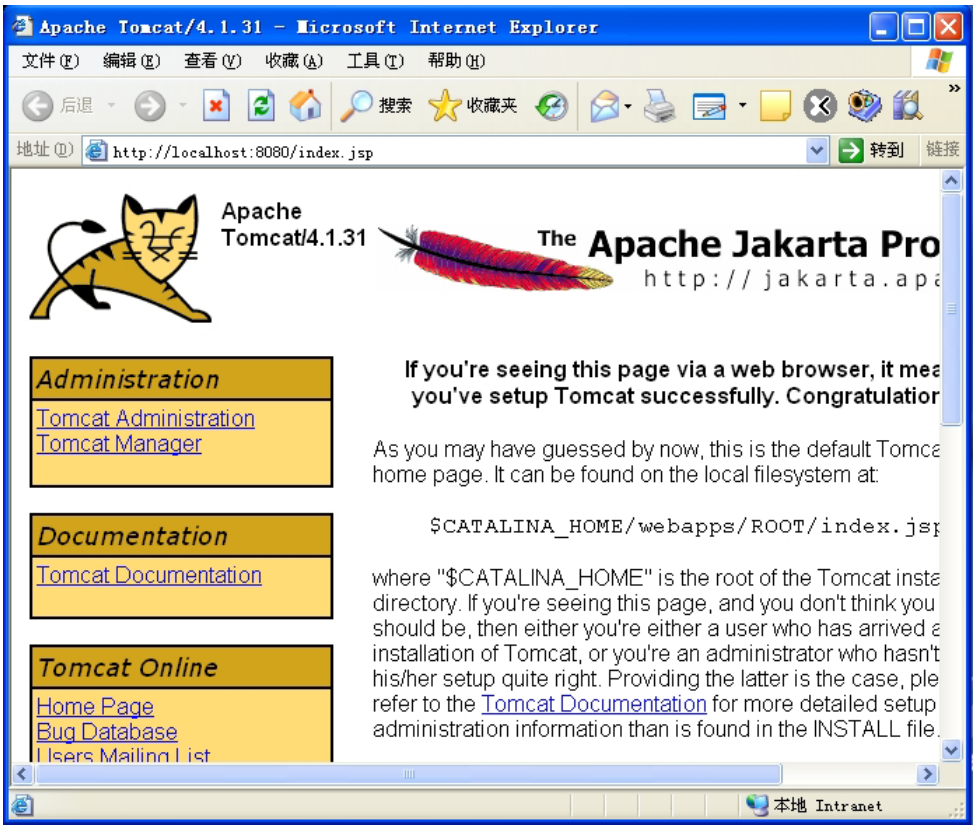

图 2-10 Tomcat 安装成功

将 Tomcat 的安装目录称为 *TOMCAT_HOME*，本书后续章节提到{TOMCAT_HOME}即指此目录。

2.4.3 下载安装 Eclipse 3.1.1

从http://www.eclipse.org 下载Eclipse 3.1.1 压缩包，解压缩到目录*D:\Eclipse*，将此目录称为ECLIPSE_HOME。从*http://www.sysdeo.com/eclipse/tomcatPlugin.html* 下载Tomcat插件压缩包。下载解压缩后将plugins目录拷贝覆盖*{ECLIPSE_HOME}\plugins*目录。双击Eclipse.exe启动Eclipse。

单击“Windows”菜单“Customize Perspective”命令，当“Submenu”下来列表框选择New 时确保“Shortcuts”列表框中“Tomcat Project”复选框被选中。如图 2-11。

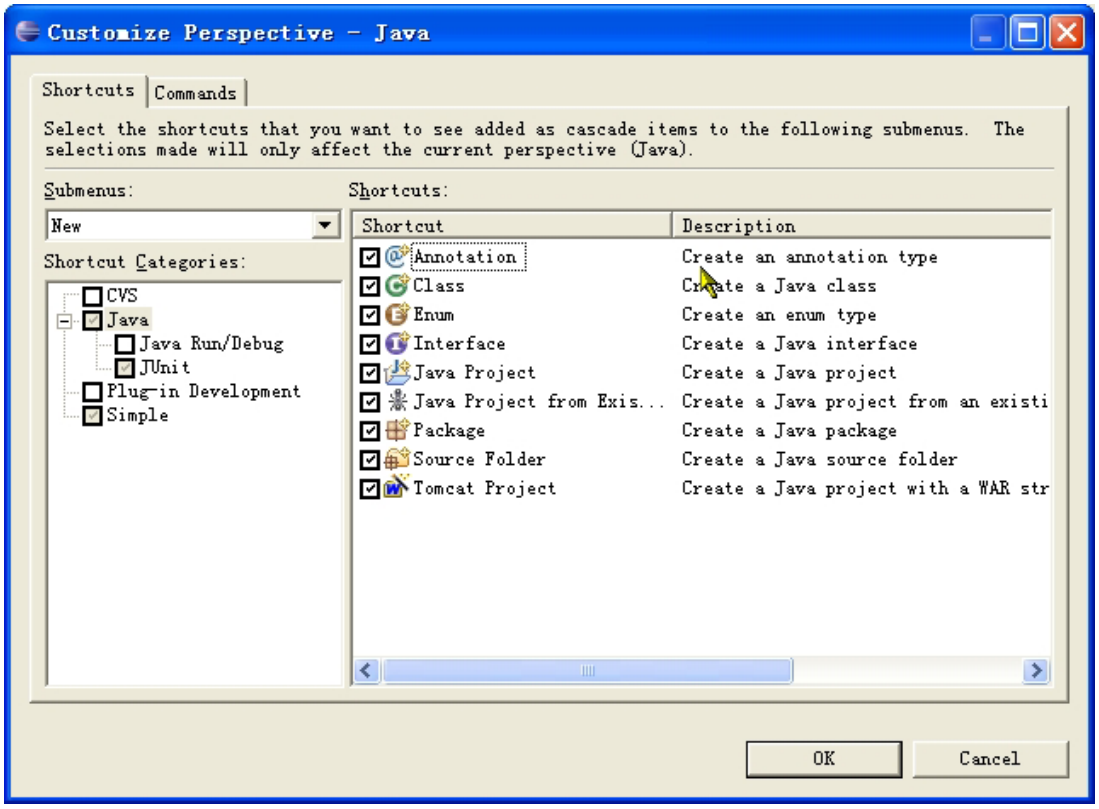

图 2-11 配置 Eclipse 的 Tomcat 菜单

单击“ Windows ”菜单“ Preferences ”命令，在“ Tomcat ”选项中设置属性如下：

- Tomcat Version：Version 4.1.x。
- Tomcat Home：C:\Tomcat 4.1。
- Context declaration mode：server.xml。

结果如图 2-12 所示。

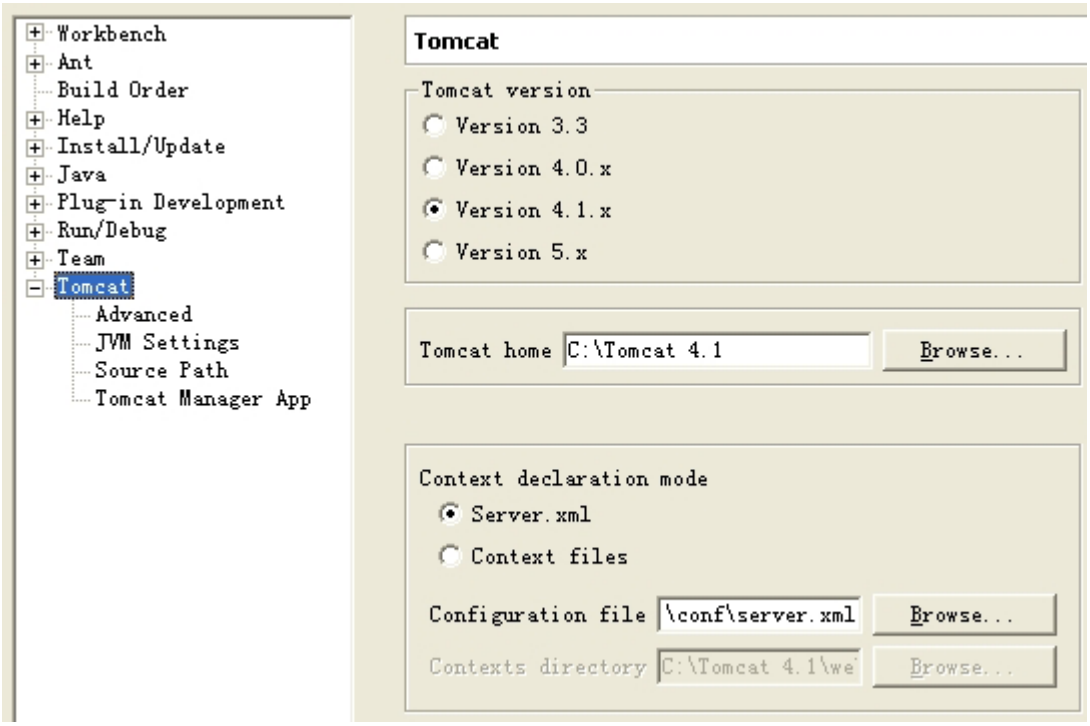

图 2-12 配置 Eclipse 的 Tomcat 插件

2.4.4 创建开发目录

按照下列步骤创建开发目录。

（1）创建文件夹：

在 D 盘新建文件夹 ajaxlab，将此目录称为 *APPLICATION_HOME*。

（2）创建工程：

启动 Eclipse，单击“ File ”-“ New ”菜单，单击“ Project ”命令，设置“ Project Name ”属性值为 ajaxlab，“ Location ”选项卡选中“ Create Project at external location ”，“ Directory ”属性值为 D:\ajaxlab；单击“ Finish ”结束。

（3）创建工程源文件夹：

单击“ File ”-“ New ”菜单，单击“ Source Folder ”命令，设置属性值“ Folder name ”为：src。

（4）创建包：

选中 src 目录，单击“ File ”-“ New ”菜单，单击“ Package ”命令，设置属性值“ name ”为：com.ajaxlab.ajax。

（5）创建 Web 文件夹：

选中 ajaxlab 工程，单击“ File ”-“ New ”菜单，单击“ New Folder ”命令，设置“ Folder Name ”属性值为 webapps。选中 webapps 目录，单击“ File ”-“ New ”菜单，单击“ New Folder ”命令，设置“ ”Folder Name 属性值为 WEB-INF。按照相似的方法，再为 WEB-INF 创建子文件夹 classes 和 lib。称*{APPLICATION_HOME}\webapps* 为 *APPLICATION_WEB_HOME*。

（6）设置输出目标文件夹：

选中 ajaxlab 工程，单击鼠标右键选择“ Properties ”命令，在“ Java Build Path ”选项中

将“ Default output folder ”属性更改为｛APPLICATION_HOME｝\webapps\WEB-INF\classes。

（7）创建 Web 部署描述文件 web.xml：

在*{APPLICATION_HOME}\webapps\WEB-INF* 目录下创建 web.xml 文件，文件内容如下：

例程 2-1：web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>Ajax Application</display-name>
  <session-config>
    <session-timeout>30</session-timeout>
  </session-config>
  <mime-mapping>
    <extension>htm</extension>
    <mime-type>text/html;charset=gb2312</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>html</extension>
    <mime-type>text/html;charset=gb2312</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>doc</extension>
    <mime-type>application/msword</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>xls</extension>
    <mime-type>application/msexcel</mime-type>
  </mime-mapping>
  <welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

（8）映射部署目录：

修改*{TOMCAT_HOME}\conf\server.xml*，为 Context 增加一个兄弟节点，内容如下：

```
<Context path="/ajaxlab" docBase="D:\ajaxlab\webapps"
debug="0" reloadable="true" crossContext="true">
</Context>
```

在对 server.xml 进行修改之前，应该对它备份，以便出现服务器出现不能运行的错误时可以正常恢复。

（9）测试部署目录：

在 *APPLICATION_WEB_HOME* 下创建一个 test.jsp 件，源码如下：

例程 2-2：test.jsp

```
<%@ page contentType="text/html; charset=gb2312" errorPage="" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>无标题文档</title>
</head>
<body>
测试文件
</body>
</html>
```

单击Eclipse的File-Tomcat菜单，单击Start Tomcat命令，启动Tomcat。在浏览器中输入：*http://localhost:8080/ajaxlab/*test.jsp 。如果页面运行正常并能看到“ 测试文件 ”的显示字符串，则Web目录部署成功。

创建完毕的开发目录如图 2-13 所示。

图 2-13 创建完毕的开发目录

2.5 小结

本章讲述了 HTTP 协议下以及 B/S 请求响应机制，以及在此机制下的 Web 系统开发模式，并部署了一个用于后续章节开发测试之用的开发测试环境。

HTTP 协议具有无状态、无连接的特点，所以 Web 系统必须不断的重新向服务发起请求信息，等服务器响应之后再更新界面内容。每次都是一个新的会话过程。

Web 开发模式历经多年的发展变化，从单层模式到二层模式，从三层模式到 MVC 封装，新开发技术也层出不穷，诸如 Struts、WebWork2、Spring 等的开发框架也逐渐成熟并广泛应用。MVC 模式的发展也促进了 Ajax 的发展。

Ajax 是一个可以被各种开发模式、各种开发语言应用的综合技术。对于采用单层模式和二层模式的 Web 系统，Ajax 引擎跟 HTML、JSP、ASP、PHP 等文件绑定在一起，与上述文件相互配合；对于采用三层模式、MVC 乃至框架的 Web 系统，Ajax 引擎包含在视图层中，直接于控制器打交道。

随着时间的推移，Ajax 技术将进一步发展，将被各种开发模式和开发技术所利用，开 B/S 结构和 Web 系统异步通信的先河。