第 1 章 Web 开发环境

本章主要介绍 Java Web 的开发环境，介绍了胖客户端程序 RCP、瘦客户端程序 TCP、B/S 与 C/S 结构、Web 访问基本原理、Web 浏览器、传统的 Web 服务器模式开发、动态展现页面技术、TOMCAT 服务器简介、Tomcat 目录结构、Apache Tomcat 5. x 的简单介绍、Apache 和 Tomcat 的配置、配置用户定制目录、在 Tomcat 中使用 CGI 脚本、改变 Tomcat 中的 JSP 编译器、Tomcat 的安全部署、HTTP 协议技术架构、协议功能、协议基础、开发环境的搭建、JSP 技术简介等内容。通过本章的学习能够自行搭建 Java Web 的开发环境。

1. 1 Web 开发概述

Web 程序也就是一般所说的网站，它由服务器、客户端浏览器以及网络组成。Web 程序的好处是使用简单不需要安装，有一台电脑、一根网线就可以使用。截至 2006 年年底互联网上的网站数量已经超过了 1 亿，中国的网站也已经有 200 万之多了，可见网络的影响力。但 Web 程序又不是一般意义上的网站。网站的目的是提供信息服务，重在内容，程序往往比较简单。但一个商用的 Web 程序往往比较复杂，背后常结合数据库等技术，例如 ERP 系统、CRM 系统、财务系统、网上办公、网上银行、在线业务办理等。下面从专业上解释 Web 程序相关的几个概念。

1. 1. 1 胖客户端程序 RCP

桌面程序 Desktop Program 也叫胖客户端程序 Rich Client Program，简称 RCP。因为桌面程序需要安装到计算机上才能运行并会导致计算机软件的体积越来越大，因此人们形象地称桌面程序为胖客户端程序。计算机上安装的任何程序都是 RCP。例如办公软件 Word、Excel，聊天工具 QQ、MSN，播放软件 Media Player、Flash Player，图像制作软件 PhotoShop 等。RCP 的优点很明显，只要安装上了软件就能高效地使用软件的功能。RCP 的缺点也很明显，就是需要安装才能使用，并且会占用大量的硬盘资源。如果某个公司的 1000 台电脑都要使用 Word，那么这 1000 台电脑都要安装 Word。

1.1.2 瘦客户端程序 TCP

与胖客户端程序相对的是瘦客户端程序。瘦客户端程序 Thin Client Program 简称 TCP，一般表现为 Web 程序，它的特点是不需要在客户端安装便能使用，只要计算机能上网就行。瘦客户端程序将软件功能的重点集中放到了服务器上，服务器端只需要提供服务。目前流行的概念“软件即服务”SAAS(Software-as-a-service)就是一种非常流行的瘦客户端应用。它是通过 Internet 提供软件，用户不用再购买软件而改用向提供商租用基于 Web 的软件来管理企业经营活动且无须对软件进行维护、升级。目前越来越多的 Web 2.0 概念的应用也都是瘦客户端的应用，随着技术的不断进步，瘦客户端程序的体验也越来越丰富。Google 已经提供了许多功能强大的 Web 程序例如在线 Word、Excel、PDF 等功能用于取代桌面程序。相信在不久的将来会有越来越多的 TCP 应用的出现。

1.1.3 B/S 与 C/S 结构

按照是否需要访问网络程序可分为网络程序与非网络程序。其中网络程序又可分为 B/S 结构与 C/S 结构。

C/S 结构是指客户端 Client/服务器 Server 模式。这种模式的客户端中需要安装一个 RCP 程序。RCP 程序负责与服务器进行数据交换。一般的网络程序都是 C/S 结构例如 QQ、MSN、PP Live、迅雷、eMule 等。以往基于客户、服务器的 C/S 结构应用程序存在很多缺点：它需要安装客户端程序，当应用程序升级时客户端同样需要下载升级程序才能使用新的功能。这样无形中会给客户端带来一定的麻烦，限制了该应用程序的广泛使用。当今更多的下载软件、即时通信软件等都是 C/S 结构的应用程序。

B/S 是指浏览器 Browser/服务器 Server 模式。一般的网站都是 B/S 结构的例如 Google、Baidu。Web 应用程序的访问不需要安装客户端程序，可以通过任意一款浏览器来访问各类 Web 应用程序。当 Web 应用程序进行升级时并不需要在客户端做任何更改。和 C/S 结构的应用程序相比，Web 应用程序可以在网络上更加广泛地进行传播和使用。

1.1.4 Web 访问基本原理

下面我们回想一下平时浏览网页的过程中浏览器和服务器端都发生了什么变化，网站是怎么实现请求和响应功能的。图 1.1 清晰地显示了浏览器访问 Web 服务器的整个过程。

图 1.1 Tomcat 响应机制

(1) 用户打开浏览器如 IE、Firefox 等输入网站的 URL 地址，也就是通常所说的网址。这个地址告诉浏览器要访问互联网中的哪台主机。

(2) 浏览器寻找到指定的主机之后向 Web 服务器发出请求 request。

(3) Web 服务器接受请求并做出相应的处理生成，处理结果大多数生成 HTML 格式，

也有其他响应的格式。

（4）服务器把响应的结果返回发送给浏览器。

（5）浏览器接收到对应的响应结果后在浏览器中显示响应结果，比如 Web 页面。

1.1.5　Web 浏览器

目前有很多 Web 浏览器，但是比较普及和流行的是 Microsoft 公司的 IE 和 Mozilla 基金会的 Firefox 浏览器。这两个浏览器都能很好地支持最新、最好的 HTML 表示标准以及各种 HTML 扩展功能。另外它们也都能支持 JavaScript 脚本语言以及类似 Applet 的 Java 小程序运行。其他的浏览器还有傲游浏览器 Maxthon、腾讯 TT 浏览器、Opera 以及 Google 最新推出的谷歌浏览器 Chrome 等。

1.1.6　传统的 Web 服务器模式开发

传统的 Web 应用开发仅仅能够提供有限的静态 Web 页面 HTML，静态页面每个 Web 页面的显示内容是保持不变的。这种模式开发的 Web 应用很不利于系统的扩展。如果网站需要提供更多新的信息资料时就只能修改以前的页面或者重新编写 HTML 页面，并提供链接。而且 Web 网站的信息更新周期一般都比较长，因为需要重新编写代码。总结起来传统 Web 应用开发模式存在如下多个不足不能提供及时信息，页面上提供的都是静态不变的信息；当需要添加新的信息时必须重新编写 HTML 文件；由于 HTML 页面是静态的，所以并不能根据用户的需求提供不同的信息，包括不同的内容和格式并不能满足多样性的需求。静态页面的应用程序存在着这么多的缺点决定了这种模式必然不能适应大中型系统和商业需求。因此，很快因特网软件工程师转向了 CGI(Common Gateway Interface)公共网关接口系统，它能够提供页面的动态生成。

1.1.7　动态展现页面技术

当发布全部为静态页面的 Web 应用程序即传统 Web 服务器模式开发时，随着企业业务的增多，HTML 页面程序会越来越多，非常不利于后期代码的维护，而且新信息发布过程非常麻烦。所以建立一个动态 Web 应用程序就显得非常重要。一方面可以根据访问者的不同请求返回不同的访问信息即满足服务的多样性；另一方面，可以直接通过后台管理页面发布和修改信息，即可不再需要修改页面程序或者添加更多页面程序。动态 Web 应用程序的建立可以给客户提供及时信息以及多样化服务，可以根据客户不同请求动态地返回不同需求信息。

1.2　Tomcat 服务器简介

Tomcat 服务器是一个免费的开放源代码的 Web 应用服务器，是 Apache 软件基金会(Apache Software Foundation)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而成。由于有了 Sun 的参与和支持，最新的 Servlet 和 JSP 规范总是

能在 Tomcat 中得到体现，Tomcat 5 支持最新的 Servlet 2.4 和 JSP 2.0 规范。因为 Tomcat 技术先进、性能稳定，而且免费，因而深受 Java 爱好者的喜爱并得到了部分软件开发商的认可，成为目前比较流行的 Web 应用服务器。目前最新版本是 9.0。

Tomcat 是 Apache 组织的 Jakarta 项目中的一个重要开源子项目，它是 Sun 公司推荐的运行 Servlet 和 JSP 的容器(引擎)。Tomcat 还具有 Web 服务器的基本功能，提供数据库连接(不常用，基本上没人用)、SSL、Proxy 等许多通用组件功能。Tomcat 也可以作为独立的 Web 服务器软件运行，但它处理静态 HTML 文件的速度比不上 Apache 和 IIS 等专业的 Web 服务器，且其作为 Web 服务器软件的功能也不如 Apache 和 IIS 强大。我们这里只关心 Tomcat 作为运行 Servlet 的容器，负责接收和解析来自客户端的请求，同时把客户端的请求传送给相应的 Servlet，并把 Servlet 的响应结果返回给客户。如图 1.2 所示。

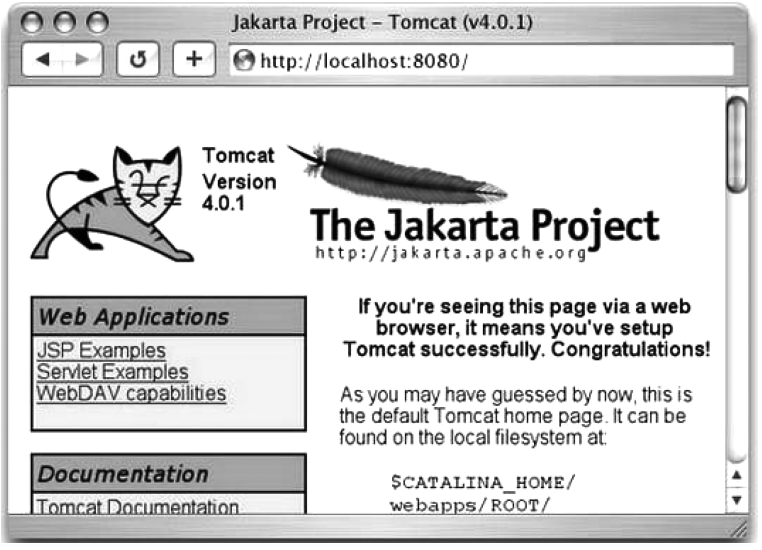


图 1.2　Tomcat 界面

Tomcat 服务器是一个免费的开放源代码的 Web 应用服务器，属于轻量级应用服务器，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。对于一个初学者来说，可以这样认为，当在一台机器上配置好 Apache 服务器，可利用它响应对 HTML 页面的访问请求。实际上 Tomcat 部分是 Apache 服务器的扩展，但它是独立运行的，所以当你运行 tomcat 时，它实际上是作为一个与 Apache 独立的进程单独运行的。诀窍是，当配置正确时，Apache 为 HTML 页面服务，而 Tomcat 实际上运行 JSP 页面和 Servlet。另外，Tomcat 和 IIS、Apache 等 Web 服务器一样，具有处理 HTML 页面的功能，另外它还是一个 Servlet 和 JSP 容器，独立的 Servlet 容器是 Tomcat 的默认模式。不过，Tomcat 处理静态 HTML 的能力不如 Apache 服务器。目前 Tomcat 最新版本为 9.0.27-Released(alpha) Released。Tomcat 网站的下载界面如图 1.3 所示。

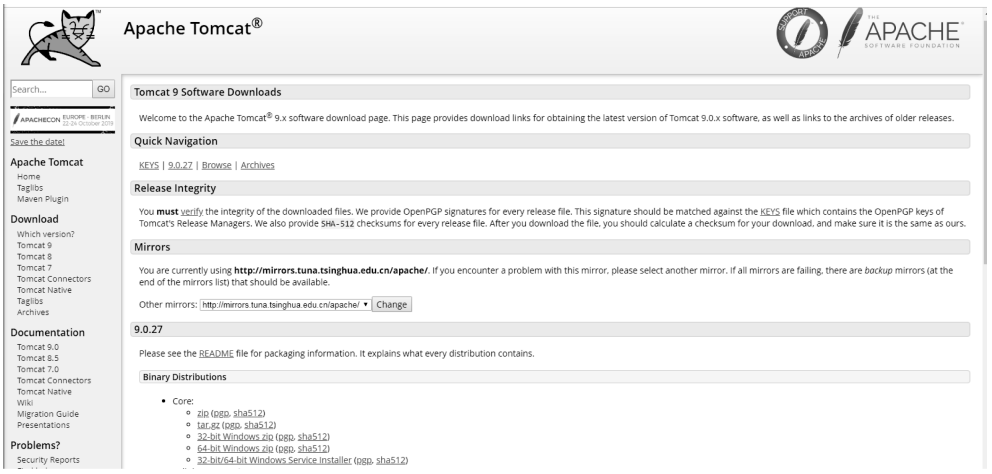


图 1.3　Tomcat 网站的下载界面

Tomcat 很受广大程序员的喜欢，因为它运行时占用的系统资源小，扩展性好，支持负载平衡与邮件服务等开发应用系统常用的功能；而且它还在不断的改进和完善中，任何一个感兴趣的程序员都可以更改它或在其中加入新的功能。

Tomcat4. 0x 中采用了新的 Servlet 容器：Catalina，完整地实现了 Servlet2. 3 和 Jsp1. 2 规范。Tomcat 提供了各种平台的版本供下载，可以从其官方网站上下载其源代码版或者二进制版。由于 Java 的跨平台特性，基于 Java 的 Tomcat 也具有跨平台性。与传统的桌面应用程序不同，Tomcat 中的应用程序是一个 WAR(Web Archive)文件。WAR 是 Sun 提出的一种 Web 应用程序格式，与 JAR 类似，也是许多文件的一个压缩包。在启动 Tomcat 之前，我们必须要配置环境变量。

JAVA_HOME：必须先配置 JAVA_HOME，因为 Tomcat 启动需要使用 JDK。

CATALANA_HOME：如果是安装版，那么还需要配置这个变量，这个变量用来指定 Tomcat 的安装路径，例如：F:\ apache-tomcat -7. 0. 42。

启动：进入%CATALANA_HOME% \ bin 目录，找到 startup. bat[开启服务器]，双击即可；

关闭：进入%CATALANA_HOME% \ bin 目录，找到 shutdown. bat[关闭服务器]，双击即可；

startup. bat 会调用 catalina. bat，而 catalina. bat 会调用 setclasspath. bat，会使用 JAVA_HOME 环境变量，所以我们必须在启动 Tomcat 之前把 JAVA_HOME 配置正确。

启动问题：点击 startup. bat 后窗口一闪即消失，检查 JAVA_HOME 环境变量配置是否正确；这个包中的文件按一定目录结构来组织：通常其根目录下包含有 Html 和 Jsp 文件或者包含这两种文件的目录，另外还会有一个 WEB-INF 目录，这个目录很重要。通常在 WEB-INF 目录下有一个 web. xml 文件和一个 classes 目录，web. xml 是这个应用的配置文件，而 classes 目录下则包含编译好的 Servlet 类和 Jsp 或 Servlet 所依赖的其他类(JavaBean)。通常这些所依赖的类也可以打包成 JAR 放到 WEB-INF 下的 lib 目录下，当然也可以放到系统的 CLASSPATH 中，但那样移植和管理起来不方便。

1.2.1 Tomcat 目录结构

- |－bin：存放 tomcat 的命令。
- |－conf：存放 tomcat 的配置信息，其中 server.xml 是核心配置文件。
- |－lib：支持 tomcat 软件运行的 jar 包，其中还有技术支持包，如 Servlet、jsp。
- |－logs：运行过程中的日志信息。
- |－temp：临时目录。
- |－webapp：tomcat 共享目录，单独文件不能共享，必须放到文件夹中。
- |－work：tomcat 的工作目录，就是 tomcat 把 jsp 转换为 class 文件的工作目录。

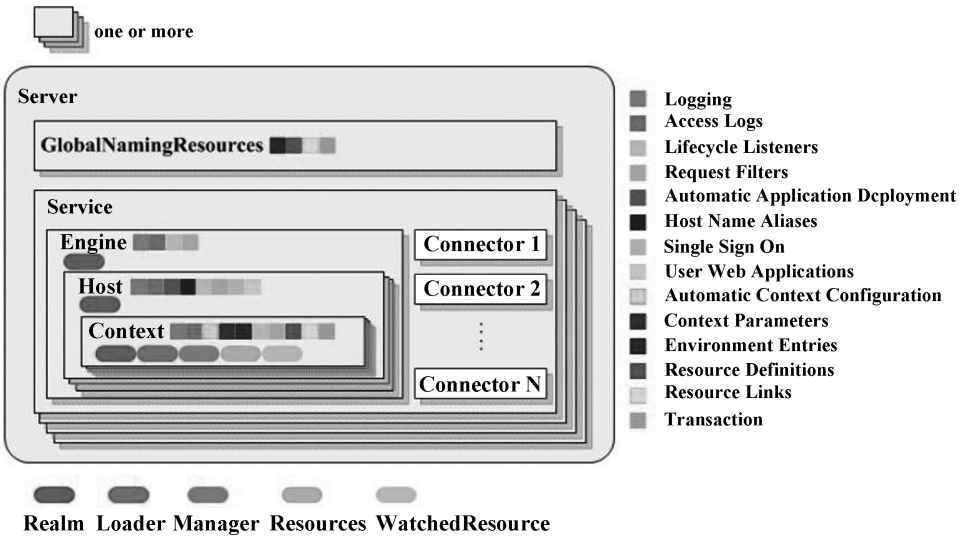


图 1.4 tomcat 的架构

Tomcat 最初是由 Sun 的软件构架师詹姆斯·邓肯·戴维森开发的。后来他帮助将其变为开源项目，并由 Sun 贡献给 Apache 软件基金会。由于大部分开源项目 O'Reilly 都会出一本相关的书，并且将其封面设计成某个动物的素描，因此他希望将此项目以一个动物的名字命名。因为他希望这种动物能够自己照顾自己，最终，他将其命名为 Tomcat(英语公猫或其他雄性猫科动物)。而 O'Reilly 出版的介绍 Tomcat 的书籍的封面也被设计成了一只公猫的形象。而 Tomcat 的 Logo 兼吉祥物也被设计成了一只公猫。目前 Tomcat 最新版本为 9.0.27-Released (alpha) Released。Apache Tomcat 7.x 是目前的开发焦点。它在汲取了 Tomcat 6.0.x 优点的基础上，实现了对于 Servlet 3.0、JSP 2.2 和 EL 2.2 等特性的支持。除此以外的改进列表如下。

Web 应用内存溢出侦测和预防增强了管理程序和服务器管理程序的安全性。一般 CSRF保护支持 web 应用中的外部内容的直接引用重构(connectors，lifecycle)及很多核心代码的全面梳理 Apache Tomcat 6.x 在汲取 Tomcat 5.5.x 优点的基础上，实现了 Servlet 2.5 和 JSP 2.1 等特性的支持。除此以外的改进列表如下。

1. 内存使用优化。

2. 更大的 IO 容量。

3. 重构聚类。

1.2.2　Apache Tomcat 5.5.x 的简单介绍

Apache Tomcat 5.5.x 和 Apache Tomcat 5.0.x 对于 Servlet 和 JSP 的支持是一样的。大量底层代码里的重大修改，带来性能的提升、稳定性的提升及整体成本。详请参照 Apache Tomcat 5.5 的更新日志。Apache Tomcat 5.0.x 在 Apache Tomcat 4.1 的基础上做了很多改动，包括：性能优化和减少垃圾回收动作，重构程序部署，通过一个可选的独立部署程序，允许在将一个 web 应用放进产品前验证和编译它基于 JMX 的服务器全面监视及 web 程序管理提高 Taglibs 的支撑能力，包括改进的数据池和 tag 插件改进平台集成性，包括 Windows 和 Unix 基于 JMX 的嵌入增强的安全管理支撑集成 session 集群文档扩充 Servlet/JSP_规范版本 Apache Tomcat 版本。

3.1/2.3_ _ _ _ 8.0.X

3.0/2.2_ _ _ _ 7.0.X

2.5/2.1_ _ _ _ 6.0.X

2.4/2.0_ _ _ _ 5.5.X

2.3/1.2_ _ _ _ 4.1.X

2.2/1.1_ _ _ _ 3.3.X

在 Tomcat 中，应用程序的部署很简单，你只需将你的 WAR 放到 Tomcat 的 webapp 目录下，Tomcat 会自动检测到这个文件，并将其解压。你在浏览器中访问这个应用的 Jsp 时，通常第一次会很慢，因为 Tomcat 要将 Jsp 转化为 Servlet 文件，然后编译。编译以后，访问将会很快。另外 Tomcat 也提供了一个应用：manager，访问这个应用需要用户名和密码，用户名和密码存储在一个 xml 文件中。通过这个应用，辅助于 Ftp，你可以在远程通过 Web 部署和撤销应用。当然本地也可以。Tomact 是用 java 写的程序，那肯定也是要 JVM 才可以运行的，你的环境变量 JAVA_HOME 一定要事先配置好，这样启动的时候它才能去找到 java。还有一个常见的错误是端口 8080 被占用了，这样的话先把占用 8080 端口的程序给找出来，然后可以把它停了。可以下载个 TcpView 小工具，直接查看使用各端口号的应用程序，可以直接关闭进程。或者也可以直接打开 CMD，用命令行来查看。先输入 netstat-ano | findstr“8080”，这样就会显示端口被进程号为 5348 的进程号占用，然后继续输入 tasklist | findstr“5348”，这样就显示出占用端口号的程序了，然后启动 window 任务管理器，找到 java.exe 这个进程，关了就行。如图 1.5 所示。

图 1.5　关闭进程

另外还有一种可能错误就是 Catalina_ Home 环境变量的设置，这个环境变量时来描述 Tomcat 的位置的，当你点击 startup. bat 启动 Tomcat 服务器的时候，会根据这个环境变量的路径去找到指定的 Tomcat。所以，如果你电脑上有几个 Tomcat 服务器分布在不同的磁盘位置的话，配置这个环境变量的时候就要小心了，因为虽然你表面上是启动不同的 Tomcat，但是实际上都是在启动 Catalina_ Home 环境变量指定路径的 Tomcat。如图 1. 6 所示。

```
<!-- A "Connector" represents an endpoint by which requests are received
     and responses are returned. Documentation at :
     Java HTTP Connector: /docs/config/http.html (blocking & non-blocking)
     Java AJP  Connector: /docs/config/ajp.html
     APR (HTTP/AJP) Connector: /docs/apr.html
     Define a non-SSL HTTP/1.1 Connector on port 8080
-->
<Connector connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="
8443"/>
<!-- A "Connector" using the shared thread pool-->
```

图 1. 6　指定路径的 Tomcat

· bin 用于放置启动和关闭 Tomcat 的可执行文件和脚本执行文件。

· conf 用于放置 Tomcat 的配置文件，其中最重要的就是里面的 server. xml 文件，该文件描述了服务器启动各种参数。

· lib 用于放置支撑 Tomcat 服务器运行的各种 jar 包。

· logs 用于放置 Tomcat 的日志记录文件，这个文件对我们平时开发排错也蛮有用的，因为当点击 startup. bat 启动 Tomcat 有报错的情况下，而且是一闪而过那种错误，这时候肯定看不到错误提示信息，那么可以来这里查看日记信息，有助于我们找到错误的原因。

· temp 这个目录是 Tomcat 本身运行产生的临时文件，普通开发者不用管。

· webapps Web 应用程序的主要发布目录，这是所有目录里面对开发者而言，最值得关注的一个目录。

· work Tomcat 的工作目录，JSP 文件翻译成的 Servlet 源文件和 class 文件放量在这里。

在 Tomcat 中配置虚拟目录 Web 应用程序指供浏览器访问的程序，通常也简称为 Web 应用。一个 Web 应用由多个静态 Web 资源和动态 Web 资源组成，如：html、css、js 文件，Jsp 文件、Java 程序、支持 jar 包、配置文件等，组成 Web 应用的这些文件通常我们会使用一个目录组织，这个目录称之为 Web 应用所在目录。Web 应用开发好后，若想供外界访问，需要把 Web 应用所在目录交给 Web 服务器管理，这个过程称之为虚拟目录的映射。

Context 表示一个 Web 应用(Context 元素在配置文件中除用于映射虚拟目录外，它还可用于为 Web 应用配置一些资源，例如：配置 Web 应用使用的数据库连接池，javamail session 等)，docBase 目录是 Web 应用所在目录，而 Path 这个路径在磁盘上是没有的，叫虚拟路径。配置好了之后，映射关系也就完成了。这样在外界访问“/Test2”虚拟目录，就相当于在访问服务器上“D:\ javaWeb”目录，这也是把 Web 应用程序提供给外界使用的过程。让 Tomcat 服务器自动管理虚拟目录的映射 Tomcat 服务器会自动管理 Webapps 目录下的所有 Web 应用，并把它映射成虚拟目录。换句话说，Tomcat 服务器 Webapps 目录中的

Web 应用，外界可以直接访问。在实际开发工作中，可以把开发好的 Web 应用直接放入 Webapps 目录下，当 tomcat 服务器启动的时候，会自动把这些 Web 应用所在目录映射成虚拟目录，映射的虚拟目录名称就是当前 Web 应用目录(文件夹)名称，直接可以供外界访问。

1.2.3　Apache 和 Tomcat 的配置

Tomcat 不仅仅是一个 Servlet 容器，它也具有传统的 Web 服务器的功能：处理 Html 页面。但是与 Apache 相比，它的处理静态 Html 的能力就不如 Apache。可以将 Tomcat 和 Apache集成到一块，让 Apache 处理静态 Html，而 Tomcat 处理 Jsp 和 Servlet. 这种集成只需要修改一下 Apache 和 Tomcat 的配置文件即可。

Tomcat 提供 Realm 支持。Realm 类似于 Unix 里面的 group。在 Unix 中，一个 group 对应着系统的一定资源，某个 group 不能访问不属于它的资源。Tomcat 用 Realm 来对不同的应用(类似系统资源)赋给不同的用户(类似 group)。没有权限的用户则不能访问这个应用。Tomcat 提供三种 Realm：(1)JDBCRealm，这个 Realm 将用户信息存在数据库里，通过 JDBC 获得用户信息来进行验证。(2)JNDIRealm，用户信息存在基于 LDAP 的服务器里，通过 JNDI 获取用户信息。(3)MemoryRealm，用户信息存在一个 xml 文件里面，上面讲的 manager 应用验证用户时即使用此种 Realm。通过 Realm 可以方便地对访问某个应用的客户进行验证。

在 Tomcat4 中，你还可以利用 Servlet2.3 提供的事件监听器功能，来对你的应用或者 Session 实行监听。Tomcat 也提供其他的一些特征，如与 SSL 集成到一块，实现安全传输。还有 Tomcat 也提供 JNDI 支持，这与那些 J2EE 应用服务器提供的是一致的。说到这里要介绍一下通常所说的应用服务器(如 WebLogic)与 Tomcat 有何区别。应用服务器提供更多的 J2EE 特征，如 EJB、JMS、JAAS 等，同时也支持 Jsp 和 Servlet，而 Tomcat 功能则没有那么强大，它不提供 EJB 等支持。但如果与 JBoss(一个开源的应用服务器)集成到一块，则可以实现 J2EE 的全部功能。既然应用服务器具有 Tomcat 的功能，那么 Tomcat 有没有存在的必要呢？事实上，很多中小应用不需要采用 EJB 等技术，Jsp 和 Servlet 已经足够，这时如果用应用服务器就有些浪费了。而 Tomcat 短小精悍，配置方便，能满足中小应用的需求，这种情况下自然会选择 Tomcat。

基于 Tomcat 的开发其实主要是 Jsp 和 Servlet 的开发，开发 Jsp 和 Servlet 非常简单，你可以用普通的文本编辑器或者 IDE，然后将其打包成 WAR 即可。这里要提到另外一个工具 Ant，Ant 也是 Jakarta 中的一个子项目，它所实现的功能类似于 Unix 中的 make。你需要写一个 build.xml 文件，然后运行 Ant 就可以完成 xml 文件中定义的工作，这个工具对于一个大的应用来说非常好，只需在 xml 中写很少的东西就可以将其编译并打包成 WAR。在很多应用服务器的发布中都包含了 Ant。另外，在 Jsp1.2 中，可以利用标签库实现 Java 代码与 Html 文件的分离，使 Jsp 的维护更方便。

Tomcat 也可以与其他一些软件集成起来实现更多的功能。如与上面提到的 JBoss 集成

起来开发 EJB，与 Cocoon（Apache 的另外一个项目）集成起来开发基于 Xml 的应用，与 OpenJMS 集成起来开发 JMS 应用，除了提到的这几种，可以与 Tomcat 集成的软件还有很多。

启动内存参数的配置

tomcat/bin/catalina. bat 如果是 linux 就是 catalina. sh

在 rem 的后面增加如下参数

set JAVA_OPTS = -Xms256m -Xmx256m -XX：MaxPermSize = 64m

修改 Tomcat 的 JDK 目录

打开 tomcat/bin/catalina. bat

在最后一个 rem 后面增加

set JAVA_HOME = C：\ Program Files \ Java \ jdk1. 6. 0

增加虚拟目录

/tomcat/conf/server. xml

配置系统管理（Admin Web Application）

大多数商业化的 JavaEE 服务器都提供一个功能强大的管理界面，且大都采用易于理解的 Web 应用界面。Tomcat 按照自己的方式，同样提供一个成熟的管理工具，并且丝毫不逊于那些商业化的竞争对手。Tomcat 的 Admin Web Application 最初在 4. 1 版本时出现，当时的功能包括管理 context、data source、user 和 group 等。当然也可以管理像初始化参数 user、group、role 的多种数据库管理等。在后续的版本中，这些功能将得到很大的扩展，但现有的功能已经非常实用了。

Admin Web Application 被定义在自动部署文件：Tomcat/webapps/admin. xml。必须编辑这个文件，以确定 Context 中的 docBase 参数是绝对路径。也就是说，Tomcat/webapps/admin. xml 的路径是绝对路径。作为另外一种选择，也可以删除这个自动部署文件，而在 server. xml 文件中建立一个 Admin Web Application 的 context，效果是一样的。不能管理 Admin Web Application 这个应用，换而言之，除了删除 Tomcat/webapps/admin. xml，可能什么都做不了。

如果使用 UserDatabaseRealm（默认），将需要添加一个 user 以及一个 role 到 Tomcat/conf/tomcat-users. xml 文件中。你编辑这个文件，添加一个名叫“admin”的 role 到该文件中，如下所示。

```
<role name = "admin"/ >
```

同样需要有一个用户，并且这个用户的角色是“admin”。像存在的用户那样，添加一个用户（改变密码使其更加安全）。

```
<user name = "admin"password = "deep_dark_secret"roles = "admin"/ >
```

当完成这些步骤后，请重新启动 Tomcat，访问 http://localhost：8080/admin，将看到一个登录界面。Admin Web Application 采用基于容器管理的安全机制，并采用了 Jakarta Struts 框架。一旦作为“admin”角色的用户登录管理界面，将能够使用这个管理界面配置

Tomcat。

配置应用管理

Manager Web Application 让你通过一个比 Admin Web Application 更为简单的用户界面，执行一些简单的 Web 应用任务。

Manager Web Application 被定义在一个自动部署文件中。

Tomcat/webapps/manager. xml。

必须编辑这个文件，以确保 context 的 docBase 参数是绝对路径，也就是说 CATALINA_HOME/server/webapps/manager 的绝对路径。

如果使用的是 UserDatabaseRealm，那么需要添加一个角色和一个用户到 Tomcat/conf/tomcat-users. xml 文件中。接下来，编辑这个文件，添加一个名为“manager”的角色到该文件中。

```
< role name = "manager" >
```

同样需要有一个角色为“manager”的用户。像已经存在的用户那样，添加一个新用户（改变密码使其更加安全）。

```
< user name = "manager"password = "deep_dark_secret" roles = "manager"/ >
```

然后重新启动 Tomcat，访问 http://localhost/manager/list，将看到一个很朴素的文本型管理界面，或者访问 http://localhost/manager/html/list，将看到一个 HMTL 的管理界面。不管是哪种方式都说明你的 Manager Web Application 现在已经启动了。

Manager application 可以在没有系统管理特权的基础上，安装新的 Web 应用，以用于测试。如果我们有一个新的 Web 应用位于/home/user/hello 下，并且想把它安装到 /hello 下，为了测试这个应用，可以这么做，在第一个文件框中输入“/hello”（作为访问时的 path），在第二个文本框中输入“file：/home/user/hello”（作为 Config URL）。

Manager application 还允许停止、重新启动、移除以及重新部署一个 Web 应用。停止一个应用使其无法被访问，当有用户尝试访问这个被停止的应用时，将看到一个 503 的错误——“503 – This application is not currentlyavailable”。移除一个 Web 应用，只是指从 Tomcat的运行拷贝中删除了该应用，如果重新启动 Tomcat，被删除的应用将再次出现（也就是说，移除并不是指从硬盘上删除）。

1. 2. 4　部署一个 Web 应用

有两个办法可以在系统中部署 Web 服务。

（1）拷贝 WAR 文件或者 Web 应用文件夹（包括该 Web 的所有内容）到 $Tomcat/webapps目录下。

（2）为 Web 服务建立一个只包括 context 内容的 XML 片断文件，并把该文件放到 $Tomcat/webapps 目录下。这个 Web 应用本身可以存储在硬盘上的任何地方。如果有一个 WAR 文件，想部署它，则只需要把该文件简单地拷贝到 Tomcat/webapps 目录下即可，文件必须以“. war”作为扩展名。一旦 Tomcat 监听到这个文件，它将（缺省的）解开该文件包

作为一个子目录，并以 WAR 文件的文件名作为子目录的名字。接下来，Tomcat 将在内存中建立一个 context，就好像在 server. xml 文件里建立一样。当然，其他必需的内容，将从 server. xml 中的 DefaultContext 获得。

部署 Web 应用的另一种方式是写一个 Context XML 片断文件，然后把该文件拷贝到 Tomcat/webapps 目录下。一个 Context 片断并非一个完整的 XML 文件，而只是一个 context 元素，以及对该应用的相应描述。这种片断文件就像是从 server. xml 中切取出来的 context 元素一样，所以这种片断被命名为“context 片断”。举个例子，如果我们想部署一个名叫 MyWebApp. war 的应用，该应用使用 realm 作为访问控制方式，我们可以使用下面这个片断。

```
<!--
Context fragment for deploying MyWebApp. war
-->
demo” docBase = "webapps/MyWebApp. war"
debug = "0" privileged = "true" >
className = "org. apache. catalina. realm. UserDatabaseRealm"
resourceName = "UserDatabase"/ >
</Context >
```

把该片断命名为“MyWebApp. xml”，然后拷贝到 Tomcat/webapps 目录下。这种 context 片断提供了一种便利的方法来部署 Web 应用，不需要编辑 server. xml，除非想改变缺省的部署特性，安装一个新的 Web 应用时不需要重启动 Tomcat。配置虚拟主机(Virtual Hosts)关于 server. xml 中“Host”这个元素，只有在设置虚拟主机时才需要修改。虚拟主机是一种在一个 Web 服务器上服务多个域名的机制，对每个域名而言，都好像独享了整个主机。实际上，大多数的小型商务网站都是采用虚拟主机实现的，这主要是因为虚拟主机能直接连接到 Internet 并提供相应的带宽，以保障合理的访问响应速度，另外虚拟主机还能提供一个稳定的固定 IP。

基于名字的虚拟主机可以被建立在任何 Web 服务器上，建立的方法就是通过在域名服务器(DNS)上建立 IP 地址的别名，并且告诉 Web 服务器把去往不同域名的请求分发到相应的网页目录。

在 Tomcat 中使用虚拟主机，需要设置 DNS 或主机数据。为了测试，为本地 IP 设置一个 IP 别名就足够了，接下来，你需要在 server. xml 中添加几行内容，如下所示。

```
<Server port = "8005" shutdown = "SHUTDOWN" debug = "0" > <Service name = "Tomcat
- Standalone" <Connector className = "org.apache.coyote.tomcat4.CoyoteConnector"
port = "8080" minProcessors = "5" maxProcessors = "75"
enableLookups = "true" redirectPort = "8443"/ >
<Connector className = "org.apache.coyote.tomcat4.CoyoteConnector"
port = "8443" minProcessors = "5" maxProcessors = "75"
```

```
acceptCount = "10" debug = "0" scheme = "https" secure = "true"/ >
 < Factory className = "org.apache.coyote.tomcat4.CoyoteServerSocketFactory"
clientAuth = "false" protocol = "TLS" / >
 < /Connector >
 < Engine name = "Standalone" defaultHost = "localhost" debug = "0" >
 < ! - -  This Host is the default Host  - - >
 < Host name = "localhost" debug = "0" appBase = "webapps"
unpackWARs = "true" autoDeploy = "true" >
 < Context path = "" docBase = "ROOT" debug = "0"/ >
 < Context path = "/orders" docBase = "/home/ian/orders" debug = "0"
reloadable = "true" crossContext = "true" >
 < /Context >
 < /Host >
 < Host name = "www.example. com" appBase = "/home/example/webapp" >
 < Context path = "" docBase = "."/ >
```

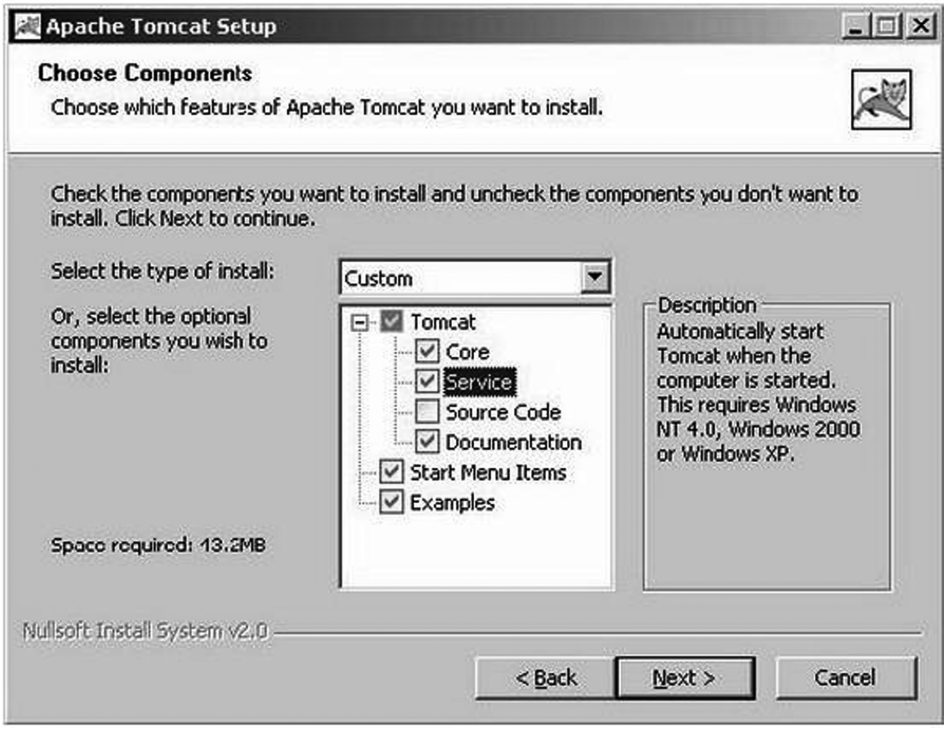


图 1.7　Tomcat 的设置

```
Tomcat < /Host >
 < /Engine >
```

</Service>

</Server>

Tomcat 的 server.xml 文件，在初始状态下，只包括一个虚拟主机，但是它容易被扩充到支持多个虚拟主机。在前面的例子中展示的是一个简单的 server.xml 版本，其中粗体部分就是用于添加一个虚拟主机。每一个 Host 元素必须包括一个或多个 context 元素，所包含的 context 元素中必须有一个是默认的 context，这个默认的 context 的显示路径应该为空（例如，path = “”）。

配置基础验证（Basic Authentication）

容器管理验证方法控制着当用户访问受保护的 Web 应用资源时，如何进行用户的身份鉴别。当一个 Web 应用使用了 Basic Authentication（BASIC 参数在 web.xml 文件中 auto-method 元素中设置），而有用户访问受保护的 Web 应用时，Tomcat 将通过 HTTP Basic Authentication方式，弹出一个对话框，要求用户输入用户名和密码。在这种验证方法中，所有密码将被以 64 位的编码方式在网络上传输。

注意：使用 Basic Authentication 通常被认为是不安全的，因为它没有强健的加密方法，除非在客户端和服务器端都使用 HTTPS 或者其他密码加密码方式（比如，在一个虚拟私人网络中）。若没有额外的加密方法，网络管理员将能够截获（或滥用）用户的密码。但是，如果是刚开始使用 Tomcat，或者你想在你的 Web 应用中测试一下基于容器的安全管理，Basic Authentication 还是非常易于设置和使用的。只需要添加和两个元素到 Web 应用的 Web.xml 文件中，并且在 Tomcat/conf/tomcat-users.xml 文件中添加适当的和即可，然后重新启动 Tomcat。

配置单点登录（Single Sign-On）

一旦设置了 realm 和验证的方法，就需要进行实际的用户登录处理。一般说来，对用户而言登录系统是一件很麻烦的事情，必须尽量减少用户登录验证的次数。作为缺省的情况，当用户第一次请求受保护的资源时，每一个 Web 应用都会要求用户登录。如果运行了多个 Web 应用，并且每个应用都需要进行单独的用户验证，那这看起来就有点像在跟用户搏斗。用户们不知道怎样才能把多个分离的应用整合成一个单独的系统，所有用户也就不知道他们需要访问多少个不同的应用，只是很迷惑，为什么总要不停地登录。

Tomcat 4 的“single sign-on”特性允许用户在访问同一虚拟主机下所有 Web 应用时，只需登录一次。为了使用这个功能，只需要在 Host 上添加一个 SingleSignOn Valve 元素即可，如下所示。

<Valve className = "org.apache.catalina.authenticator.SingleSignOn"

debug = "0"/>

在 Tomcat 初始安装后，server.xml 的注释里面包括 SingleSignOn Valve 配置的例子，只需要去掉注释，即可使用。那么，任何用户只要登录过一个应用，则对于同一虚拟主机下的所有应用同样有效。

使用 single sign-on valve 有一些重要的限制。

（1）value 必须被配置和嵌套在相同的 Host 元素里，并且所有需要进行单点验证的 Web 应用(必须通过 context 元素定义)都位于该 Host 下。

（2）包括共享用户信息的 realm 必须被设置在同一级 Host 中或者嵌套之外。

（3）不能被 context 中的 realm 覆盖。

（4）使用单点登录的 Web 应用最好使用一个 Tomcat 的内置的验证方式(被定义在 Web. xml 中的中)，这比自定义的验证方式强，Tomcat 内置的验证方式包括 basic、digest、form 和 client-cert。

（5）如果你使用单点登录，还希望集成一个第三方的 Web 应用到你的网站中来，并且这个新的 Web 应用使用它自己的验证方式，而不使用容器管理安全，那你基本上就没招了。用户每次登录原来所有应用时需要登录一次，并且在请求新的第三方应用时还得再登录一次。

（6）单点登录需要使用 cookies。

1.2.5　配置用户定制目录

一些站点允许个别用户在服务器上发布网页。例如，一所大学的学院可能想给每一位学生一个公共区域，或者是一个 ISP 希望给一些 Web 空间给他的客户，但这又不是虚拟主机。在这种情况下，一个典型的方法就是在用户名前面加一个特殊字符(~)，作为每位用户的网站，比如：

http://www. cs. myuniversity. edu/ ~username 提供两种方法在主机上映射这些个人网站，主要使用一对特殊的 Listener 元素。Listener 的 className 属性应该是 org. apache. catalina. startup. UserConfig，userClass 属性应该是几个映射类之一。如果电脑系统是 Unix，它将有一个标准的/etc/passwd 文件，该文件中的账号能够被运行中的 Tomcat 很容易地读取，该文件指定了用户的主目录，使用 PasswdUserDatabase 映射类。

http://members. mybigisp. com/ ~username

Tomcat

```
< Listener className = "org. apache. catalina. startup. UserConfig"
directoryName = "public_html"
userClass = "org. apache. catalina. startup. PasswdUserDatabase"/ >
```

Web 文件需要放置在像/home/users/ian/public_ html 或者/users/jbrittain/public_html 一样的目录下面。当然你也可以改变 public_ html 到其他任何子目录下。

实际上，这个用户目录根本不一定需要位于用户主目录下面。如果你没有一个密码文件，但你又想把一个用户名映射到公共的像/home 一样目录的子目录里面，则可以使用 HomesUserDatabase 类。

```
< Listener className = "org. apache. catalina. startup. UserConfig"
directoryName = "public_html" homeBase = "/home"
userClass = "org. apache. catalina. startup. HomesUserDatabase"/ >
```

这样一来，Web 文件就可以位于像/home/ian/public_html 或者/home/jasonb/public_html 一样的目录下。这种形式对 Windows 而言更加有利，你可以使用一个像 C:\ home 这样的目录。

这些 Listener 元素，如果出现，则必须在 Host 元素里面，而不能在 context 元素里面，因为它们都用应用于 Host 本身。

1.2.6 在 Tomcat 中使用 CGI 脚本

Tomcat 主要是作为 Servlet/JSP 容器，但它也有许多传统 web 服务器的性能。支持通用网关接口(Common Gateway Interface，即 CGI)就是其中之一，CGI 提供一组方法在响应浏览器请求时运行一些扩展程序。CGI 之所以被称为通用，是因为它能在大多数程序或脚本中被调用，包括：Perl、Python、awk、Unix shell scripting 等，甚至包括 Java。不会把一个 Java 应用程序当作 CGI 来运行，毕竟这样太过原始。一般而言，开发 Servlet 总要比 CGI 具有更好的效率，因为当用户点击一个链接或一个按钮时，不需要从操作系统层开始进行处理。

Tomcat 包括一个可选的 CGI Servlet，允许你运行遗留下来的 CGI 脚本。

为了使 Tomcat 能够运行 CGI，必须做以下几件事。

1. 把 servlets-cgi. renametojar (在 CATALINA_HOME/server/lib/目录下)改名为 servlets-cgi. jar。处理 CGI 的 servlet 应该位于 Tomcat 的 CLASSPATH 下。

2. 在 Tomcat 的 Tomcat/conf/web. xml 文件中，把关于 < servlet-name > CGI 的那段的注释去掉(默认情况下，该段位于第 241 行)。

3. 同样，在 Tomcat 的 Tomcat/conf/web. xml 文件中，把关于对 CGI 进行映射的那段的注释去掉(默认情况下，该段位于第 299 行)。注意，这段内容指定了 HTML 链接到 CGI 脚本的访问方式。

4. 可以把 CGI 脚本放置在 WEB-INF/cgi 目录下(注意，WEB-INF 是一个安全的地方，你可以把一些不想被用户看见或基于安全考虑不想暴露的文件放在此处)，或者也可以把 CGI 脚本放置在 context 下的其他目录下，并为 CGI Servlet 调整 cgiPathPrefix 初始化参数。这就指定的 CGI Servlet 的实际位置，且不能与上一步指定的 URL 重名。

5. 重新启动 Tomcat，你的 CGI 就可以运行了。

在 Tomcat 中，CGI 程序缺省放置在 WEB-INF/cgi 目录下，正如前面所提示的那样，WEB-INF 目录受保护的，通过客户端的浏览器无法窥探到其中内容，所以对于放置含有密码或其他敏感信息的 CGI 脚本而言，这是一个非常好的地方。为了兼容其他服务器，尽管你也可以把 CGI 脚本保存在传统的/cgi-bin 目录，但要知道，在这些目录中的文件有可能被网上好奇的冲浪者看到。另外，在 Unix 中，请确定运行 Tomcat 的用户有执行 CGI 脚本的权限。

1.2.7 改变 Tomcat 中的 JSP 编译器

在 Tomcat 4.1(或更高版本)，JSP 的编译由包含在 Tomcat 里面的 Ant 程序控制器直接

执行。这听起来有一点点奇怪，但这正是 Ant 有意为之的一部分，有一个 API 文档指导开发者在没有启动一个新的 JVM 的情况下使用 Ant。这是使用 Ant 进行 Java 开发的一大优势。另外，这也意味着你现在能够在 Ant 中使用任何 Java 支持的编译方式，这里有一个关于 Apache Ant 使用手册的 Java page 列表。使用起来是容易的，因为你只需要在元素中定义一个名字叫“compiler”，并且在 value 中有一个支持编译的编译器名字，示例如下。

```
< servlet >
< servlet-name > jsp < /servlet-name >
< servlet-class >
org. apache. jasper. servlet. JspServlet
< /servlet-class >
< init-param >
< param-name > logVerbosityLevel < /param-name >
< param-value > WARNING < /param-value >
< /init-param >
< init-param >
< param-name > compiler < /param-name >
jikes
< /init-param >
< load-on-startup > 3 < /load-on-startup >
< /servlet >
```

当然，给出的编译器必须已经安装在你的系统中，并且 CLASSPATH 可能需要设置，那取决于你选择的是何种编译器。

限制特定主机访问

有时，可能想限制对 Tomcat web 应用的访问，比如，希望只有指定的主机或 IP 地址可以访问应用。这样一来，就只有那些指定的客户端可以访问服务的内容了。为了实现这种效果，Tomcat 提供了两个参数供你配置：RemoteHostValve 和 RemoteAddrValve。

通过配置这两个参数，可以让你过滤来自请求的主机或 IP 地址，并允许或拒绝哪些主机/IP。与之类似的，在 Apache 的 httpd 文件里有对每个目录的允许/拒绝指定。

可以把 Admin Web application 设置成只允许本地访问，设置如下。

```
< Context path = "/path/to/secret_files"… > < Valve className =
"org. apache. catalina. valves. RemoteAddrValve"
allow = "127. 0. 0. 1"deny = ""/ >
< /Context >
```

如果没有给出允许主机的指定，那么与拒绝主机匹配的主机就会被拒绝，除此之外的都是允许的。

tomcat 目录结构

/bin：存放 windows 或 Linux 平台上启动和关闭 Tomcat 的脚本文件。

/conf：存放 Tomcat 服务器的各种全局配置文件，其中最重要的是 server. xml 和 web. xml。

/doc：存放 Tomcat 文档。

/server：包含三个子目录，classes、lib 和 Web webapps。

/server/lib：存放 Tomcat 服务器所需的各种 JAR 文件。

/server/webapps：存放 Tomcat 自带的两个 Web 应用 admin 应用和 manager 应用。

/common/lib：存放 Tomcat 服务器以及所有 Web 应用都可以访问的 jar 文件。

/shared/lib：存放所有 Web 应用都可以访问的 jar 文件(但是不能被 Tomcat 服务器访问)。

/logs：存放 Tomcat 执行时的日志文件。

/src：存放 Tomcat 的源代码。

/webapps：Tomcat 的主要 Web 发布目录，默认情况下把 Web 应用文件放于此目录。

/work：存放 JSP 编译后产生的 class 文件。

1.2.8 Tomcat 的安全部署

Tomcat 是一个世界上广泛使用的支持 jsp 和 servlets 的 Web 服务器。它在 Java 上运行时能够很好地运行并支持 Web 应用部署。会因为设置不当，造成灾难性的后果。在 Tomcat 默认安装，Tomcat 作为一个系统服务运行，如果没有将其作为系统服务运行，几乎所有 Web 服务器管理员都是缺省地将其以 Administrator 权限运行。这两种方式都允许 Java 运行时访问 Windows 系统下任意文件夹中的任何文件。缺省情况下，Java 运行时授予安全权限。当 Tomcat 以系统管理员身份或作为系统服务运行时，Java 运行取得了系统用户或系统管理员所具有的全部权限。这样一来，Java 运行时就取得了所有文件夹中所有文件的全部权限。并且 Servlets(JSP 在运行过程中要转换成 Servlets)取得了同样的权限。所以 Java 代码可以调用 Java SDK 中的文件 API、列出文件夹中的全部文件、删除任何文件，最大的危险在于以系统权限运行一个程序。当任一 Servlets 含有如下代码：

```
b4ae04fd6dYsJkr5    Runtime rt = Runtime. getRuntime( );
rt. exec(":\SomeDirectory\SomeUnsafePRogram. exe")[2]
```

其服务是以 system 权限启动。根据权限最小安全原则，降低了脚本所获取的操作本地系统权限。此操作如下。

新建一个账户

1. 用“ITOMCAT_计算机名”建立一个普通用户。
2. 为其设置一个密码。
3. 保证“密码永不过期”(PassWord Never Expires)被选中。

修改 Tomcat 安装文件夹的访问权限

1. 选定环境参数 CATALINA_HOME 或 TOMCAT_HOME 指向的 Tomcat 安装文件夹。
2. 为“ITOMCAT_计算机名”用户赋予读、写、执行的访问权限。
3. 为“ITOMCAT_计算机名”用户赋予对 WebApps 文件夹的只读访问权限。

4. 如果某些 Web 应用程序需要写访问权限，单独为其授予对那个文件夹的写访问权限。

Tomcat 作为系统服务

1. 到“控制面板”，选择“管理工具”，然后选择“服务”。
2. 找到 Tomcat：比如 Apache Tomcat. exe 等，打开其“属性”。
3. 选择其“登录”(Log)标签。
4. 选择“以…登录”(Log ON Using)选项。
5. 键入新建的“ITOMCAT_计算机名”用户作为用户名。
6. 输入密码。
7. 重启机器。

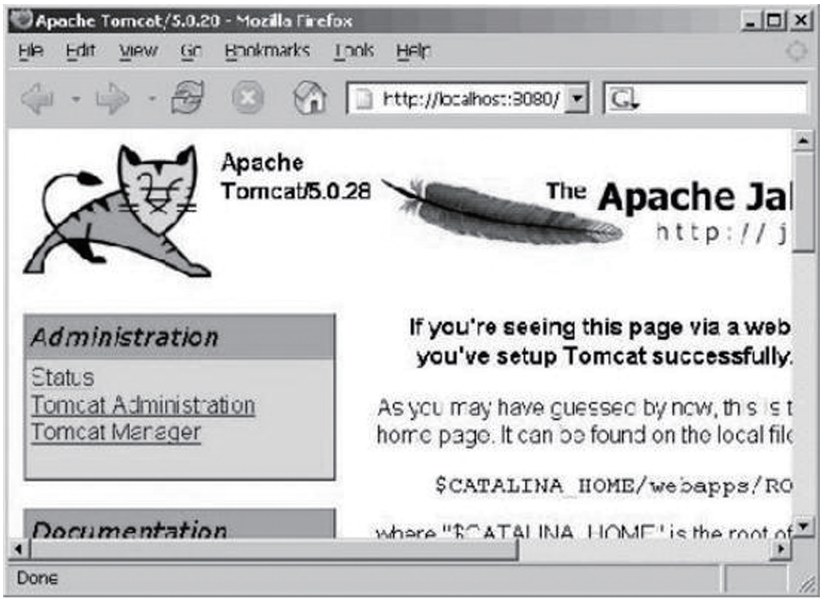


图 1.8　Tomcat 的登录界面

Tomcat 在 DOS 窗口下运行 Tomcat 步骤

1. 在“开始”按钮的“运行”框中键入 CMD 以打开一个 DOS 窗口。
2. 键入“RunAs/user：ITOMCAT_计算机名 CMD. exe”命令。
3. 在询问“ITOMCAT_计算机名”用户的密码时输入设置的密码。
4. 这将打开一个新的 DOS 窗口。
5. 在新开的 DOS 窗口中，转换到 Tomcat 的 bin 文件夹内。
6. 键入“catalina run”命令。
7. 关闭第一个 DOS 窗口。

设置一下程序

CMD. EXE NET. EXE ATTRIB. EXE At. EXE NET1. EXE FTP. EXE TELNET. EXE COMMAND. COM CAcls. EXEnetstat. exe；system 全部权限，其他用户无权限。

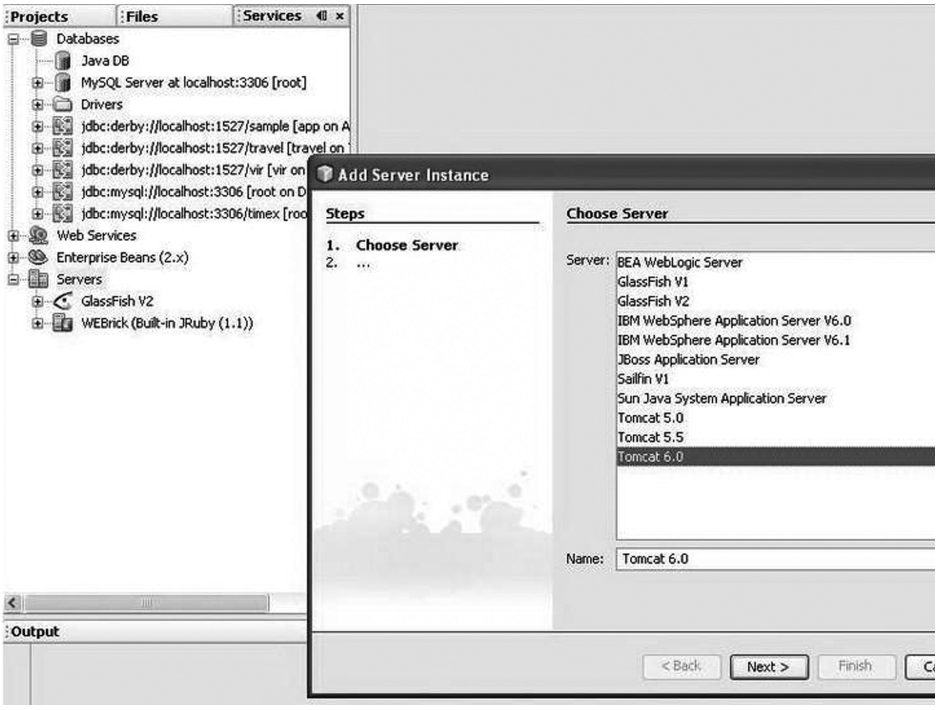


图 1.9　程序设置

Tomcat 第一行是以前默认存在的，第二行是新增的。

path = "" docBase = "ROOT" debug = "0" reloadable = "true" >

jsp/a" reloadable = "true" docBase = "E:\workplace\www. java2000. net\WebContent" / >

使用默认配置的 tomcat，另外虚拟目录也可这样设置。

test" docBase = "webContent" reloadable = "true"/ >

因为默认情况下，tomcat 启动过程中配置虚拟目录的时候会从 webapps 目录下查找 webContent 应用。

这样配置好了，即使以后从一台服务器移植到另一台服务器，不做任何修改也能运行起来。

GET 方式 URL 乱码问题解决

打开 tomcat/conf/server. xml，查找下面这部分，在最后增加一段代码就可以了。

port = "80" maxHttpHeaderSize = "8192"

. . . . . .

URIEncoding = "UTF - 8" useBodyEncodingForURI = "true"

. . . . . .

/ >

其中的 UTF-8 请根据你的需要自己修改，比如 GBK。

虚拟主机配置文件

tomcat/conf/server. xml

```
<!-- 默认的主机 -->
name = "localhost" appBase = "webapps"
unpackWARs = "true" autoDeploy = "true"
xmlValidation = "false" xmlNamespaceAware = "false" >
<Context path = "" docBase = "ROOT" debug = "0" reloadable = "true" > </Context >
...
</host >
<!-- 以下是新增的虚拟主机 -->
<Host name = "www. java2000. net" appBase = "webapps"
unpackWARs = "true" autoDeploy = "true"
xmlValidation = "false" xmlNamespaceAware = "false" >
<Context path = "" docBase = "d:/www. java2000. net" debug = "0" reloadable = "true" >
</Context >
<!-- 虚拟目录 -->
count" docBase = "d:/counter. java2000. net" debug = "0" reloadable = "true" >
</Host >
<Host name = "java2000. net" appBase = "webapps"
unpackWARs = "true" autoDeploy = "true"
xmlValidation = "false" xmlNamespaceAware = "false" >
<Context path = "" docBase = "d:/www. java2000. net" debug = "0" reloadable = "true" >
</Context >
<Context path = "/count" docBase = "d:/counter. java2000. net" debug = "0" reloadable
= "true" > </Context >
</Host >
```

1.3　HTTP 协议

超文本传输协议(HTTP，HyperText Transfer Protocol)是互联网上应用最为广泛的一种网络协议。所有的 WWW 文件都必须遵守这个标准。设计 HTTP 最初的目的是为了提供一种发布和接收 HTML 页面的方法。1960 年美国人 Ted Nelson 构思了一种通过计算机处理文本信息的方法，并称之为超文本(hypertext)，这成为 HTTP 超文本传输协议标准架构的发展根基。Ted Nelson 组织协调万维网协会(World Wide Web Consortium)和互联网工程工作小组(Internet Engineering Task Force)共同合作研究，最终发布了一系列的 RFC，其中著名的 RFC 2616 定义了 HTTP 1.1。

1.3.1　技术架构

HTTP 是一个客户端和服务器端请求和应答的标准(TCP)。客户端是终端用户，服务

器端是网站。通过使用 Web 浏览器、网络爬虫或者其他的工具，客户端发起一个到服务器上指定端口(默认端口为 80)的 HTTP 请求。我们称这个客户端叫用户代理(user agent)。应答的服务器上存储着(一些)资源，比如 HTML 文件和图像。我们称这个应答服务器为源服务器(origin server)。在用户代理和源服务器中间可能存在多个中间层，比如代理、网关，或者隧道(tunnels)。尽管 TCP/IP 协议是互联网上最流行的应用，HTTP 协议并没有规定必须使用它和(基于)它支持的层。事实上，HTTP 可以在任何其他互联网协议上，或者在其他网络上实现。HTTP 只假定(其下层协议提供)可靠的传输，任何能够提供这种保证的协议都可以被其使用。如图 1.10 所示。

图 1.10 HTTP 的默认设置

通常，由 HTTP 客户端发起一个请求，建立一个到服务器指定端口(默认是 80 端口)的 TCP 连接。HTTP 服务器则在那个端口监听客户端发送过来的请求。一旦收到请求，服务器(向客户端)发回一个状态行，比如“HTTP/1.1 200 OK”，和(响应的)消息，消息的消息体可能是请求的文件、错误消息，或者其他一些信息。如图 1.11 所示。

图 1.11 返回的状态行信息

HTTP 使用 TCP 而不是 UDP 的原因在于(打开)一个网页必须传送很多数据，而 TCP 协议提供传输控制，按顺序组织数据，和错误纠正。通过 HTTP 或者 HTTPS 协议请求的资源由统一资源标示符(Uniform Resource Identifiers)(或者，更准确一些，URLs)来标识。

1.3.2　协议功能

HTTP 协议(HyperText Transfer Protocol，超文本传输协议)是用于从 WWW 服务器传输超文本到本地浏览器的传输协议。它可以使浏览器更加高效，使网络传输减少。它不仅保证计算机正确快速地传输超文本文档，还确定传输文档中的哪一部分，以及哪部分内容首先显示(如文本先于图形)等。

HTTP 是客户端浏览器或其他程序与 Web 服务器之间的应用层通信协议。在 Internet 上的 Web 服务器上存放的都是超文本信息，客户机需要通过 HTTP 协议传输所要访问的超文本信息。HTTP 包含命令和传输信息，不仅可用于 Web 访问，也可以用于其他因特网/内联网应用系统之间的通信，从而实现各类应用资源超媒体访问的集成。

我们在浏览器的地址栏里输入的网站地址叫作 URL(Uniform Resource Locator，统一资源定位符)。就像每家每户都有一个门牌地址一样，每个网页也都有一个 Internet 地址。当你在浏览器的地址框中输入一个 URL 或是单击一个超级链接时，URL 就确定了要浏览的地址。浏览器通过超文本传输协议(HTTP)，将 Web 服务器上站点的网页代码提取出来，并翻译成漂亮的网页。

图 1.12　根据 URL 翻译而来的网页

1.3.3　协议基础

HTTP(HyperText Transport Protocol)是超文本传输协议的缩写，它用于传送 WWW 方式的数据，关于 HTTP 协议的详细内容请参考 RFC2616。HTTP 协议采用了请求/响应模型。客户端向服务器发送一个请求，请求头包含请求的方法、URL、协议版本以及包含请求修饰符、客户信息和内容的类似于 MIME 的消息结构。服务器以一个状态行作为响应，响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。

通常 HTTP 消息包括客户机向服务器的请求消息和服务器向客户机的响应消息。这两种类型的消息由一个起始行、一个或者多个头域、一个指示头域结束的空行和可选的消息体组成。HTTP 的头域包括通用头、请求头、响应头和实体头四个部分。每个头域由一个

域名、冒号(:)和域值三部分组成。域名是大小写无关的，域值前可以添加任何数量的空格符，头域可以被扩展为多行，在每行开始处，使用至少一个空格或制表符。

1. 通用头域

通用头域包含请求和响应消息都支持的头域，通用头域包含 Cache-Control、Connection、Date、Pragma、Transfer-Encoding、Upgrade、Via。对通用头域的扩展要求通信双方都支持此扩展，如果存在不支持的通用头域，一般将会作为实体头域处理。下面简单介绍几个在 UPnP 消息中使用的通用头域。

(1) Cache-Control 头域

Cache-Control 指定请求和响应遵循的缓存机制。在请求消息或响应消息中设置 Cache-Control 并不会修改另一个消息处理过程中的缓存处理过程。请求时的缓存指令包括 no-cache、no-store、max-age、max-stale、min-fresh、only-if-cached，响应消息中的指令包括 public、private、no-cache、no-store、no-transform、must-revalidate、proxy-revalidate、max-age。各个消息中的指令含义如下。

Public：指示响应可被任何缓存区缓存。

Private：指示对于单个用户的整个或部分响应消息，不能被共享缓存处理。这允许服务器仅仅描述当用户的部分响应消息，此响应消息对于其他用户的请求无效

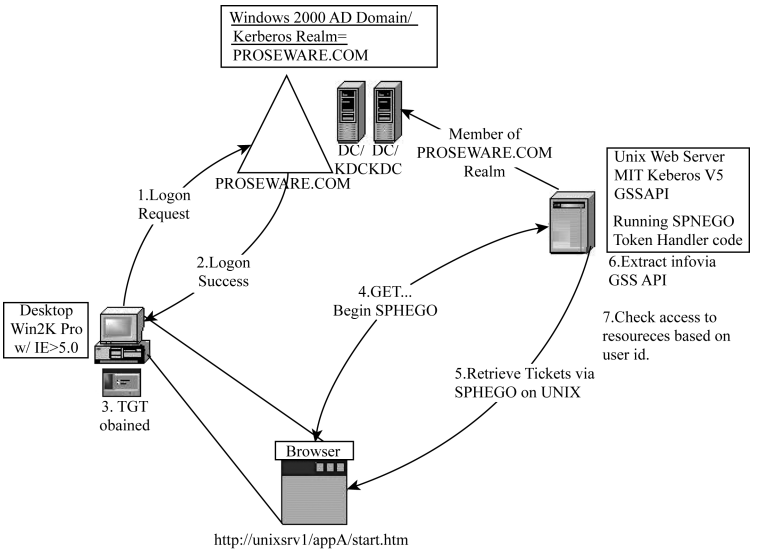


图 1.13　协议基础架构

no-cache：指示请求或响应消息不能缓存。

no-store：用于防止重要的信息被无意的发布。在请求消息中发送将使得请求和响应消息都不使用缓存。

max-age：指示客户机可以接收生存期不大于指定时间(以秒为单位)的响应。

min-fresh：指示客户机可以接收响应时间小于当前时间加上指定时间的响应。

max-stale：指示客户机可以接收超出超时期间的响应消息。如果指定 max-stale 消息的

值，那么客户机可以接收超出超时期指定值之内的响应消息。

HTTP Keep-Alive

Keep-Alive 功能使客户端到服务器端的连接持续有效，当出现对服务器的后继请求时，Keep-Alive 功能避免了建立或者重新建立连接。市场上的大部分 Web 服务器，包括 iPlanet、IIS 和 Apache，都支持 HTTP Keep-Alive。对于提供静态内容的网站来说，这个功能通常很有用。但是，对于负担较重的网站来说，这里存在另外一个问题：虽然为客户保留打开的连接有一定的好处，但它同样影响了性能，因为在处理暂停期间，本来可以释放的资源仍旧被占用。当 Web 服务器和应用服务器在同一台机器上运行时，Keep-Alive 功能对资源利用的影响尤其突出。

KeepAliveTime 值控制 TCP/IP 尝试验证空闲连接是否完好的频率。如果这段时间内没有活动，则会发送保持活动信号。如果网络工作正常，而且接收方是活动的，它就会响应。如果需要对丢失接收方敏感，换句话说，需要更快地发现丢失了接收方，请考虑减小这个值。如果长期不活动的空闲连接出现次数较多，而丢失接收方的情况出现较少，您可能会要提高该值以减少开销。缺省情况下，如果空闲连接 7 200 000 毫秒内没有活动，Windows 就发送保持活动的消息。通常，1 800 000 毫秒是首选值，从而一半的已关闭连接会在 30 分钟内被检测到。KeepAliveInterval 值定义了如果未从接收方收到保持活动消息的响应，TCP/IP 重复发送保持活动信号的频率。当连续发送保持活动信号但未收到响应的次数超出 TcpMaxDataRetransmissions 的值时，就会放弃该连接。如果期望较长的响应时间，您可能需要提高该值以减少开销。如果需要减少花在验证接收方是否已丢失上的时间，请考虑减小该值 TcpMaxDataRetransmissions 值。缺省情况下，在未收到响应而重新发送保持活动的消息之前，Windows 会等待 1000 毫秒(1 秒)。KeepAliveTime 根据你的需要设置就行，比如 10 分钟，注意要转换成 MS。XXX 代表这个间隔值的大小。

(2) Date 头域

Date 头域表示消息发送的时间，时间的描述格式由 rfc822 定义。例如，Date：Mon，31Dec200104：25：57GMT。Date 描述的时间表示世界标准时，换算成本地时间，需要知道用户所在的时区。

(3) Pragma 头域

Pragma 头域用来包含实现特定的指令，最常用的是 Pragma：no-cache。在 HTTP/1.1 协议中，它的含义和 Cache-Control：no-cache 相同。

2. 请求消息

请求消息的第一行为下面的格式。

MethodSPRequest-URISPHTTP-VersionCRLFMethod 表示对于 Request-URI 完成的方法，这个字段是大小写敏感的，包括 OPTIONS、GET、HEAD、POST、PUT、DELETE、TRACE。方法 GET 和 HEAD 应该被所有的通用 Web 服务器支持，其他所有方法的实现是可选的。GET 方法取回由 Request-URI 标识的信息。HEAD 方法也是取回由 Request-URI 标识的信息，只是可以在响应时，不返回消息体。POST 方法可以请求服务器接收包含在请求中的实体信息，可以用于提交表单，向新闻组、BBS、邮件群组和数据库发送消息。

SP 表示空格。Request-URI 遵循 URI 格式，在此字段为星号(*)时，说明请求并不用于某个特定的资源地址，而是用于服务器本身。HTTP-Version 表示支持的 HTTP 版本，例如为 HTTP/1. 1。CRLF 表示换行回车符。请求头域允许客户端向服务器传递关于请求或者关于客户机的附加信息。请求头域可能包含下列字段 Accept、Accept-Charset、Accept-Encoding、Accept-Language、Authorization、From、Host、If-Modified-Since、If-Match、If-None-Match、If-Range、If-Range、If-Unmodified-Since、Max-Forwards、Proxy-Authorization、Range、Referer、User-Agent。对请求头域的扩展要求通信双方都支持，如果存在不支持的请求头域，一般将会作为实体头域处理。

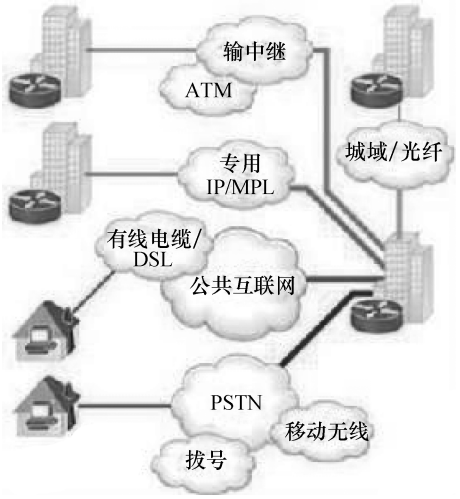


图 1. 14　请求过程

典型的请求消息：

Host: download. *******. de

Accept: */*

Pragma: no-cache

Cache-Control: no-cache

User-Agent: Mozilla/4. 04[en](Win95; I; Nav)

Range: bytes = 554554 -

上例第一行表示 HTTP 客户端(可能是浏览器、下载程序)通过 GET 方法获得指定 URL 下的文件。棕色的部分表示请求头域的信息，绿色的部分表示通用头部分。

(1) Host 头域

Host 头域指定请求资源的 Intenet 主机和端口号，必须表示请求 url 的原始服务器或网关的位置。HTTP/1. 1 请求必须包含主机头域，否则系统会以 400 状态码返回。

(2) Referer 头域

Referer 头域允许客户端指定请求 uri 的源资源地址，这可以允许服务器生成回退链表，可用来登陆、优化 cache 等。他也允许废除的或错误的连接由于维护的目的被追踪。如果请求的 uri 没有自己的 uri 地址，Referer 不能被发送。如果指定的是部分 uri 地址，则此地

址应该是一个相对地址。

（3）Range 头域

Range 头域可以请求实体的一个或者多个子范围。例如：

表示头 500 个字节：bytes = 0 - 499

表示第二个 500 字节：bytes = 500 - 999

表示最后 500 个字节：bytes = - 500

表示 500 字节以后的范围：bytes = 500 -

第一个和最后一个字节：bytes = 0 - 0，-1

同时指定几个范围：bytes = 500 - 600，601 - 999

但是服务器可以忽略此请求头，如果无条件 GET 包含 Range 请求头，响应会以状态码 206（PartialContent）返回而不是以 200。

（4）User-Agent 头域

User-Agent 头域的内容包含发出请求的用户信息。

3. 响应消息

响应消息的第一行为下面的格式。

HTTP-VersionSPStatus-CodeSPReason-PhraseCRLF

HTTP-Version 表示支持的 HTTP 版本，例如为 HTTP/1.1。Status-Code 是一个三个数字的结果代码。Reason-Phrase 给 Status-Code 提供一个简单的文本描述。Status-Code 主要用于机器自动识别，Reason-Phrase 主要用于帮助用户理解。Status-Code 的第一个数字定义响应的类别，后两个数字没有分类的作用。第一个数字可能取 5 个不同的值。

1xx：信息响应类，表示接收到请求并且继续处理。

2xx：处理成功响应类，表示动作被成功接收、理解和接受。

3xx：重定向响应类，为了完成指定的动作，必须接受进一步处理。

4xx：客户端错误，客户请求包含语法错误或者是不能正确执行。

5xx：服务端错误，服务器不能正确执行一个正确的请求。

响应头域允许服务器传递不能放在状态行的附加信息，这些域主要描述服务器的信息和 Request-URI 进一步的信息。响应头域包含 Age、Location、Proxy-Authenticate、Public、Retry-After、Server、Vary、Warning、WWW-Authenticate。对响应头域的扩展要求通信双方都支持，如果存在不支持的响应头域，一般将会作为实体头域处理。

典型的响应消息：

HTTP/1.0200OK

Date：Mon，31Dec200104：25：57GMT

Server：Apache/1.3.14（Unix）

Content-type：text/html

Last-modified：Tue，17Apr200106：46：28GMT

Etag："a030f020ac7c01：1e9f"

Content-length：39725426

Content-range：bytes55 ****** /40279980

上例第一行表示 HTTP 服务端响应一个 GET 方法。

（1）Location 响应头

Location 响应头用于重定向接收者到一个新 URI 地址。

（2）Server 响应头

Server 响应头包含处理请求的原始服务器的软件信息。此域能包含多个产品标识和注释，产品标识一般按照重要性排序。

4. 实体信息

请求消息和响应消息都可以包含实体信息，实体信息一般由实体头域和实体组成。实体头域包含关于实体的原信息，实体头包括 Allow、Content-Base、Content-Encoding、Content-Language、Content-Length、Content-Location、Content-MD5、Content-Range、Content-Type、Etag、Expires、Last-Modified、extension-header。extension-header 允许客户端定义新的实体头，但是这些域可能无法被接受方识别。实体可以是一个经过编码的字节流，它的编码方式由 Content-Encoding 或 Content-Type 定义，它的长度由 Content-Length 或 Content-Range 定义。

（1）Content-Type 实体头

Content-Type 实体头用于向接收方指示实体的介质类型，指定 HEAD 方法送到接收方的实体介质类型，或 GET 方法发送的请求介质类型。

（2）Content-Range 实体头

Content-Range 实体头用于指定整个实体中的一部分的插入位置，他也指示了整个实体的长度。在服务器向客户返回一个部分响应，它必须描述响应覆盖的范围和整个实体长度。一般格式：

Content-Range：bytes-unitSPfirst-byte-pos-last-byte-pos/entity-legth

例如，传送头 500 个字节次字段的形式：Content-Range：bytes0-499/1234 如果一个 http消息包含此节（例如，对范围请求的响应或对一系列范围的重叠请求），Content-Range 表示传送的范围，Content-Length 表示实际传送的字节数。

（3）Last-modified 实体头

Last-modified 实体头指定服务器上保存内容的最后修订时间。

例如，传送头 500 个字节次字段的形式：Content-Range：bytes0-499/1234 如果一个 http消息包含此节（例如，对范围请求的响应或对一系列范围的重叠请求），Content-Range 表示传送的范围，Content-Length 表示实际传送的字节数。

1.3.4 运作方式

在 WWW 中，“客户”与“服务器”是一个相对的概念，只存在于一个特定的连接期间，即在某个连接中的客户在另一个连接中可能作为服务器。基于 HTTP 协议的客户/服务器模式的信息交换过程，它分四个过程：建立连接、发送请求信息、发送响应信息、关闭连接。

HTTP 协议是基于请求/响应范式的。一个客户机与服务器建立连接后，发送一个请求给服务器，请求方式的格式为统一资源标识符、协议版本号，后边是 MIME 信息包括请求

修饰符、客户机信息和可能的内容。服务器接到请求后，给予相应的响应信息，其格式为一个状态行包括信息的协议版本号、一个成功或错误的代码，后边是 MIME 信息包括服务器信息、实体信息和可能的内容。如图 1.15 所示。

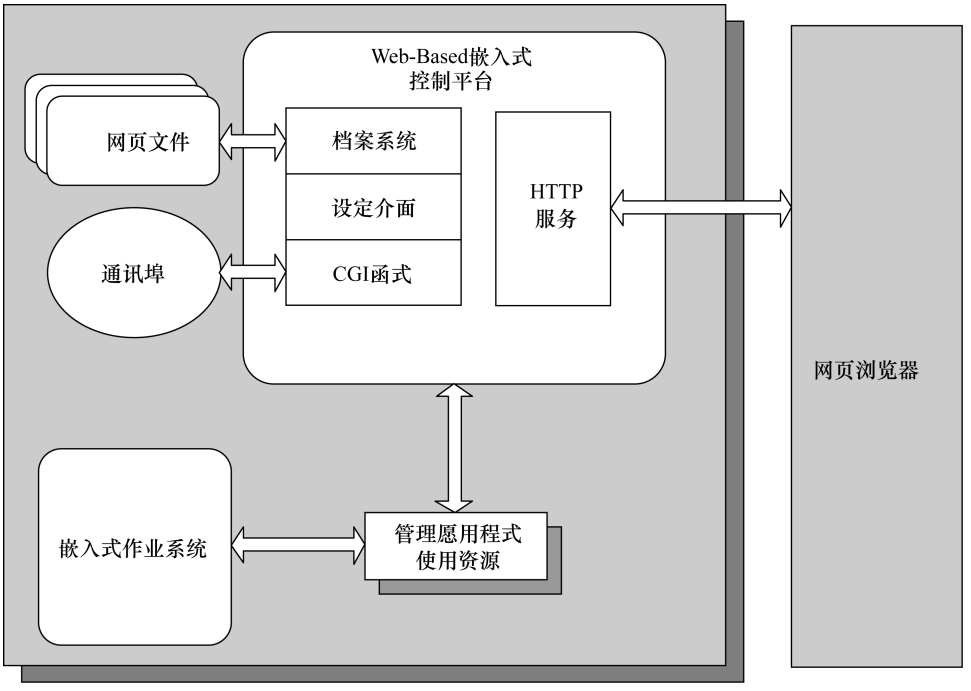


图 1.15　运作方式

其实简单说就是任何服务器除了包括 HTML 文件以外，还有一个 HTTP 驻留程序，用于响应用户请求。你的浏览器是 HTTP 客户，向服务器发送请求，当浏览器中输入了一个开始文件或点击了一个超级链接时，浏览器就向服务器发送了 HTTP 请求，此请求被送往由 IP 地址指定的 URL。驻留程序接收到请求，在进行必要的操作后回送所要求的文件。在这一过程中，在网络上发送和接收的数据已经被分成一个或多个数据包(packet)，每个数据包包括：要传送的数据；控制信息，即告诉网络怎样处理数据包。TCP/IP 决定了每个数据包的格式。如果事先不告诉你，你可能不会知道信息被分成用于传输和再重新组合起来的许多小块。

许多 HTTP 通讯是由一个用户代理初始化的并且包括一个申请在源服务器上资源的请求。最简单的情况可能是在用户代理(UA)和源服务器(O)之间通过一个单独的连接来完成。

当一个或多个中介出现在请求/响应链中时，情况就变得复杂一些。中介有三种：代理(Proxy)、网关(Gateway)和通道(Tunnel)。一个代理根据 URI 的绝对格式来接受请求，重写全部或部分消息，通过 URI 的标识把已格式化过的请求发送到服务器。网关是一个接收代理，作为一些其他服务器的上层，并且如果必须的话，可以把请求翻译给下层的服务器协议。一个通道作为不改变消息的两个连接之间的中继点。当通信需要通过一个中介(例如：防火墙等)或者是中介不能识别消息的内容时，通道经常被使用。

1.3.5 报文格式

HTTP 报文由从客户机到服务器的请求和从服务器到客户机的响应构成。请求报文格式如下。

请求行 – 通用信息头 – 请求头 – 实体头 – 报文主体

请求行以方法字段开始，后面分别是 URL 字段和 HTTP 协议版本字段，并以 CRLF 结尾。SP 是分隔符。除了在最后的 CRLF 序列中 CF 和 LF 是必需的之外，其他都可以不要。有关通用信息头，请求头和实体头方面的具体内容可以参照相关文件。

应答报文格式如下。

状态行 – 通用信息头 – 响应头 – 实体头 – 报文主体

状态码元由 3 位数字组成，表示请求是否被理解或被满足。原因分析是对原文的状态码做简短的描述，状态码用来支持自动操作，而原因分析用来供用户使用。客户机无须用来检查或显示语法。有关通用信息头、响应头和实体头方面的具体内容可以参照相关文件。

1.3.6 工作原理

一次 HTTP 操作称为一个事务，其工作过程可分为四步。首先客户机与服务器需要建立连接。只要单击某个超级链接，HTTP 的工作就开始了。建立连接后，客户机发送一个请求给服务器，请求方式的格式为：统一资源标识符(URL)、协议版本号，后边是 MIME 信息包括请求修饰符、客户机信息和可能的内容。服务器接到请求后，给予相应的响应信息，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码，后边是 MIME 信息包括服务器信息、实体信息和可能的内容。客户端接收服务器所返回的信息通过浏览器显示在用户的显示屏上，然后客户机与服务器断开连接。

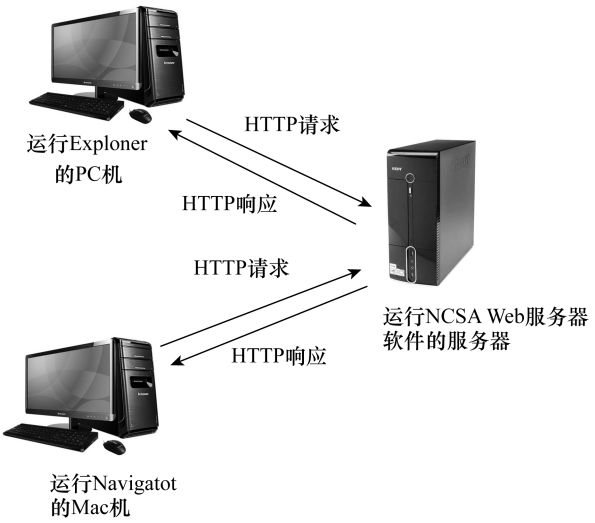


图 1.16 工作原理

如果在以上过程中的某一步出现错误，那么产生错误的信息将返回到客户端，由显示屏输出。对于用户来说，这些过程是由 HTTP 自己完成的，用户只要用鼠标点击，等待信

息显示就可以了。

许多 HTTP 通信是由一个用户代理初始化的并且包括一个申请在源服务器上资源的请求。最简单的情况可能是在用户代理和服务器之间通过一个单独的连接来完成。在Internet上，HTTP 通信通常发生在 TCP/IP 连接之上。缺省端口是 TCP 80，但其他的端口也是可用的。但这并不预示着 HTTP 协议在 Internet 或其他网络的其他协议之上才能完成。HTTP 只预示着一个可靠的传输。

这个过程就好像我们打电话订货一样，我们可以打电话给商家，告诉他我们需要什么规格的商品，然后商家再告诉我们什么商品有货，什么商品缺货。这些，我们是通过电话线用电话联系(HTTP 是通过 TCP/IP)，当然我们也可以通过传真，只要商家那边也有传真。

1.3.7　状态消息

表 1.1　信息状态消息

消息	描述
100 Continue	服务器仅接收到部分请求，但是一旦服务器并没有拒绝该请求，客户端应该继续发送其余的请求
101 Switching Protocols	服务器转换协议：服务器将遵从客户的请求转换到另外一种协议

表 1.2　成功状态消息

消息	描述
200 OK	请求成功(其后是对 GET 和 POST 请求的应答文档)
201 Created	请求被创建完成，同时新的资源被创建
202 Accepted	供处理的请求已被接受，但是处理未完成
203 Non-authoritative Information	文档已经正常地返回，但一些应答头可能不正确，因为使用的是文档的拷贝
204 No Content	没有新文档，浏览器应该继续显示原来的文档，如果用户定期地刷新页面，而 Servlet 可以确定用户文档足够新，这个状态代码是很有用的
205 Reset Content	没有新文档，但浏览器应该重置它所显示的内容，用来强制浏览器清除表单输入内容
206 Partial Content	客户发送了一个带有 Range 头的 GET 请求，服务器完成了它

表 1.3　重定向状态消息

消息	描述
300 Multiple Choices	多重选择，链接列表，用户可以选择某链接到达目的地，最多允许五个地址
301 Moved Permanently	所请求的页面已经转移至新的 url
302 Found	所请求的页面已经临时转移至新的 url
303 See Other	所请求的页面可在别的 url 下被找到
304 Not Modified	未按预期修改文档，客户端有缓冲的文档并发出了一个条件性的请求(一般是提供 If-Modified-Since 头表示客户只想比指定日期更新的文档)，服务器告诉客户，原来缓冲的文档还可以继续使用
305 Use Proxy	客户请求的文档应该通过 Location 头所指明的代理服务器提取
306 Unused	此代码被用于前一版本，目前已不再使用，但是代码依然被保留
307 Temporary Redirect	被请求的页面已经临时移至新的 url

表 1.4　客户端错误状态消息

消息	描述
400 Bad Request	服务器未能理解请求
401 Unauthorized	被请求的页面需要用户名和密码
401.1	登录失败
401.2	服务器配置导致登录失败
401.3	由于 ACL 对资源的限制而未获得授权
401.4	筛选器授权失败
401.5	ISAPI/CGI 应用程序授权失败
401.7	访问被 Web 服务器上的 URL 授权策略拒绝，这个错误代码为 IIS 6.0 所专用
402 Payment Required	此代码尚无法使用
403 Forbidden	对被请求页面的访问被禁止
403.1	执行访问被禁止
403.2	读访问被禁止
403.3	写访问被禁止
403.4	要求 SSL
403.5	要求 SSL 128
403.6	IP 地址被拒绝

续表

消息	描述
403. 7	要求客户端证书
403. 8	站点访问被拒绝
403. 9	用户数过多
403. 10	配置无效
403. 11	密码更改
403. 12	拒绝访问映射表
403. 13	客户端证书被吊销
403. 14	拒绝目录列表
403. 15	超出客户端访问许可
403. 16	客户端证书不受信任或无效
403. 17	客户端证书已过期或尚未生效
403. 18	在当前的应用程序池中不能执行所请求的 URL，这个错误代码为 IIS 6. 0 所专用
403. 19	不能为这个应用程序池中的客户端执行 CGI，这个错误代码为 IIS 6. 0 所专用
403. 20	Passport 登录失败，这个错误代码为 IIS 6. 0 所专用
404 Not Found	服务器无法找到被请求的页面
404. 0	(无)没有找到文件或目录
404. 1	无法在所请求的端口上访问 Web 站点
404. 2	Web 服务扩展锁定策略阻止本请求
404. 3	MIME 映射策略阻止本请求
405 Method Not Allowed	请求中指定的方法不被允许
406 Not Acceptable	服务器生成的响应无法被客户端所接受
407 Proxy Authentication Required	用户必须首先使用代理服务器进行验证，这样请求才会被处理
408 Request Timeout	请求超出了服务器的等待时间
409 Conflict	由于冲突，请求无法被完成
410 Gone	被请求的页面不可用
411 Length Required	“Content-Length”未被定义，如果无此内容，服务器不会接受请求
412 Precondition Failed	请求中的前提条件被服务器评估为失败
413 Request Entity Too Large	由于所请求的实体的太大，服务器不会接受请求
414 Request-url Too Long	由于 url 太长，服务器不会接受请求，当 post 请求被转换为带有很长的查询信息的 get 请求时，就会发生这种情况

续表

消息	描述
415 Unsupported Media Type	由于媒介类型不被支持，服务器不会接受请求
416 Requested Range Not Satisfiable	服务器不能满足客户在请求中指定的 Range 头
417 Expectation Failed	执行失败
423	锁定的错误

表 1.5　服务器错误状态消息

消息	描述
500 Internal Server Error	请求未完成，服务器遇到不可预知的情况
500. 12	应用程序正忙于在 Web 服务器上重新启动
500. 13	Web 服务器太忙
500. 15	不允许直接请求 Global. asa
500. 16	UNC 授权凭据不正确，这个错误代码为 IIS 6. 0 所专用
500. 18	URL 授权存储不能打开，这个错误代码为 IIS 6. 0 所专用
500. 100	内部 ASP 错误。
501 Not Implemented	请求未完成，服务器不支持所请求的功能
502 Bad Gateway	请求未完成，服务器从上游服务器收到一个无效的响应
502. 1	CGI 应用程序超时
502. 2	CGI 应用程序出错
503 Service Unavailable	请求未完成，服务器临时过载或当机
504 Gateway Timeout	网关超时
505 HTTP Version Not Supported	服务器不支持请求中指明的 HTTP 协议版本

1. 3. 8　版本历史

超文本传输协议已经演化出了很多版本，它们中的大部分都是向下兼容的。在 RFC 2145 中描述了 HTTP 版本号的用法。客户端在请求的开始告诉服务器它采用的协议版本号，而后者则在响应中采用相同或者更早的协议版本。

HTTP/0. 9 已过时，只接受 GET 一种请求方法，没有在通信中指定版本号，且不支持请求头。由于该版本不支持 POST 方法，所以客户端无法向服务器传递太多信息。

HTTP/1. 0 这是第一个在通信中指定版本号的 HTTP 协议版本，至今仍被广泛采用，特别是在代理服务器中。

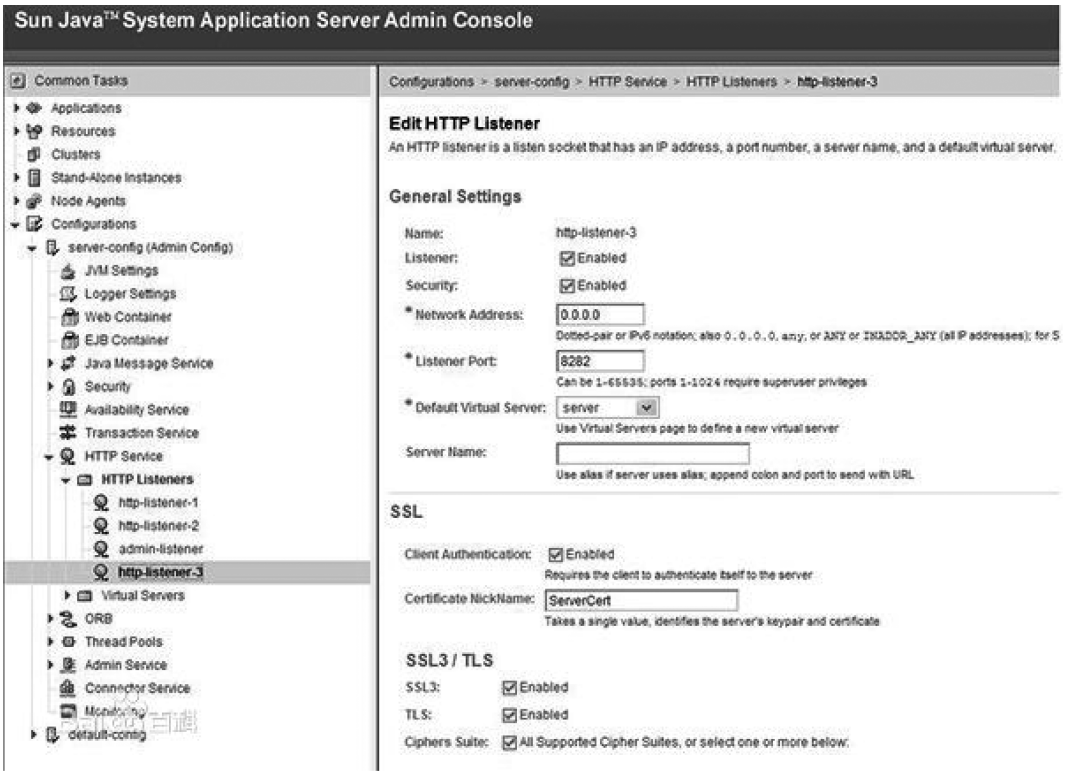


图 1.17　版本历史查看

HTTP/1.1，当前版本，持久连接被默认采用，并能很好地配合代理服务器工作。还支持以管道方式同时发送多个请求，以便降低线路负载，提高传输速度。

HTTP/1.1 相较于 HTTP/1.0 协议的区别主要体现在：

① 缓存处理；

② 带宽优化及网络连接的使用；

③ 错误通知的管理；

④ 消息在网络中的发送；

⑤ 互联网地址的维护；

⑥ 安全性及完整性。

1.4　搭建 Java Web 开发环境

1.4.1　下载 JDK

要运行 Java 程序，首先要安装 Java 的运行环境，即 Jre(java runtime environment)。所以需要安装 Java 的开发环境即下载并安装 JDK。JDK 可以到 sun 公司的官网上去下载，其网址为 http://www.oracle.com/technetwork/java/javase/downloads/index.html。界面如图 1.18所示。

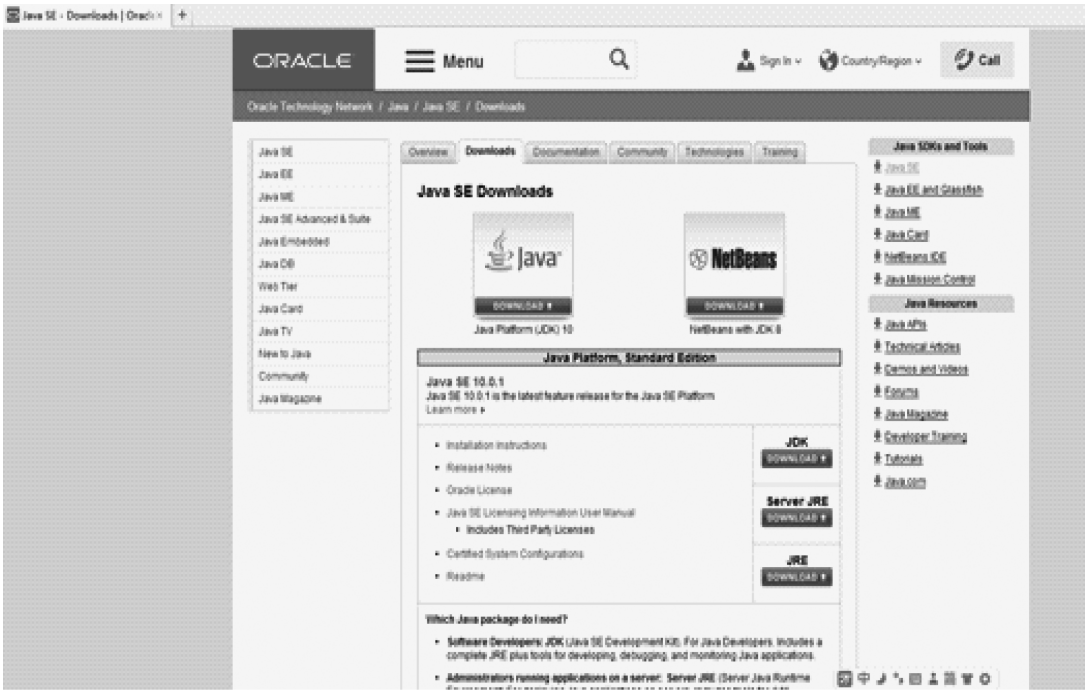


图 1.18　JRE 下载界面

直接下载即可。下载安装之后还要进行环境变量的设置。

1.4.2　Win7 下 Java 环境变量设置并测试

1. 右击我的电脑打开属性对话框，如图 1.19 所示。

图 1.19　利用属性打开环境变量设置

2. 单击高级系统设置，单击环境变量，如图 1.20 所示。

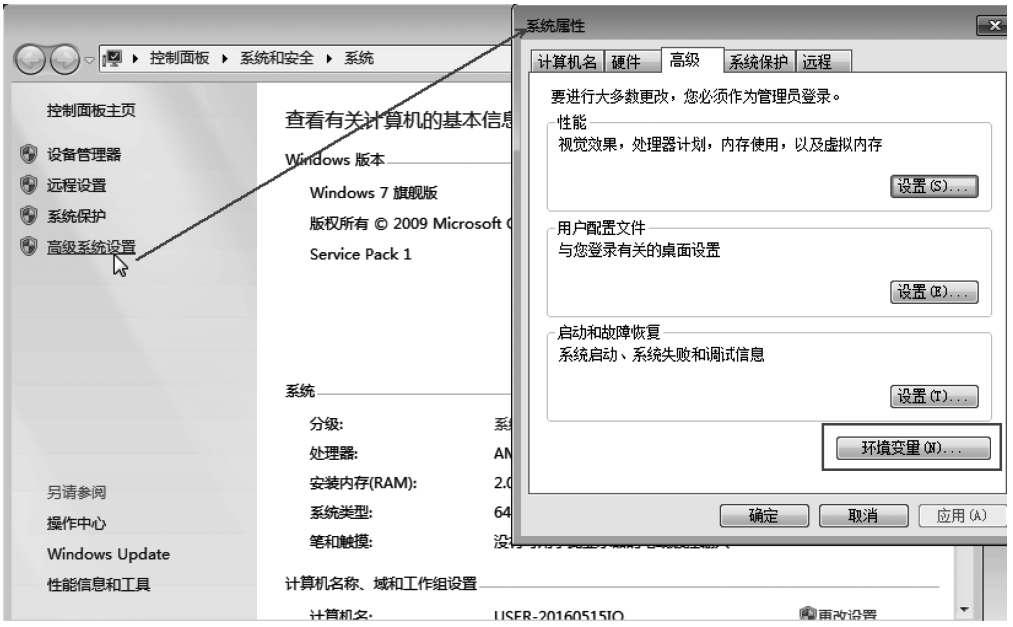


图 1.20　打开环境变量

3. 在系统变量中单击新建，其界面如图 1.21 所示。

图 1.21　新建变量

4. 变量名为 JAVA_HOME，变量值为 jdk 安装路径，如图 1.22 所示。

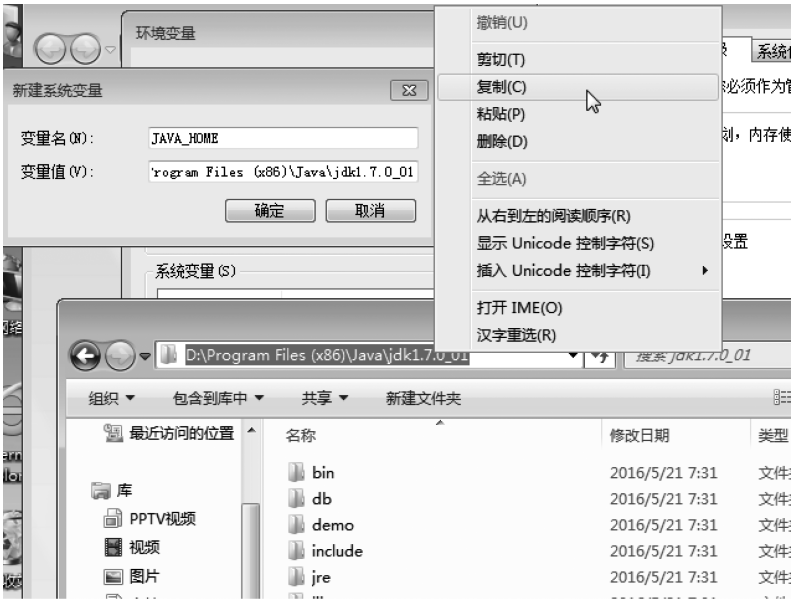


图 1.22　JAVA_ HOME 变量的建立过程

5. 单击确定后，再单击新建 变量名为 PATH，如图 1.23 所示。

变量值为　　%JAVA_HOME%\bin;%JAVA_HOME%\jre\bin;

图 1.23　PATH 变量的修改

6. 单击确定后，再单击新建 变量名为 CLASSPATH，如图 1.24 所示。

变量值为　　.；JAVA_HOME%\lib\dt.jar;%JAVA_HOME%lib\tools.jar

图 1.24　CLASSPATH 变量的建立

7. 设置完成后单击确定，界面如图 1.25 所示。

图 1.25　CLASSPATH 变量成功建立界面

8. 单击开始运行命令提示符，界面如图 1.26 所示。

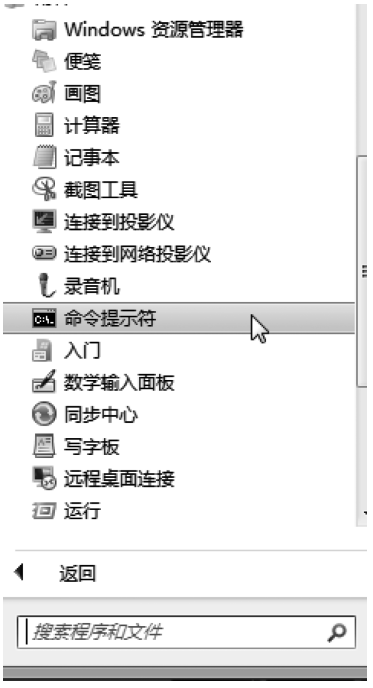


图 1.26　利用开始菜单打开 CMD

9. 键入 Java，界面如图 1.27 所示。

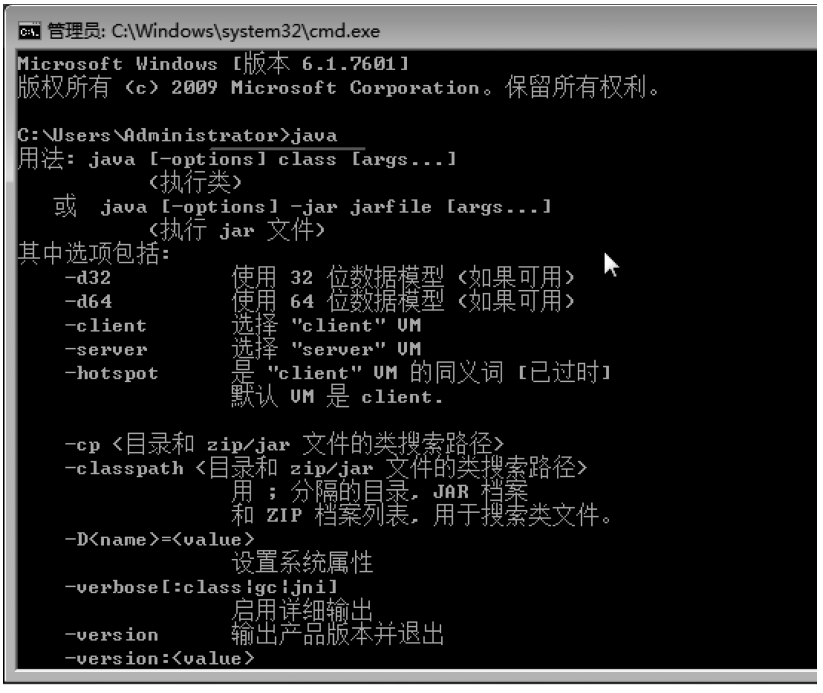


图 1.27　键入 JAVA 后的界面

10. 键入 Javac，界面如图 1.28 所示。

图 1.28　键入 JAVAC 的成功界面

11. 新建记事本文件，命名为 test.java，界面如图 1.29 所示。

在记事本中输入以下文字。

```
import java.io.*;
public class test{
    public static void main(String args[]){
      System.out.println("欢迎来到 JAVA");}
}
```

图 1.29　新建 java 文件

12. 在 CMD 中编译，输入 javac test. java，生成 test. class，如图 1. 30 所示。

图 1. 30　对新建的 Java 文件进行编译并生成可执行文件

13. 运行 test 文件，输入 java test 测试成功，界面如图 1. 31 所示。

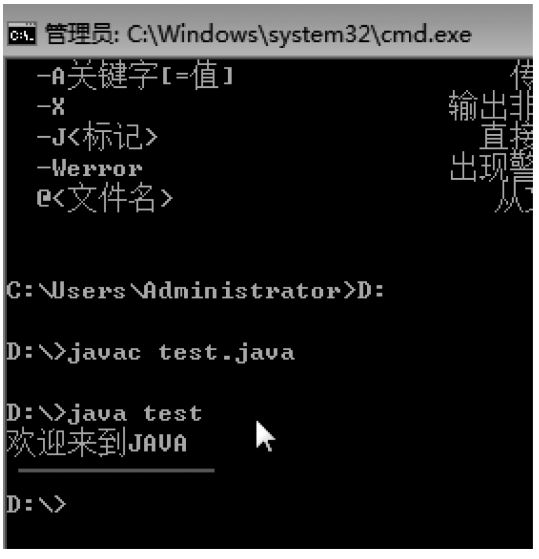


图 1. 31　执行文件验证环境变量的设置成功

1. 4. 3　Win10 下 JAVA 环境变量设置

操作系统的不同，环境变量的设置过程也稍有不同。

1. 右击我的电脑打开属性对话框，如图 1. 32 所示。

图 1.32　利用属性打开环境变量的设置

2. 单击高级系统设置，单击环境变量，界面如图 1.33 所示。

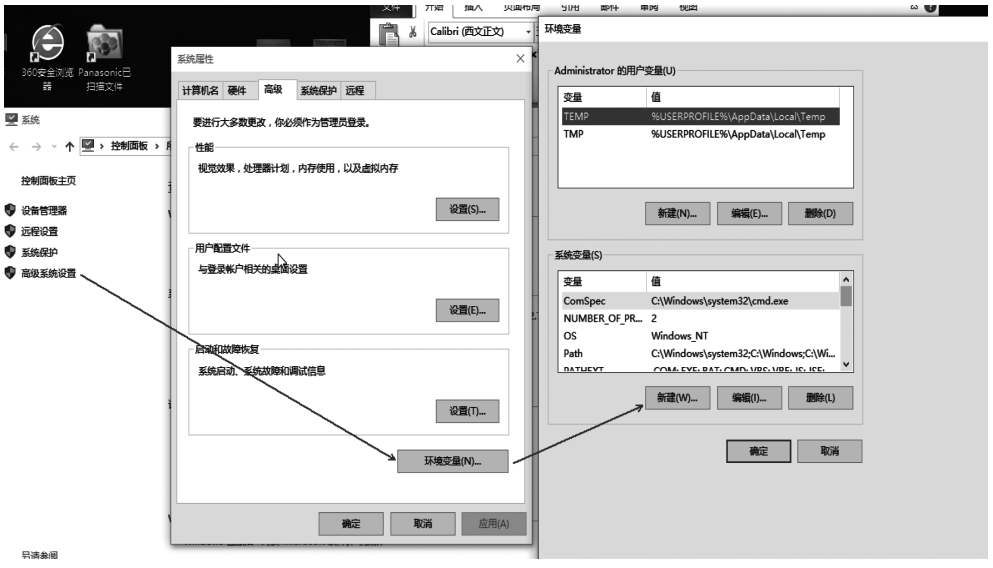


图 1.33　环境变量设置界面

3. 在系统变量中单击新建，如图 1.34 所示。

图 1.34　新建环境变量

4. 变量名为 JAVA_HOME，变量值为 jdk 安装路径，界面如图 1.35 和 1.36 所示。

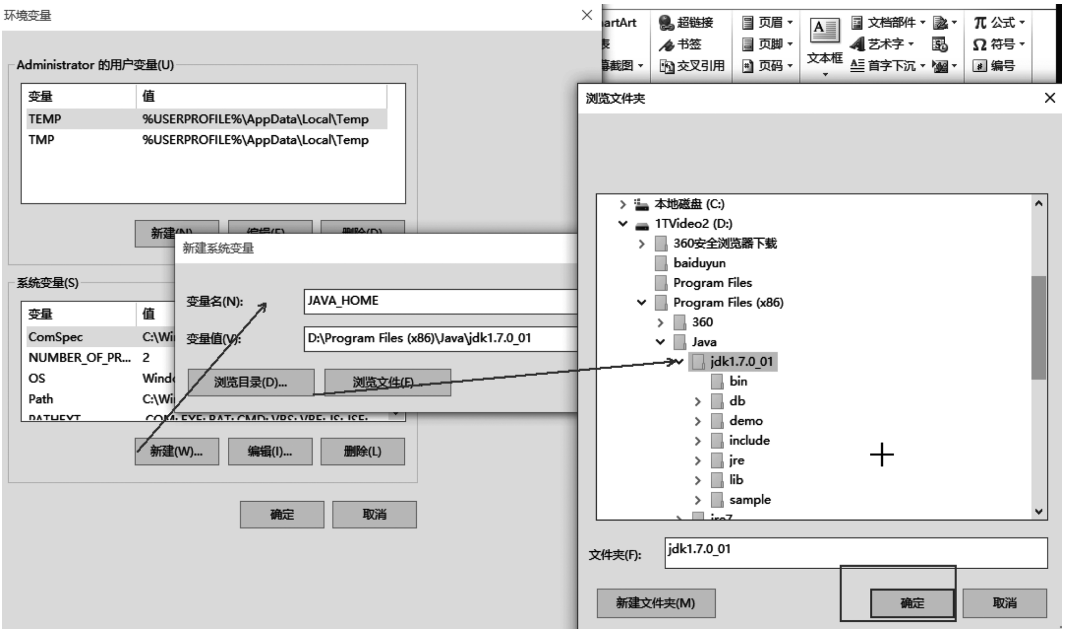


图 1.35　JAVA_ HOME 环境变量的设置

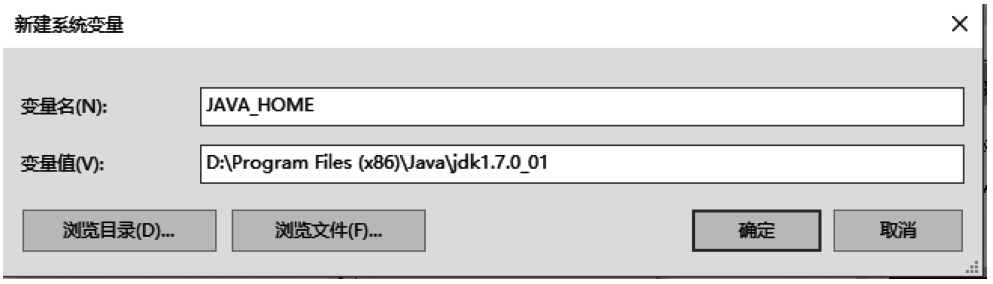


图 1.36　AVA_ HOME 环境变量的变量值设置

5. 单击确定后，再对变量名为 Path　的进行编辑，如没有 path，可新建
编辑时新建两个键值,分别为%JAVA_HOME% \bin
%JAVA_HOME% \jre\bin(如图 1.37 所示)。

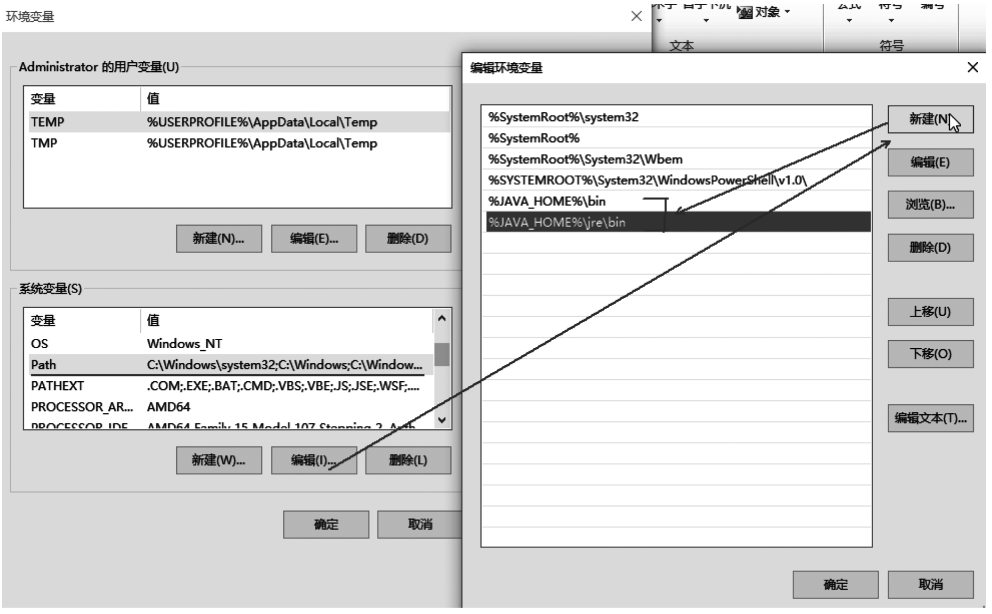


图 1.37　path 变量的修改

6. 单击确定后，再单击新建变量名为 CLASSPATH。

变量值为 . ; JAVA_ HOME% \ lib \ dt. jar;% JAVA_ HOME% lib \ tools. jar，如图 1.38 所示。

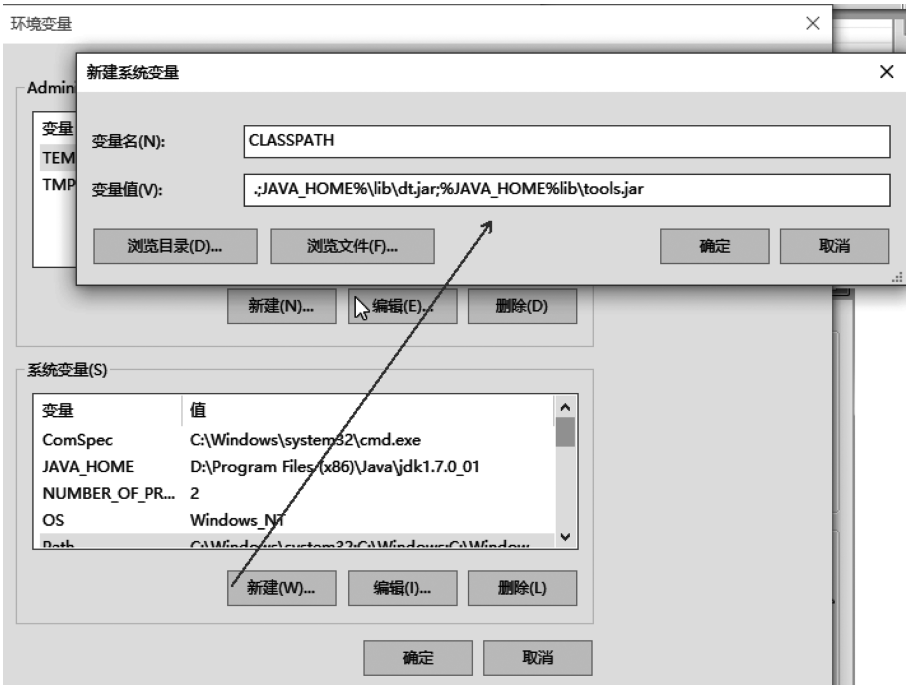


图 1.38　CLASSPATH 变量的设置

7. 设置完成后单击确定，界面如图 1.39 所示。

图 1.39　CLASSPATH 变量设置成功

8. 鼠标右键单击开始运行命令提示符(管理员)(A)，界面如图 1.40 所示。

图 1.40　利用开始菜单打开 CMD

9. 键入 Java，界面如图 1.41 所示。

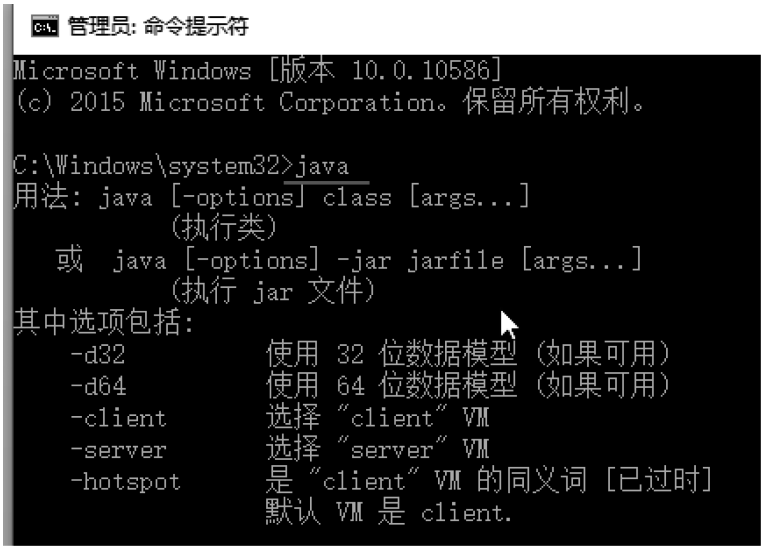


图 1.41　键入 Java 后应显示的界面

10. 键入 Javac，如图 1.42 所示。

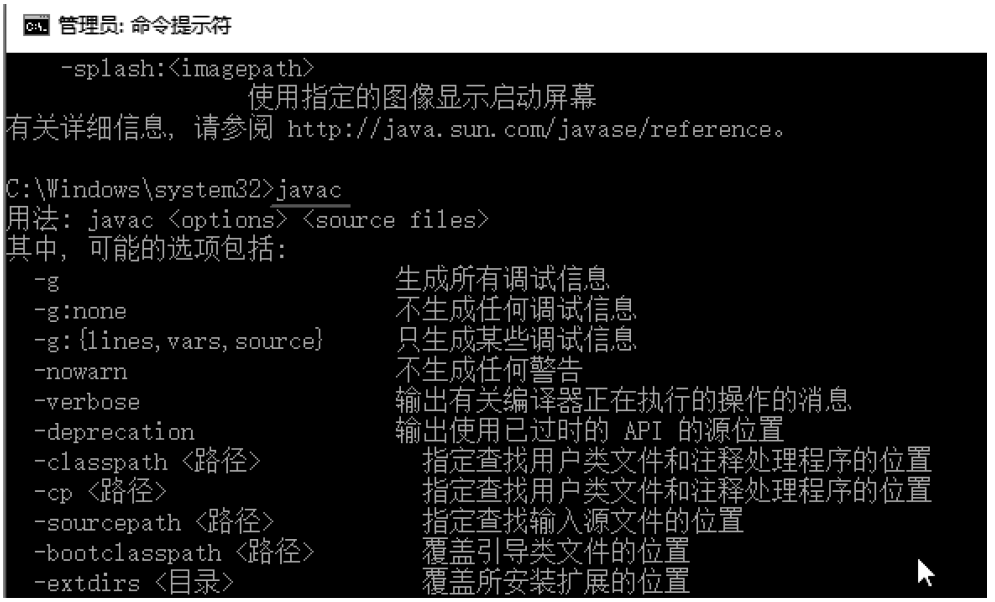


图 1.42　键入 Javac 后应显示的内容

11. 新建记事本文件，命名为 test.java。

在记事本中输入以下文字：

```
import java.io.*;
public class test{
    public static void main(String args[]){
        System.out.println("欢迎来到 JAVA");}
}
```

如图 1.43 所示。

图 1.43　新建 Java 文件

12. 在 CMD 中编译，输入 javac test. java，生成 test. class，界面如图 1. 44 所示。

图 1. 44　对新建的 Java 件进行编译

13. 运行 test 文件，输入 java test 测试成功，界面如图 1. 45 所示。

图 1. 45　执行文件并测试环境变量是否设置成功

安装完成 Java 的运行环境后，Java 的编译可以使用记事本来进行编写，也可下载 netbeans、eclipse、dreamweaver 等专门的 Java 编译工具进行编写。

1. 5　JSP 技术简介

JSP(JavaServerPages)是由 Sun Microsystems 公司倡导、许多公司参与一起建立的一种动态网页技术标准。JSP 技术有点类似 ASP 技术，它是在传统的网页 HTML 文件(*. htm，*. html)中插入 Java 程序段(Scriptlet)和 JSP 标记(tag)，从而形成 JSP 文件(*. jsp)。用 JSP 开发的 Web 应用是跨平台的，既能在 Linux 下运行，也能在其他操作系统上运行。JSP 技术使用 Java 编程语言编写类 XML 的 tags 和 scriptlets，来封装产生动态网页的处理逻辑。网页还能通过 tags 和 scriptlets 访问存在于服务端的资源的应用逻辑。JSP 将网页逻辑与网页设计和显示分离，支持可重用的基于组件的设计，使基于 Web 的应用程序的开发变得

迅速和容易。

Web 服务器在遇到访问 JSP 网页的请求时，首先执行其中的程序段，然后将执行结果连同 JSP 文件中的 HTML 代码一起返回给客户。插入的 Java 程序段可以操作数据库、重新定向网页等，以实现建立动态网页所需要的功能。

JSP 与 Java Servlet 一样，是在服务器端执行的，通常返回该客户端的就是一个 HTML 文本，因此客户端只要有浏览器就能浏览。JSP 的 1.0 规范的最后版本是 1999 年 9 月推出的，同年 12 月又推出了 1.1 规范。目前较新的是 JSP1.2 规范，JSP2.0 规范的征求意见稿也已出台。JSP 页面由 HTML 代码和嵌入其中的 Java 代码所组成。服务器在页面被客户端请求以后对这些 Java 代码进行处理，然后将生成的 HTML 页面返回给客户端的浏览器。JavaServlet 是 JSP 的技术基础，而且大型的 Web 应用程序的开发需要 JavaServlet 和 JSP 配合才能完成。JSP 具备了 Java 技术的简单易用，完全的面向对象，具有平台无关性且安全可靠，主要面向因特网的所有特点。自 JSP 推出后，众多大公司都支持 JSP 技术的服务器，如 IBM、Oracle、Bea 公司等，所以 JSP 迅速成为商业应用的服务器端语言。

1. JSP2.0 介绍

新的 JSP 规范版本包括新的用于提升程序员工作效率功能，主要有：AnExpressionLanguage(EL)允许开发者创建 Velocity-样式 templates（amongotherthings）. 更快更简单的创建新标签的方法 MVC 模式为了把表现层 presentation 从请求处理 requestprocessing 和数据存储 datastorage 中分离开来，SUN 公司推荐在 JSP 文件中使用一种模-视图-控件 Model-view-controller 模式。规范的 SERVLET 或者分离的 JSP 文件用于处理请求。当请求处理完后，控制权交给一个只作为创建输出作用的 JSP 页。有几种平台都基于服务于网络层的模-视图-控件模 JSP 技术的强势

（1）一次编写，到处运行。在这一点上 Java 比 PHP 更出色，除了系统之外，代码不用做任何更改。

（2）系统的多平台支持。基本上可以在所有平台上的任意环境中开发，在任意环境中进行系统部署，在任意环境中扩展。相比 ASP/PHP 的局限性是显而易见的。

（3）强大的可伸缩性。从只有一个小的 Jar 文件就可以运行 Servlet/JSP，到由多台服务器进行集群和负载均衡，到多台 Application 进行事务处理，消息处理，一台服务器到无数台服务器，Java 显示出了巨大的生命力。

（4）多样化和功能强大的开发工具支持。这一点与 ASP 很像，Java 已经有了许多非常优秀的开发工具，而且许多可以免费得到，并且其中许多已经可以顺利地运行于多种平台之下。

2. JSP 技术的弱势

（1）与 ASP 一样，Java 的一些优势正是它致命的问题所在。正是由于为了跨平台的功能，为了极度的伸缩能力，所以极大地增加了产品的复杂性。

（2）Java 的运行速度是用 class 常驻内存来完成的，所以它在一些情况下所使用的内存比起用户数量来说确实是“最低性能价格比”了。但同时它还需要硬盘空间来储存一系列的 . java 文件和 . class 文件以及对应的版本文件。

3. JSP 的技术方法

为了快速方便地进行动态网站的开发，JSP 在以下几个方面做了改进，使其成为快速建立跨平台的动态网站的首选方案。

（1）将内容的生成和显示进行分离用 JSP 技术，Web 页面开发人员可以使用 HTML 或者 XML 标识来设计和格式化最终页面，并使用 JSP 标识或者小脚本来生成页面上的动态内容(内容是根据请求变化的，例如请求账户信息或者特定的一瓶酒的价格等)。生成内容的逻辑被封装在标识和 JavaBeans 组件中，并且捆绑在脚本中，所有的脚本在服务器端运行。由于核心逻辑被封装在标识和 JavaBeans 中，所以 Web 管理人员和页面设计者，能够编辑和使用 JSP 页面，而不影响内容的生成。

在服务器端，JSP 引擎解释 JSP 标识和脚本，生成所请求的内容(例如，通过访问JavaBeans 组件，使用 JDBC 技术访问数据库或者包含文件)，并且将结果以 HTML(或者 XML)页面的形式发送回浏览器。这既有助于作者保护自己的代码，又能保证任何基于 HTML 的 Web 浏览器的完全可用性。

（2）可重用组件绝大多数 JSP 页面依赖于可重用的、跨平台的组件(JavaBeans 或者 EnterpriseJavaBeans 组件)来执行应用程序所要求的复杂的处理。开发人员能够共享和交换执行普通操作的组件，或者使得这些组件为更多的使用者和客户团体所使用。基于组件的方法加速了总体开发过程，并且使得各种组织在他们现有的技能和优化结果的开发努力中得到平衡。

(3)采用标识。Web 页面开发人员不会都是熟悉脚本语言的编程人员。JSP 技术封装了许多功能，这些功能是在易用的、与 JSP 相关的 XML 标识中进行动态内容生成所需要的。标准的 JSP 标识能够访问和实例化 JavaBeans 组件，设置或者检索组件属性，下载 Applet，以及执行用其他方法更难于编码和耗时的功能。

（4）适应平台几乎所有平台都支持 Java，JSP + JavaBeans 几乎可以在所有平台下通行无阻。从一个平台移植到另外一个平台，JSP 和 JavaBeans 甚至不用重新编译，因为 Java 字节码都是标准的，与平台无关的。

（5）数据库连接。Java 中连接数据库的技术是 JDBC，Java 程序通过 JDBC 驱动程序与数据库相连，执行查询、提取数据等操作。Sun 公司还开发了 JDBC-ODBCbridge，利用此技术 Java 程序可以访问带有 ODBC 驱动程序的数据库，目前大多数数据库系统都带有 ODBC驱动程序，所以 Java 程序能访问诸如 Oracle、Sybase、MSSQLServer 和 MSAccess 等数据库。此外，通过开发标识库，JSP 技术可以进一步扩展。第三方开发人员和其他人员可以为常用功能创建自己的标识库。这使得 Web 页面开发人员能够使用熟悉的工具和如同标识一样的执行特定功能的构件来进行工作。JSP 技术很容易整合到多种应用体系结构中，以利用现存的工具和技巧，并且能扩展到支持企业级的分布式应用中。作为采用 Java 技术家族的一部分，以及 Java2(企业版体系结构)的一个组成部分，JSP 技术能够支持高度复杂的基于 Web 的应用。由于 JSP 页面的内置脚本语言是基于 Java 的，而且所有的 JSP 页面都被编译成为 JavaServlets，所以 JSP 页面具有 Java 技术的所有好处，包括健壮的存储管理和安全性。作为 Java 平台的一部分，JSP 拥有 Java 编程语言“一次编写，各处运行”的

特点。

4. JSP 的应用模型

利用 JSP 技术，动态信息由 JSP 页面来表现，JSP 页面由安装在 Web 服务器或者使用 JSP 的应用服务器上的 JSP 引擎执行。JSP 引擎接受客户端对 JSP 页面的请求，并且生成 JSP 页面作为对客户端的响应。

JSP 页面通常被编译成为 JavaServlets，这是一个标准的 Java 扩展。页面开发人员能够访问全部的 Java 应用环境，以利用 Java 技术的扩展性和可移植性。当 JSP 页面第一次被调用时，如果它还不存在，就会被编译成为一个 JavaServlets 类，并且存储在服务器的内存中。这就使得在接下来的对该页面的调用中，服务器会有非常快的响应。（这避免了 CGI-BIN 为每个 HTTP 请求生成一个新的进程的问题）

JSP 页面可以包含在多种不同的应用体系结构或者模型中，可以用于由不同协议、组件和格式所组成的联合体中。基于 JSP 的动态信息发布技术是一个开放的、可扩展的建立动态 Web 页面的标准。不论采用什么创建工具，开发人员都可以使用 JSP 页面来创建可移植的 Web 应用，在不同的 Web 应用服务器上运行。

第 2 章 JSP 编译指令

2.1 指令元素

2.1.1 JSP 基本介绍

JSP 全名为 Java Server Pages，中文名叫 Java 服务器页面，其根本是一个简化的 Servlet 设计，它是由 Sun Microsystems 公司倡导、许多公司参与一起建立的一种动态网页技术标准。JSP 技术是在传统的网页 HTML 文件(*.htm，*.html)中插入 Java 程序段(Scriptlet)和 JSP 标记(tag)，从而形成 JSP 文件，后缀名为(*.jsp)。用 JSP 开发的 Web 应用是跨平台的，既能在 Linux 下运行，也能在其他操作系统上运行。

它实现了 Html 语法中的 Java 扩展(以<%，% >形式)。JSP 与 Servlet 一样，是在服务器端执行的。通常返回给客户端的就是一个 HTML 文本，因此客户端只要有浏览器就能浏览。

JSP 技术使用 Java 编程语言编写类 XML 的 tags 和 scriptlets，来封装产生动态网页的处理逻辑。网页还能通过 tags 和 scriptlets 访问存在于服务端的资源的应用逻辑。JSP 将网页逻辑与网页设计的显示分离，支持可重用的基于组件的设计，使基于 Web 的应用程序的开发变得迅速和容易。JSP(JavaServer Pages)是一种动态页面技术，它的主要目的是将表示逻辑从 Servlet 中分离出来。

2.1.2 JSP 指令

JSP 指令主要用来提供整个 JSP 页面的相关信息和设定 JSP 页面的相关属性，如设定网页的编码方式、脚本语言及导入需要用到的包等。其语法格式如下。

<%@指令名 属性名 = "属性值"% >

常用的有 3 条指令：page、include 和 taglib。

(1) page 指令。page 指令主要用来设定整个 JSP 文件的属性和相关功能，如：

<%@page contentType = "text/html, charset = gb2312"% >

一般用到的 page 指令还有导入需要的包，用法如下。

<%@page import = "java. util. List" % >

(2) include 指令。include 指令用来解决这个问题，其用来导入包含静态的文件，如

JSP 网页文件、HTML 网页文件，但不能包含用 <% = 和% > 表示的代表表达式的文件。其语法格式如下。

<%@include file = "被包含文件 url" % >

如有 head. jsp 文件，其内容如下。

<%@page language = "java" contentType = "text/html;charset = gb2312"% >
<%@page import = "java. sql. ResultSet"% >

现在在另一个文件中调用它。

<%@include file = "head. jsp"% >
<html >
<head > <title > 输出页面 </title > </head >
<body > 这句话是我想输出的 </body >
</html >

（3）taglib 指令。taglib 指令语法格式如下。

<%@taglib uri = "tagLibraryURI" prefix = "tagPrefix" % >

其中 uri = “tagLibraryURI”指明标签库文件的存放位置。而 prefix = “tagPrefix”则表示该标签使用时的前缀。例如，在 Struts 2 中用到标签。

<%@taglib uri = "/struts - tags" prefix = "s"% >

2.2 脚本元素

小脚本（scriptlets）是嵌入在 JSP 页面中的 Java 代码段。小脚本是以 <% 开头，以% > 结束的标签。例如 <% count ++ ; % >。

小脚本在每次访问页面时都被执行，因此 count 变量在每次请求时都增 1。由于小脚本可以包含任何 Java 代码，所以它通常用来在 JSP 页面嵌入计算逻辑。同时还可以使用小脚本打印 HTML 模板文本。

2.2.1 JSP 数据定义

在 JSP 中可以用 <%！和% > 定义一个或多个变量。在其中定义的变量为该页面级别的共享变量，可以被访问此页面的所有用户访问。其语法格式如下。

<%!变量声明 % >

如下面的代码片段。

```
<%!
    String name = "liu";
    int i = 0;
% >
```

此外，这种声明方式还可以定义一个方法或类，定义方法的格式如下。

<%!
返回值数据类型 函数名（数据类型，参数，…）{

```
            语句;
            return (返回值);
        }
    % >
```

定义一个类，如下面的代码片段：

```
    <%!
        puiblic class A{…}
    % >
```

2.2.2　JSP 程序块

【例 2.1】已知的圆的半径 10，计算圆的周长和面积，并在屏幕上输出。

在 MyEclipse 2014 中，选择主菜单【File】→【New】→【Web Project】，出现如图 2.1 所示的【New Web Project】窗口，填写“Project Name”栏(为项目起名)为“Practice”。在“Java EE version”下拉列表中选择“JavaEE 7-Web 3.1”，“Java version”下拉列表中选择“1.7”。(本节项目以 MyEclipse 2014 及 Tomcat8 为开发环境，后面的例子都按这个方法建立项目)

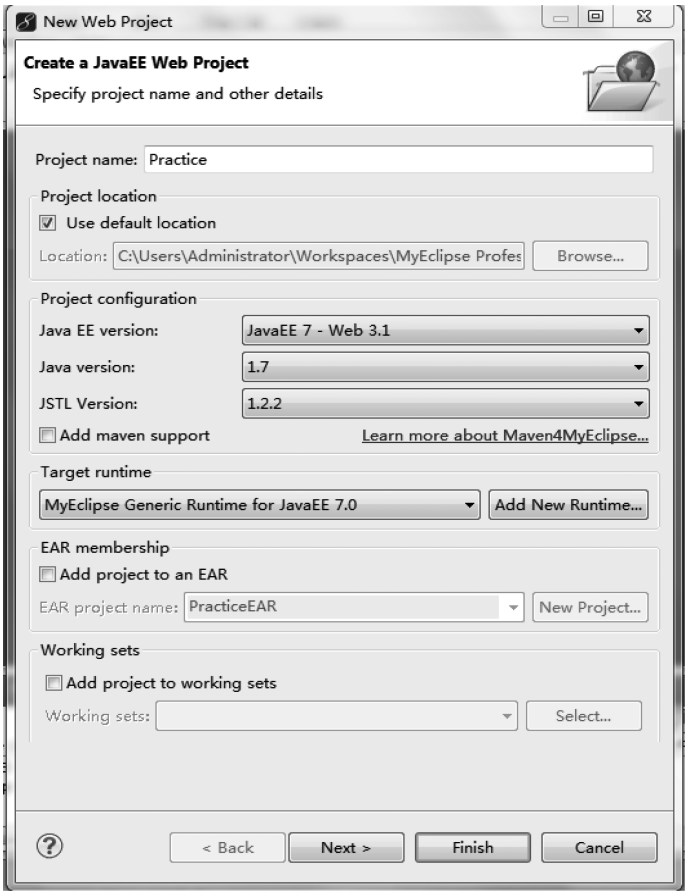


图 2.1　New Web Project 窗口

右击工程目录下 WebRoot，选择【new】菜单下的【file】，在 File name 中输入 circle. jsp。

输入代码：

```
<%@page contentType = "text/html; charset = utf - 8"% >
<html >
  <head >
      <title >
        计算圆的面积和周长
      </title >
  </head >
  <body >
  <%
      double pi = 3. 14, r = 10;
      double area, l;
      area = pi * r * r;
      l = 2 * pi * r;
      out. println("圆的面积是: " + area);
      out. println("圆的周长是: " + l);
    % >
    </body >
</html >
```

部署、启动 Tomcat。

在浏览器输入“http：//localhot：9080/Practice/circle. jsp”，在窗口中显示圆面积的值“314. 0”，圆的周长的值是“62. 8”。本章后续的小程序皆用这个 Practice 项目，在其中创建源文件部署执行即可。

注意：

Tomcat8 的下载网址为 http://tomcat. apache. org/，下载软件按默认路径安装，软件安装完成后，在浏览器中输入网址 http://localhost：8080/，跳入到 Apache Tomcat/8. 5. 11 版本，说明 Tomcat 安装成功。如果软件安装没问题，是端口被占用，则在软件安装路径 C：\ Program Files \ Apache Software Foundation \ Tomcat 8. 5 \ conf 目录下 server. xml 的记事本文件中。

2. 2. 3　JSP 表达式

从上面的例子中可以发现，要输出面积 s 的值，先计算 s 的值，然后输出结果。JSP 中提供了一种表达式，可以很方便地输出运算结果，其格式如下。

```
<% = Java 表达式% >
```

于是，circle. jsp 文件的代码可以修改如下。

```
<%@page contentType = "text/html;charset = utf - 8"% >
<html >
  <head >
      <title >
      计算圆的面积和周长
      </title >
  </head >
  <body >
  <%
      double pi = 3. 14, r = 10;
      double area, l;
      area = pi* r* r;
      l = 2* pi* r;
  % >
      圆的面积是: <% = area% >
  <br/ >
      圆的周长是: <% = l% >
  </body >
</html >
```

可以输出同样的运算结果。

2. 2. 4　JSP 应用举例

【例 2. 2】定义一个类 Student，该类有属性姓名 name 和学号 ID，有方法 setName( )，setID，getName( )，getID( )，分别用来设定和获取姓名、学号，通过类定义一个实例，从网页输入姓名、学号，调用方法在网页中输出姓名和学号。

1. 创建项目，名称为 Practice，具体步骤参考【例 2. 1】。

2. 右击工程目录下 WebRoot，选择【new】菜单下的【file】，在 File name 中输入 Student. jsp。

3. 输入代码。

```
<%@page contentType = "text/html;charset = utf - 8"% >
<%@page import = "java. io. * "% >
<font size = "4" >
<form action = " " method = "get" name = "form" >
请输入姓名: <input type = "text" name = "cat" value = " " >
<br/ > <br/ >
请输入学号: <input type = "text" name = "cat1" value = " " >
<br/ > <r/ >            
```

```
<input type = "submit" value = "提交" name = "submit" >
</form >
<%!
public class Student
{
   String name;
   String id;
void setName(String name)
{
      this. name = name;
}
void setId(String id)
{
      this. id = id;
}
String getName( )
{
   return this. name;
}
String getId( )
{
      return this. id;
   }
}
% >
<%
String strName = request. getParameter("cat");
String strId = request. getParameter("cat1");
double r;
Student a = new Student( );
a. setName(strName);
a. setId(strId);
% >
<p >学生的姓名是:
      <% = a. getName( )% >
</p > </font > <p > <font size = "4" >     学生的学号是:
      <% = a. getId( )% >
```

```
</font>
</p>
```

4. 部署、启动 Tomcat。

5. 在浏览器输入“http：//localhot：9080/Practice/Student. jsp”，出现图 2. 2 界面。

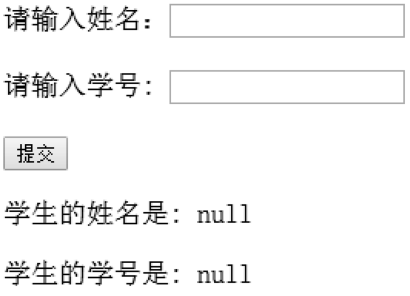


图 2. 2　例题 2. 2 图

6. 输入姓名：张三，学号：123456，出现图 2. 3 所示界面。

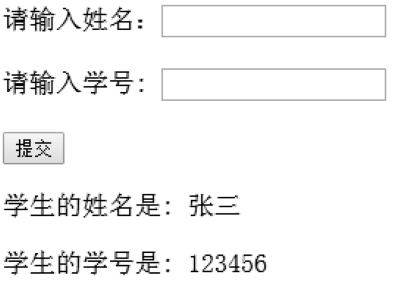


图 2. 3　例题 2. 2 图

第 3 章 JSP 动作指令

动作指令与编译指令不同，编译指令时通知 Servlet 引擎的处理消息，而动作指令只是运行时的动作。编译指令在将 JSP 编译成 Servlet 时起作用，而处理指令通常可替换成 JSP 脚本，它只是 JSP 脚本的标准化写法。

（1）jsp：forward 执行页面转向，将请求的处理转发到下一个页面。

（2）jsp：param 用于传递参数，必须与其他支持参数的标签一起使用。

（3）jsp：include 用于动态引入一个 JSP 页面。

（4）jsp：plugin 用于下载 JavaBean 或者 Applet 到客户端执行。

（5）jsp：useBean 创建一个 Javabean 实例。

（6）jsp：setProperty 设置 JavaBean 实例的属性值。

（7）jsp：getProperty 获取 JavaBean 实例的属性值。

下面我们就来讲述一下 JSP 常用的几种动作指令。

3.1 JSP 动作指令——<jsp:useBean>

用于在 JSP 页面中查找或者创建一个 JavaBean 实例，并通过属性设置将此实例存放在 JSP 指定范围内。使用 JavaBean 可以方便地在 JSP 中发挥 JavaBean 组件重用的优势。语法如下。

第一种方式：

<jsp:useBean id = "变量名"class = "类路径"scope = "存储范围"type = "数据类型" > 子动作 </jsp:useBean >

第二种方式：

<jsp:useBean id = "变量名" class = "类路径" scope = "存储范围" type = "数据类型"/ >

当 <jsp：useBean > 动作包含子动作时，使用第二种格式，否则可以使用第一种格式，其中，

id：指定查找或实例化的 JavaBean 的名称，可以通过 id 值访问已有的 JavaBean；

class：指定实例化 JavaBean 的类名，如果有包名，则需要加上对应的包，如果需要创建一个新的 JavaBean 实例，容器会根据 class 指定的类，调用其无参的构造方法完成Java

Bean 的实例化；

scope：设置 JavaBean 的作用范围，可以使用 page、request、session、application，默认为 page。

（1）page 可以在包含 <jsp：useBean > 动作的 JSP 文件以及该文件包含的所有静态资源中使用。

（2）request：在请求范围内有效，只能在用户的当前请求中使用。

（3）session：在当前 HttpSession 生命周期内使用，并创建 JavaBean 实例的 JSP 页面中 page 指令的 session 属性必须为 true。

（4）application：从创建 JavaBean 实例开始，可以在使用相同的 application 的所有 JSP 页面中使用该实例。Application 对象在应用服务器启动时被创建，服务器关闭时被释放。范围为 application 的 JavaBean 一旦被创建，只有当 application 被销毁时，才会被释放。

type：指定 JavaBean 对象的类型。其值可以与类名相同，或者是一个父类，或者是一个实现类的接口。如果是查找已存在的 JavaBean 对象，并且确定其存在，则 type 属性可以省略。

JavaBean 是一个可重复使用的软件组件，遵循特定的接口标准，对 Java 类的方法命名、行为、继承与实现等有特定的要求。

JavaBean 是 JavaWeb 应用的模型组件。

JSP 和 servlet 可以调用 JavaBean 完成业务逻辑的操作。

JSP 只支持非可视化的 JavaBean。

非可视化的 JavaBean 又可分为业务 Bean 和数据 Bean。业务 Bean 用来封装业务逻辑和数据库操作等，数据 Bean 用来封装数据。

利用 JavaBean 可以实现业务逻辑和前台应用程序的分离，实现代码的重用，从而提高软件的开发效率。

（1）必须是一个公共的类，即访问控制符为 public。

（2）必须包含有一个访问控制符为 public 的无参构造方法。

（3）属性都是私有的，即访问控制符为 private。

（4）属性通过一组存取方法来访问，在 eclipse 中可以自动生成属性的 get/set 方法。

学生实体 Bean 的示例代码如下。

```
package student;

public class Student {
    private  String name;
    private  String password;
    private  String email;
    public  String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPassword() {
        return password;
```

```
}
public void setPassword(String password) {
    this.password = password;
}
public String getEmail() {
    return email;
}
public void setEmail(String email) {
    this.email = email;
}
```

3.2 JSP 动作指令——<jsp:setProperty>

<jsp：setProperty>动作常和<jsp：useBean>动作一起使用，为 JavaBean 中属性赋值，可以通过以下四种方式赋值。

（1）<jsp:setProperty name = "id"property = "* "/>。

在提交数量较多的 JSP 页面中，一般使用这种方法接收并设置 JavaBean 中的属性值。当接收到一个 request 对象时，JSP 的内在机制会将 request 对象中的参数名和 JavaBean 中的属性进行匹配，并将名字相同的参数赋值给 JavaBean 的同名属性。使用这种方法时，要求 JavaBean 中的属性名和 request 对象中的参数名字匹配。如果参数和 JavaBean 中的属性值不匹配，会导致无法赋值。

（2）<jsp:setproperty name = "id" property = "属性名"value = "值"/>。

Value 值可以是一个字符串，也可以是表达式。这种方式采用特定的值给 JavaBean 中的特定属性赋值，要求 value 值和 JavaBean 中的属性类可以成功转化。

（3）<jsp:setProperty name = "id"property = "属性名"param = "参数名"/>。

property 指的是 JavaBean 中待赋值的属性名，param 值请求中的参数名，此参数可以来自于表单，也可以是来自于 URL 传递的参数。JSP 内部机制从 request 对象中取得相应的参数值，并赋值给 JavaBean 中指定的属性值。

注意：如果指定了 param 值，就不可以指定 value 值，即 param 和 value 不可以同时使用。

（4）<jsp:setProperty name = "id"property = "属性名"/>。

利用表单对 JavaBean 进行赋值时，通常使用这种方式。既没有指定 value，也没有指定 param，那么执行该动作的时候，会从 request 对象中寻找与属性名相同的参数值，如果有，则将该参数值赋值给 JavaBean 中的属性。

注意：如果利用表单给 JavaBean 赋值，除非使用 parma 指定参数，负责表单中的控件必须与 JavaBean 中的属性值相匹配。

3.3 JSP 动作指令——<jsp:getProperty>

<jsp：getProperty>用于获取 JavaBean 中的属性值，并且将属性值转化成字符串发送

到输入流中，并且显示到页面上。语法如下。

< jsp: getProperty name = "实例名"property = "属性名"/ >

Name：指定某一范围的 JavaBean 实例的引用。

Property：指定 JavaBean 属性名称。

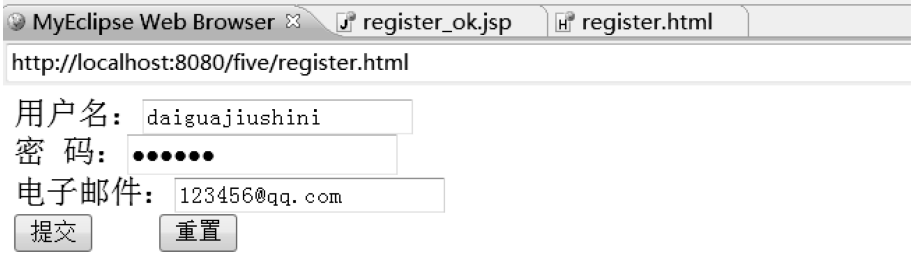
< jsp:setProperty > 动作、< jsp:useBean > 动作以及 < jsp:getProperty > 动作。

示例如下。

register. html 为学生信息的注册页面，表单提交给 register_ ok. jsp 页面处理。

register. html 页面代码如下。

```
<html>
  <head>
    <title> 学生信息注册 </title>
    <meta charset="utf-8">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
  </head>
  <body>
   <form action="/five/register_ok.jsp" method="post">
  用户名：<input type="text" name="name"><br>
  密　码：<input type="password" name="password"><br>
  电子邮件：<input type="text" name="email"><br>
            <input type="submit" value="提交">   
            <input type="reset" value="重置">
   </form>
  </body>
</html>
<!-- 注意：/five为项目名称 -->
```



运行结果如下：

register_ ok. jsp 获取名为 student 的 JavaBean 中的属性值并显示。

register_ ok. jsp 页面代码如下。

```
<html>
  <head>
    <base href="<%=basePath%>">
    <title>显示JavaBean属性的值</title>
  </head>
  <body>
    <jsp:useBean id="student" scope="page" class="student.Student"/>
    <jsp:setProperty property="*" name="student"/>
```

```
        使用getProperty获取JavaBean中的属性值并输出：<br>
        用户名：<jsp:getProperty property="name" name="student"/><br>
        密　码：<jsp:getProperty property="password" name="student"/><br>
        电子邮件：<jsp:getProperty property="email" name="student"/><br>
        <br>
        使用JavaBean的get方法获取属性值并输出：<br>
        用户名：<%=student.getName()%><br>
        密　码：<%=student.getPassword()%><br>
        电子邮件：<%=student.getEmail()%><br>
    </body>
</html>
```

运行结果如下。

3.4 JSP 动作指令——<jsp:include>

这个动作标签可以将另外一个文件内容包含到当前 JSP 页面中，被包含的页面可以是静态的，也可以是动态代码，语法如下。

<jsp:include page="url" flush="false|true"/>

或者是

<jsp:include page="url" flush="false|true">子动作</jsp:include>

page：该属性用于指定被包含文件的相对路径。

flush：可选参数，用于设置是否刷新缓冲区，如果为 true，则在当前页面输出使用了缓冲区的情况下，将先刷新缓冲区，然后再执行包含操作。如果为 false 则相反；默认为 false，当<jsp:include>动作包含子动作时，使用第二种格式，否则可使用第一种格式。

<jsp:include>动作和 include 指令的差别在于，include 动作包含的页面在运行时被加入，而 include 指令在编译的时候就被加入了。

<jsp:include>动作指令示例：

triangle.jsp 在页面上输出用“*”组成的直角三角形，include.jsp 包含 triangle.jsp 页面。

triangle.jsp 代码如下。

```
<html>
  <head>
    <base href="<%=basePath%>">
    <title>输出直角三角形</title>
  </head>
  <body>
     <%
     for(int i=1;i<=9;i++){
        for(int j=1;j<=i;j++){
        out.print("*");
        }
        out.print("<br>");
     }
      %>
  </body>
</html>
```

```
<html>
  <head>
    <base href="<%=basePath%>">
    <title>动作包含</title>
  </head>
  <body>
     <h4>输出直角三角形</h4>
     <jsp:include page="triangle.jsp"/>
  </body>
</html>
```

include. jsp 代码如下：

运行结果如下：

3. 5　JSP 动作指令——＜jsp：param＞

该标签可以作为其他标签的子标签，＜jsp：include＞或者＜jsp：forward＞为其标签传递参数。

语法如下。

＜jsp：param name＝”Name” value＝”Value”／＞

Name：用于设定参数名称。

图 3. 1　执行结果

Value：用于设定对应参数的值。

＜jsp：param＞参数标签示例：

在 third. jsp 中，向被转发的资源 fourth. jsp 传递两个参数 paramName 和 paramPwd，在 fourth. jsp 中使用 request. getParameter( )方法获取参数值。

third. jsp 页面代码如下。

```
<html>
  <head>
    <title>third.jsp</title>
  </head>
  <body>
    <h3>这是third.jsp页面</h3>
    <%
        request.setAttribute("name","zhangxiaoming");
     %>
     <jsp:forward page="fourth.jsp">
     <jsp:param value="liyang" name="paramName"/>
     <jsp:param value="123456" name="paramPwd"/>
     </jsp:forward>
  </body>
</html>
```

fourth. jsp 页面代码如下。

```
<html>
  <head>
    <base href="<%=basePath%>">

    <title>fourth.jsp</title>

  </head>

  <body>
    <h3>这是fourth.jsp页面</h3>
    request对象中的name值是:<%=request.getAttribute("name")%><br>
    param动作传递的参数paramName的值是:
    <%=request.getParameter("paramName") %><br>
    param动作传递的paramPwd的值是:
    <%=request.getParameter("paramPwd") %>
  </body>
</html>
```

运行结果如下。

3.6　JSP 动作指令——<jsp:forward>

这个动作标签是请求转发标签，可以将当前页面的请求转发给其他 Web 资源，可以是 JSP 页面、HTML 页面、Servlet 等。请求转发指的是停止执行当前的 JSP，而执行被转发的资源，转发后与转发前在同一个请求范围内，地址栏的地址并不会发生变化。

语法如下。

格式一：

<jsp:forward page = "url"/>

格式二：

<jsp:forward page = "url">子动作</jsp:forward>

Page 属性指明被转发的资源，当前 JSP 页面和被转发的资源必须位于相同的上下文环境中。当<jsp：forward>动作不包含子动作时，使用第一种格式，否则使用第二种格式。

注：forward 指令是一个重定向指令，所以 forward 指令下面的指令是不会被执行到的。

<jsp：forward>转发指令示例。

由 first. jsp 请求转发到 second. jsp，在 first. jsp 中向 request 对象中保存变量，在 second. jsp中读取变量 name，由于请求转发仍在同一个请求中，所以 request 对象中的变量仍然有效。

first. jsp 代码如下。

```
<html>
  <head>
    <base href="<%=basePath%>">
    <title>My JSP 'first.jsp' starting page</title>
  </head>
  <body>
    <h3>这是first.jsp页面</h3>
    <%
        request.setAttribute("name","wangxiaoming");
     %>
     <jsp:forward page="second.jsp"/>
  </body>
</html>
```

second. jsp 代码如下。

```
<html>
  </head>
  <body>
   <h3>这是second.jsp页面，欢迎您！</h3>
   request对象中的name值是:<%=request.getAttribute("name") %>
  </body>
</html>
```

运行结果如图 3. 2 所示。

图 3. 2　运行结果