1. Servlet基础

**1. Servlet基础**

**1.1. Web应用的演变**

**1.1.1. 单机程序**

软件从附着于电脑硬件之日起，就在不断的进行着自我完善和演变。从其使用模式的角度出发，可以简单分为单机程序和网络程序。发展到今时今日仍有大量的不依赖网络的单机程序被我们使用，如记事本、Excel、PPT、ZIP压缩等软件都是大家熟知的装机必备软件。

**1.1.2. 网络程序**

当电脑越来越多的参与到日常生产生活中，单机程序已经不能满足企业的需要。企业级应用要求能够最大程度的让更多的客户端参与到协同办公之中，所以依赖于网络的程序开始大力发展起来。最早的网络程序是基于主机+终端模式的，也就是整个应用中只有一台大型主机，各个操作地点都是使用一条专线与主机相连，终端不提供任何运算和界面，类似于Unix形式，所有的运算和处理都由主机来完成。主机一般处理能力非常强大，并且稳定，主要机型都是由IBM这样的大公司提供。但主机的高昂的价格以及扩展难、维护费用高等弊端并不是一般企业所能承受，所以除银行、航空订票、证券等大企业在使用以外，大多数企业开始转投CS架构的程序，即客户端服务器架构。

**1.1.3. 网络程序-CS架构**

CS架构的发展过程经历了两层CS架构，三层CS架构以及多层CS架构的演变。

两层的CS架构是由客户端和后面的数据库组成的。数据库用于存放数据，并且使用数据库编程语言编写业务逻辑，客户端则使用VB、VC、Delphi这样的可视化编程方便的语言来开发客户端的输入输出界面。用户通过界面向服务器发送请求，服务器发回的数据则通过界面进行显示，服务器的角色就由数据库来充当。这样做的好处就是开发效率高，满足企业需求。但是这种架构存在着很大的弊端，第一是可移植性差，如当数据库从SQL Server更换为Oracle时就必须将业务逻辑用新的语言再重新编写一遍；第二则是大型系统做不了，因为客户端与数据库需要建立持续的连接，而数据库能够支持的最大连接数是有限制的。所以在2000年这样的架构流行之后，慢慢的就开始向三层CS架构转变。

三层的CS架构指的是客户端+应用服务器+数据库，即将混合在数据库端的业务逻辑从中分离出来，放入到应用服务器中，数据库只负责数据的管理、存储及检索。客户端负责界面。三层之中的应用服务器其实也是程序，类似于前面讲过的TCP、Socket编程，任何支持TCP编程的语言都可以作为应用服务器。三层CS架构的工作流程如图 – 1 所示。

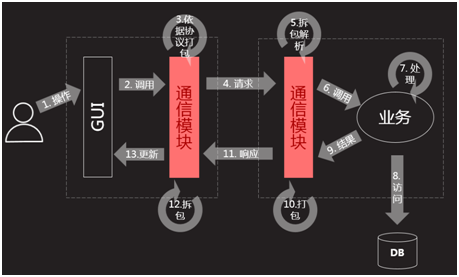


图- 1

用户通过GUI（图形用户界面）进行操作，然后调用客户端的通信模块，通信模块依据自定义协议将请求数据打包，通过网络发送该请求，到达应用服务器时，应用服务器同样也有一个通信模块，将收到的数据包按照协议进行拆包，调用相应的业务处理模块，处理数据，其中可能需要访问数据库来完成数据的获取，将处理完的结果再次发送给通信模块，通信模块将结果按照自定义协议进行打包，然后将数据包发送给客户端的通信模块，客户端进行拆包获取响应数据，将结果显示在界面上，更新界面上的数据显示。

这样的程序结构虽然在一定程度上降低了对数据库编程的依赖，并且能够适应大型的应用程序，但数据通信模块的增加却提升了开发的难度以及整体架构的复杂度。

**1.1.4. 网络程序-BS架构**

为了降低三层CS架构中与通信有关的复杂度，BS架构开始成为了网络程序中一大重要的架构类型。

BS架构即Browser + Web Server + DB。其工作流程如图 – 2所示。

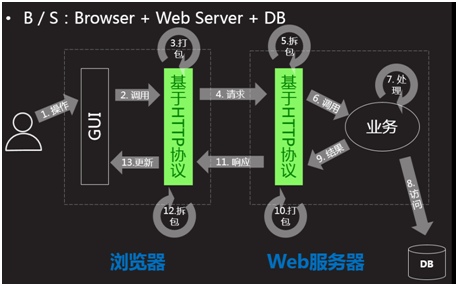


图- 2

由于三层CS架构中，自定义协议提升了整体的复杂度，那么就将自定义协议变成标准的HTTP协议。于是客户端使用HTTP协议进行数据打包拆包的程序即各厂商依据标准开发的浏览器，Web服务器也是基于HTTP协议由一些厂商提供，如IIS，Apache等。这样基于浏览器和服务器的架构中，由于协议已被限定，所以与通信有关的数据打包拆包的过程都不用我们开发人员来编写程序，只需要考虑将HTTP协议解析出来的数据进行业务处理，以及将什么样的结果提供给响应即可。也就是开发过程中只需要考虑7,8,9这三个步骤即可。于是大大降低了网络程序的开发难度，所以这种架构得到了大量的应用。

**1.2. 什么是Servlet**

**1.2.1. 扩展Web服务器功能**

在BS架构中，早期的Web服务器只能处理静态资源的请求，也就是无法根据请求进行计算后再生成相应的HTML内容。为了补充Web服务器的这个缺陷，于是增强服务器功能的CGI技术最早产生了。CGI（Common Gateway Interface通用网关接口）也是一种规范，可以使用不同的语言来开发，如Perl，C，Java等都可以。当客户端请求静态资源时，Web服务器会自己处理并返回，当客户端请求动态资源时，Web服务器会把请求转交给扩展程序来处理，并将扩展程序的处理结果返回给客户端。但是CGI技术开发复杂，性能较差，只要有一个请求到达，Web服务器就会单独分配一个进程来进行处理，可移植性不好，所以慢慢就由后来的Servlet技术所取代。

Servlet技术是使用Java语言开发的一套组件规范，不再像CGI技术那样需要分配单独的进程来处理请求，而是单独分配一个线程来处理，于是大大提升了处理效率。并且Java语言是跨平台的语言，也提升了Web服务器扩展程序的可移植性，已经取代了CGI技术，成为BS架构中的主流技术。所有后续的BS架构中的主流框架本质上都是基于Servlet来实现的。

**1.2.2. 组件规范**

组件规范是依靠一套API来实现的，也就是说开发中只要基于Sun公司提供的这套API，按照一定的规则来编写程序，那么就可以实现针对Web服务器的功能扩展。

但是组件只是对部分功能的一个实现，不能单独运行，必须放在一定的环境中才能运行。而这个针对各个组件进行管理、创建、销毁的运行环境即容器。

**1.2.3. Servlet组件运行原理**

Servlet作为补充Web服务器功能的组件，需要依赖于Servlet容器才能运行，它的运行原理如图 – 3所示。

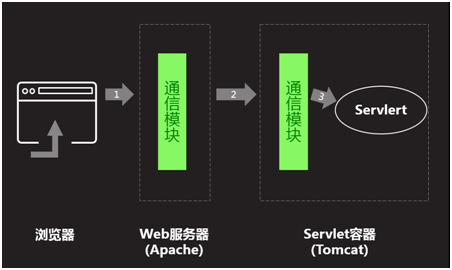


图- 3

在浏览器中输入请求地址后，浏览器会依据IP地址及端口号找到对应的Web服务器，如果请求的是静态资源，Web服务器直接提供响应；如果请求的是动态资源，Web服务器的通信模块会将该请求传递给Servlet容器的通信模块，Servlet容器负责创建Servlet实例，并将请求中的数据解析出来传递给Servlet。在Servlet处理完数据之后，响应结果也是由容器的通信模块负责返回给Web服务器。后续的Servlet的销毁及管理都由容器来负责。

能够充当Servlet容器这个角色的有很多软件，如Tomcat、Weblogic、JBoss等。而这些Servlet容器不仅仅具备了管理Servlet组件的功能，也具备了Web服务器的一些功能，所以很多时候只要安装一个Tomcat软件就同时具备了Web服务器及Servlet容器的双重功能。

**1.3. 如何开发一个Servlet**

**1.3.1. 安装Tomcat**

Servlet是Web服务器功能的补充，要想能够运行必须依赖Servlet容器的管理才可以，为了进行下一步的Servlet的具体开发，则需要先准备用于运行Servlet的环境，在学习过程中我们选择的是主流的Tomcat作为Web服务器及Servlet容器。

下面，简单介绍下Tomcat的安装步骤：

**步骤一：下载并解压安装文件**

在浏览器中输入 tomcat.apache.org，在Download菜单中选择Tomcat7.0后会得到如图- 4 所示的下载链接。

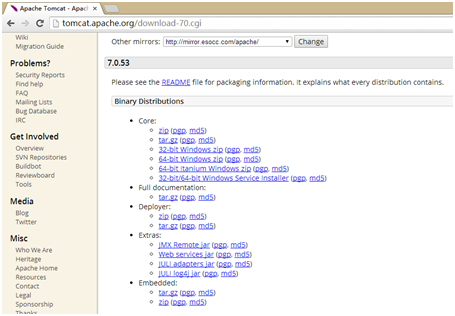


图- 4

选择Core下面符合当前操作系统的版本后下载，下载后的文件为压缩文件，解压到硬盘即可。假定Tomcat解压后的路径为 c:\java\tomcat7.0

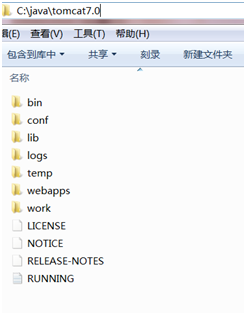


图- 5

**步骤二：配置环境变量**

选择“我的电脑”右键 (“属性”( “高级系统设置” ( “环境变量” 用于新建及修改一些环境变量，以保证系统记录解压的Tomcat软件的位置。如图-6所示：

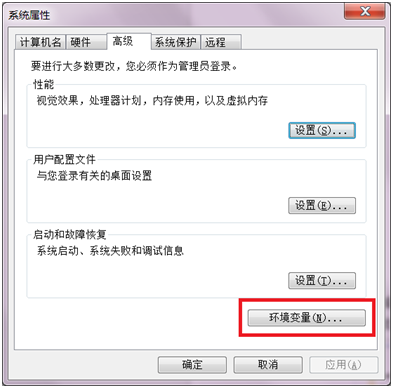


图- 6

点击系统变量下的“新建”按钮，创建“变量名”为 CATALINA\_HOME ， “变量值“ 为 c:\java\tomcat7.0的环境变量后点击确定按钮，如图- 7所示。

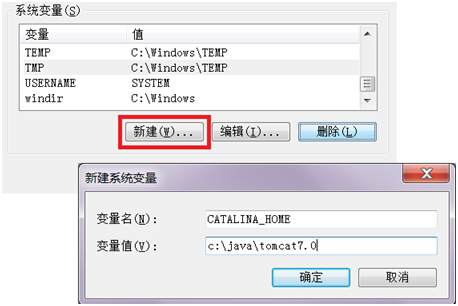


图- 7

选中系统变量中的“Path“后点击编辑，在原有内容的最前面添加 “ %CATALINA\_HOME%\bin; ” 后点击确定按钮。注意：修改值的时候光标放在最前面，并且要用分号将路径与原有Path的值分开。如图- 8所示。

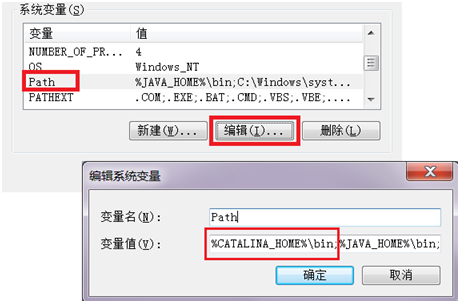


图- 8

若安装jdk时已配置了CLASS\_PATH变量，则选中系统变量中的“CLASS\_PATH“后点击编辑，在原有内容的最前面添加 “ %CATALINA\_HOME%\lib\servlet-api.jar; ” 后点击确定按钮。注意：修改值的时候光标放在最前面，并且要用分号将路径与原有CLASS\_PATH的值分开。如图- 9所示。

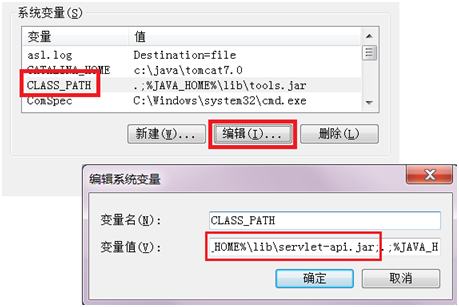


图- 9

至此，环境变量配置完毕。

**步骤三：启动Tomcat**

为了检测环境变量是否配置成功，以及Tomcat是否能够成功启动，首先进入命令行，windows系统下可以使用 “cmd“命令进入。在命令行窗口下输入“startup”命令，会得到如图 – 10 所示结果。

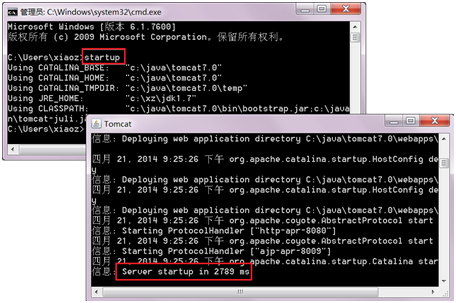


图- 10

打开浏览器，输入 “ http://localhost:8080 ”地址后，看到如图 – 11 所示结果。

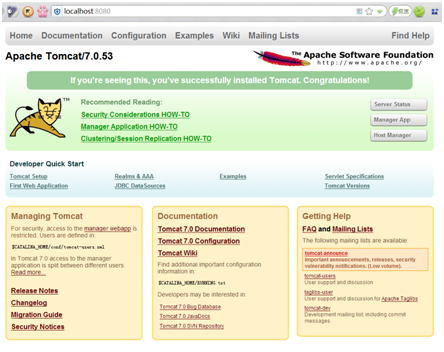


图- 11

至此，看到该页面即代表Tomcat启动成功。

**步骤四：关闭Tomcat**

在命令提示符下输入“shutdown”命令后，Tomcat的命令提示符窗口会关闭，在浏览器中再次输入http://localhost:8080后，看不到 图- 11，代表关闭Tomcat

**1.3.2. 开发Servlet的步骤**

**步骤一：编写一个实现Servlet接口或继承HttpServlet的Java类**

在硬盘上新建一个文件名为“HelloServlet.java”的源文件，并编写代码如下：

1. **package** web**;**
2. **import** java**.**io**.**IOException**;**
3. **import** java**.**io**.**PrintWriter**;**
4. **import** javax**.**servlet**.**ServletException**;**
5. **import** javax**.**servlet**.**http**.**HttpServlet**;**
6. **import** javax**.**servlet**.**http**.**HttpServletRequest**;**
7. **import** javax**.**servlet**.**http**.**HttpServletResponse**;**
8. **public** **class** HelloServlet **extends** HttpServlet **{**
9. **protected** **void** service**(**HttpServletRequest req**,** HttpServletResponse resp**)throws** ServletException**,** IOException **{**
10. resp**.**setContentType**(**"text/html"**);**
11. PrintWriter out **=** resp**.**getWriter**();**
12. out**.**write**(**"<h1>Hello Servlet"**);**
13. out**.**close**();**
14. **}**
15. **}**

**步骤二：使用javac命令编译源文件为字节码文件**

在命令行提示符下，进入到HelloServlet.java文件所在的位置，输入如下命令：

1. javac HelloServlet**.**java –cp c**:\**java**\**tomcat7**.0\**lib**\**servlet**-**api**.**jar

在没有报错的情况下会在java源文件的同级目录内出现HelloServlet.class的文件，代表编译成功。

**步骤三：将编译完的组件打包**

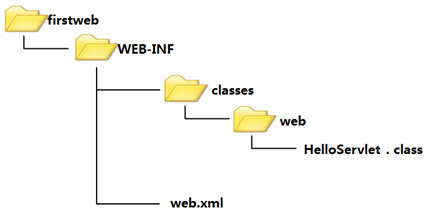


图- 12

按照如图 – 12 所示的结构将组建打包，并新建web.xml文件，内容如下：

1. **<?**xml version**=**"1.0" encoding**=**"UTF-8"**?>**
2. **<**web**-**app version**=**"2.5"
3. xmlns**=**"http://java.sun.com/xml/ns/javaee"
4. xmlns**:**xsi**=**"http://www.w3.org/2001/XMLSchema-instance"
5. xsi**:**schemaLocation**=**"http://java.sun.com/xml/ns/javaee
6. http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"**>**
7. **<**servlet**>**
8. **<**servlet**-**name**>**helloServlet**</**servlet**-**name**>**
9. **<**servlet**-class>**web**.**HelloServlet**</**servlet**-class>**
10. **</**servlet**>**
11. **<**servlet**-**mapping**>**
12. **<**servlet**-**name**>**helloServlet**</**servlet**-**name**>**
13. **<**url**-**pattern**>/**sayHi**</**url**-**pattern**>**
14. **</**servlet**-**mapping**>**
15. **</**web**-**app**>**

**步骤四：部署**

将整个firstweb文件夹拷贝到c:\java\tomcat7.0\webapps

**步骤五：启动Tomcat，并访问Servlet**

在命令提示符下输入“startup”命令启动Tomcat，启动成功后，打开浏览器，在地址栏中输入 “ http://localhost:8080/firstweb/sayHi “回车，得到第一个Web应用程序的运行结果。如图 – 13所示。



图- 13

**1.4. 常见错误及解决方法**

**1.4.1. 404错误产生原因及解决方法**

在刚开始进行Web应用开发的时候，经常看见页面出现404这个数字，我们一般会称之为运行产生了404错误。类似于404这个数字，还有可能在页面上看到405、500这两个数字，他们都是服务器执行完客户端的请求以后，返回给客户端的一个关于执行结果的状态编码说明。 如果在运行结果页面中没有看到期待的页面，却看到了404、405、500这样的数字，那么代表着服务器告诉客户端运行产生了错误，掌握何种错误情况产生对应的数字将有利于问题的解决。

404产生的原因为Web服务器（容器）根据请求地址找不到对应资源，以下情况都会出现404的错误提示：

1. 输入的地址有误（应用名大小写不正确，名称拼写不正确）
2. 在web.xml文件中<servlet>和<servlet-mapping>中的两个<servlet-name>不一致
3. 没有将工程部署到Tomcat的webapps下面
4. Web工程部署时的结构没有遵循Servlet的规范

具体的解决办法就是根据上面提到的4种情况，逐条进行检查。

**1.4.2. 容器如何找到service（）方法**

当在浏览器中输入 http://localhost:8080/firstweb/sayHi 这个地址后，容器是如何找到 HelloServlet.class这个文件并执行的呢？

首先容器会根据firstweb这个应用名找到位于webapps下面对应的文件夹，然后根据地址中的“/sayHi”到web.xml文件中寻找与之匹配的<url-pattern>节点，找到匹配的节点后会找到与该节点紧邻的<servlet-name>节点，获取名称并在此寻找与该名称相等的<servlet-name>节点，找到相等的节点后，搜寻该节点下面紧邻的<servlet-class>节点，于是获取到了与该地址相对应的类名，借助于ClassLoader加载该类文件，创建对象并调用service（）方法，客户端即看到了与该地址匹配的运行结果。

**1.4.3. 405错误产生原因及解决方法**

405这个错误的产生原因是容器找不到service方法来处理请求。以下情况容器都将视为没有找到service（）方法

1. service方法名写错，字母顺序不对或大小写错误
2. service方法参数类型与标准不一致，或参数顺序与标准不一致。一定是先HttpServletRequest，然后是HttpServletResponse类型
3. service方法异常、返回值类型与标准不一致，一定是抛出ServletException、IOException。返回值类型是void。

解决405错误的方法即检查service方法是否存在，签名（方法名、参数、返回值、异常类型）是否与覆盖的父类中的方法一致。

**1.4.4. 500错误产生原因及解决方法**

500的错误原因是请求的Servlet在执行service方法的过程中出现了错误，程序不能继续运行了。以下情况容器都将视为500的一种情况，而返回给客户端这个错误说明。

1. Servlet没有继承HttpServlet或实现Servlet接口
2. web.xml文件中的<servlet-class>写错，与真正的包名类名不符
3. service方法中的代码运行时抛出异常

解决500的方法为依据上面三种情况依次进行检测，逐条排除。

**1.5. HTTP协议**

**1.5.1. 什么是HTTP协议**

HTTP协议是HyperText Transfer Protocol的缩写，即超文本传输协议。是由w3c（万维网联盟）制定的一种应用层协议，用来定义浏览器与web服务器之间如何通信以及通信的数据格式。

因为BS架构中的通信模块就是以HTTP这个协议作为标准协议的，所以对该协议有所了解可以更好的编写程序。

**1.5.2. HTTP协议是如何通信的**

HTTP协议的通信过程可以分为以下四个步骤：

1. 浏览器根据IP地址和端口号与服务器建立连接
2. 向Web服务器发送请求数据包
3. Web服务器接收请求数据包后，发送相应的响应数据包
4. 浏览器接收响应数据后关闭连接

在这个过程中，一次请求对应一次连接，当浏览器再次发请求给服务器时，Web服务器并不知道这就是上次发请求的客户端，这也是HTTP协议的一个特点-无状态协议。这种需要时建立连接，使用结束后立即断开连接的方式使得Web服务器可以利用有限的连接为尽可能多的客户提供服务。也正是具备了这样的特点，才使得BS结构能够承载企业级应用的大量访问。

**1.5.3. HTTP协议中的数据格式**

在HTTP协议控制的数据走向中，既包括客户端发送给服务器端的请求数据也包括服务器端返回给客户端的响应信息，而具有一定规范的数据格式是保证通信标准的第一要素。借助于一些浏览器的插件或者浏览器本身内嵌的功能模块，可以实现对请求数据、响应数据的抓取，熟练掌握这些数据的主要部分，可以有利于理解BS模型中的一些处理问题的方式。通常情况下的请求数据包及响应数据包结构如图 – 14所示。

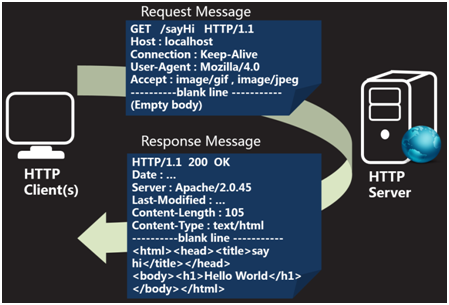


图- 14

**1.5.4. 请求数据包组成**

请求数据包包含三个部分：

1. 请求行，包括请求方式，请求资源路径，以及协议版本
2. 消息头是由若干的键值对组成，主要用于传递给Web服务器一些信息。如，浏览器版本、接受资源的类型等
3. 在消息头的下面会跟着一个空白行，这样就可以区分出消息头与实体内容这两个部分。而在请求数据包中，大部分情况实体内容为空，只有当请求方式为POST时，实体内容才有要提交的表单数据

请求数据包常见结构如图 – 15 所示。

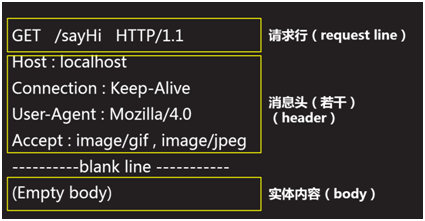


图- 15

**1.5.5. 响应数据包组成**

响应数据包也包含三个部分：

1. 状态行，包括协议类型、版本、状态码、状态描述
2. 消息头（若干），也是多组键值对组成的信息说明。如返回的Content-Type消息头可以告诉浏览器返回的结果数据类型和字符集
3. 实体内容，位于消息头下面的一个空白行之后。空白行意在通知浏览器消息头的结束以及结果内容的开始，浏览器就从这里开始数据的逐行解析。

响应数据包常见结构如图 – 16 所示。

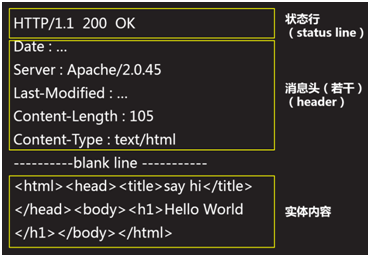


图- 16

**1.6. Servlet如何处理HTTP协议**

**1.6.1. 如何控制通信数据**

HTTP协议在Web容器这端主要表现为通信数据的到达以及响应数据的返回。于是Web容器将这两部分数据解释为两个对象，一个是与请求数据对应的HttpServletRequest对象，一个是与响应数据对应的HttpServletResponse对象。对于Servlet来讲，主要的业务逻辑过程就是从请求对象中获取数据，经过加工后将结果附着在响应对象中发送回客户端。

**1.6.2. HttpServletRequest对象**

借助于HttpServletRequest对象可以实现很多操作，如读取请求行、消息头信息，取得路径信息等。使用如下代码可以实现相关功能。

1. **package** web**;**
2. **import** java**.**io**.**IOException**;**
3. **import** java**.**io**.**PrintWriter**;**
4. **import** java**.**util**.**Arrays**;**
5. **import** java**.**util**.**Enumeration**;**
6. **import** javax**.**servlet**.**ServletException**;**
7. **import** javax**.**servlet**.**http**.**HttpServlet**;**
8. **import** javax**.**servlet**.**http**.**HttpServletRequest**;**
9. **import** javax**.**servlet**.**http**.**HttpServletResponse**;**
10. **public** **class** RequestServlet **extends** HttpServlet **{**
12. **protected** **void** service**(**HttpServletRequest req**,** HttpServletResponse resp**)**
13. **throws** ServletException**,** IOException **{**
14. Enumeration e **=** req**.**getHeaderNames **();**
15. **while** **(**e**.**hasMoreElements **()){**
16. String headerName **=** e**.**nextElement **().**toString **();**
17. System**.**out**.**println **(**headerName**+**":"**+**req**.**getHeader **(**headerName**));**
18. **}**
19. System**.**out**.**println **(**req**.**getMethod **());**
20. System**.**out**.**println **(**req**.**getProtocol **());**
21. System**.**out**.**println **(**req**.**getRequestURI **());**
22. System**.**out**.**println **(**req**.**getRequestURL **());**
23. System**.**out**.**println **(**req**.**getServletPath **());**
24. **}**
25. **}**

**1.6.3. HttpServletResponse对象**

HttpServletResponse对象最主要的作用即设置给浏览器的响应内容及浏览器的解码方式，此外设置cookie及重定向也都是响应对象的功能。这些功能对应的代码实现如下所示。

1. **package** web**;**
2. **import** java**.**io**.**IOException**;**
3. **import** java**.**io**.**PrintWriter**;**
4. **import** javax**.**servlet**.**ServletException**;**
5. **import** javax**.**servlet**.**http**.**HttpServlet**;**
6. **import** javax**.**servlet**.**http**.**HttpServletRequest**;**
7. **import** javax**.**servlet**.**http**.**HttpServletResponse**;**
8. **public** **class** ResponseServlet **extends** HttpServlet **{**
10. **protected** **void** service**(**HttpServletRequest req**,** HttpServletResponse resp**)**
11. **throws** ServletException**,** IOException **{**
12. resp**.**setContentType**(**"text/html;charset=gbk"**);**
13. resp**.**setStatus**(404);**
14. **}**
16. **}**

**JAVA SERVLETJSP DAY01**

1. 安装和配置Tomcat
2. 手动开发第一个Web项目
3. 使用工具开发Web项目
4. 输出请求数据包中的内容
5. 向客户端输出中文

**1 安装和配置Tomcat**

**1.1 问题**

windows系统下实现Tomcat的安装和配置。

**1.2 步骤**

**步骤0：安装JDK及配置JAVA\_HOME、PATH等环境变量**

到Oracle官方网站下载JDK的Windows专用安装程序后，执行安装，假定安装路径为“d:\java\jdk1.7”，则配置环境变量过程如下：

“我的电脑”右键“属性”(高级系统设置(环境变量。如图-12，图-13，图-14所示：



图 - 12

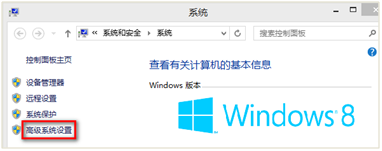


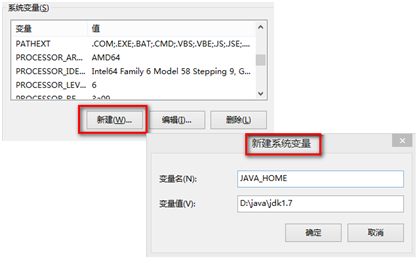
图 -13



图– 14

对“系统变量”做以下三个操作：

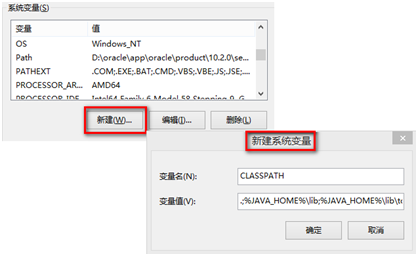
新建系统变量，变量名为“JAVA\_HOME”，变量值为“D：\java\jdk1.7”。即JDK的安装路径的根路径。如图-15所示：



图– 15

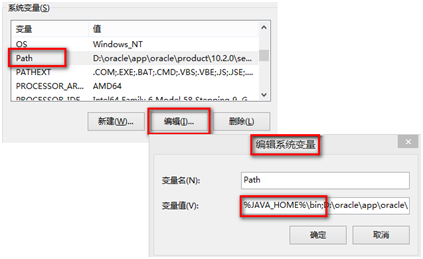
新建系统变量，变量名为“CLASSPATH”，变量值为“.；%JAVA\_HOME%\lib;%JAVA\_HOME\

lib\tools.jar”。第一个为“.”点，代表当前路径，即在当前路径下寻找类，后面为安装的jdk中的jar文件的路径，中间用“；“分号隔开如图-16所示。



图– 16

修改原有系统变量“Path“，将光标移动到变量值的最前面（注意，一定不要删除原有的变量值），添加”%JAVA\_HOME%\bin;”。一定要以分号结束，作为与原有值的分隔标识。安装的Tomcat的bin目录下有很多可执行程序和命令，记录这个地址后，不管在任何目录下输入java和javac命令，系统都能找到对应的命令并执行。



图– 17

通过以上三个环境变量的设置以后，为了验证JDK是否配置成功，则通过在命令行中输入命令来检测。在“运行”窗口中输入“cmd”进入到命令行界面，输入“java -version”查看输出结果，如果提示“java version 1.7.X\_XX”则代表配置成功。如图-18所示。

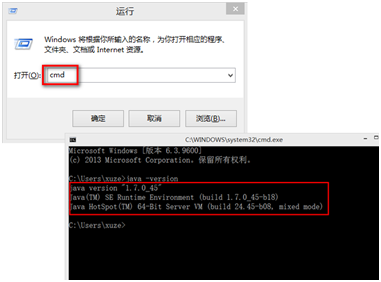


图 - 18

**步骤一：下载及解压Tomcat**

在apache网站的指定下载页面，选择适合Windows操作系统的Tomcat版本后下载。如图-19所示。

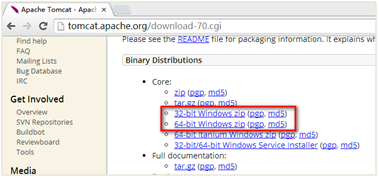


图 - 19

将下载的文件解压缩到指定路径。假定本次解压路径为“d:\java\tomcat7”，结果如图-20所示。注意“tomcat7”下面不再包含子文件夹，直接包含tomcat的各文件。

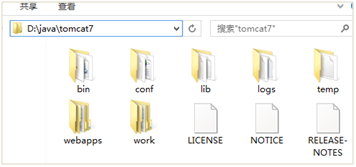
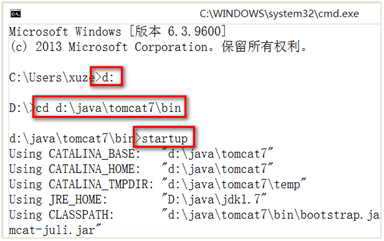


图 -20

**步骤二：启动Tomcat**

使用命令行工具，输入“d:”切换到D盘，再输入“cd d:\java\tomcat7\bin“进入到tomcat文件夹中的bin目录下，输入“startup”命令即可启动Tomcat。如图-21,22所示。



图– 21

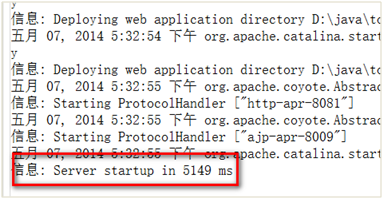


图 - 22

打开浏览器输入”http://localhost:8080“验证启动成功。如图-23。

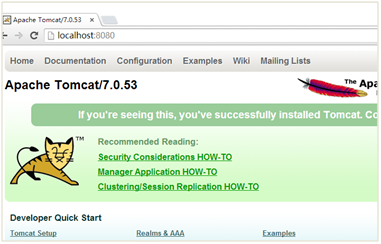


图 - 23

**步骤三：关闭Tomcat**

在命令行中输入“shutdown”命令可以关闭Tomcat。如图-24所示。

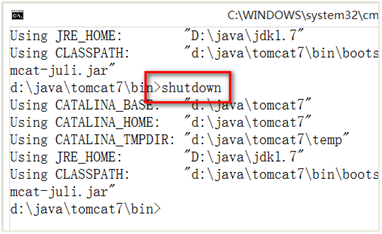


图 - 24

**2 手动开发第一个Web项目**

**2.1 问题**

使用文本编辑工具开发第一个Servlet程序。

**2.2 步骤**

开发Servlet程序的步骤如下

1.编写java源文件，文件中的类必须实现Servlet接口或继承HttpServlet类

2 . 编译java源文件，生成class文件

3.打包，将编译后的文件按照规定目录结构进行整理

4 . 部署，将目录结构复制到Web容器的指定位置

5 . 启动Web容器，输入符合规范的地址访问Servlet

按照以上五步即可完成Web程序的开发。

***Windows下开发第一个Web程序***

**步骤一：编写源文件**

新建java源文件，假定保存路径为“d:\java”，内容如图-33所示。

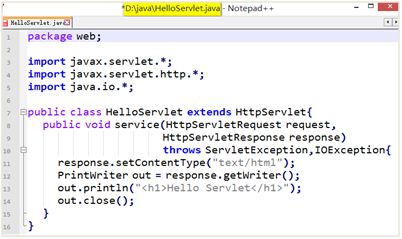
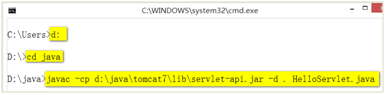


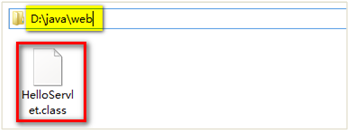
图 - 33

**步骤二：编译HelloServlet.java文件**

进入到命令行工具，输入命令“d:”切换盘符，输入命令“cd java”进入到HelloServlet.java源文件所在的目录，输入命令“javac –cp d:\java\tomcat7\lib\servlet-api.jar –d . HelloServlet.java”编译java文件到当前目录下。命令及输出结果如图-34，图-35所示。



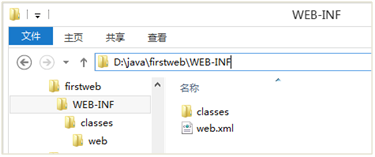
图– 34



图– 35

**步骤三：打包**

按照如下结构进行整理应用的目录结构。



图– 36

详细结构可以参考图-29。web.xml文件内容参考图-30。

**步骤四：部署**

将整个firstweb文件夹拷贝到tomcat安装路径下的webapps文件夹下。如图-37所示。

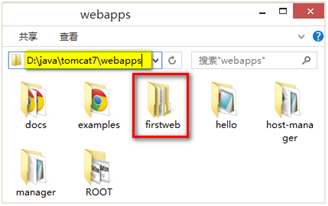
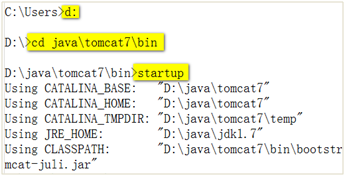


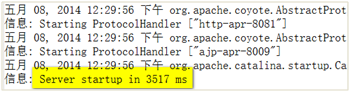
图 - 37

**步骤五：启动容器，访问Servlet查看结果**

进入到命令行工具，输入“d:”切换盘符，输入“cd java\tomcat7\bin”进入到tomcat的工具目录，输入“startup”回车，启动tomcat。如图-38，图-39所示。



图– 38



图– 39

打开浏览器，输入“http://localhost:8080/firstweb/hello”查看运行结果。



图 - 40

**2.3 完整代码**

本案例的完整代码如下所示：

HelloServlet.java代码：

package web;

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.io.\*;

public class HelloServlet extends HttpServlet{

    public void service(HttpServletRequest request,

                                    HttpServletResponse response)

                                    throws ServletException,IOException{

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        out.println("<h1>Hello Servlet</h1>");

        out.close();

    }

}

web.xml文件代码：

**<?**xml version**=**"1.0" encoding**=**"UTF-8"**?>**

**<**web**-**app xmlns**=**"http://java.sun.com/xml/ns/javaee"

xmlns**:**xsi**=**"http://www.w3.org/2001/XMLSchema-instance"

xsi**:**schemaLocation**=**"http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"

version**=**"3.0"

metadata**-**complete**=**"true"**>**

**<**servlet**>**

**<**servlet**-**name**>**helloServlet**</**servlet**-**name**>**

**<**servlet**-class>**web**.**HelloServlet**</**servlet**-class>**

**</**servlet**>**

**<**servlet**-**mapping**>**

**<**servlet**-**name**>**helloServlet**</**servlet**-**name**>**

**<**url**-**pattern**>**/hello</url**-**pattern**>**

**</**servlet**-**mapping**>**

**</**web**-**app**>**

**3 使用工具开发Web项目**

**3.1 问题**

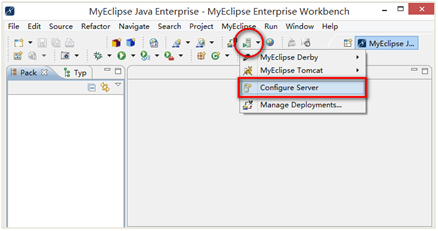
使用MyEclipse进行Web应用的开发。

**3.2 步骤**

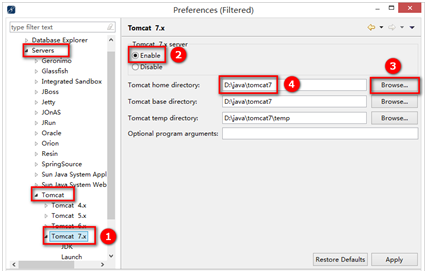
在第一次使用MyEclipse进行Web应用的开发之前，需要将MyEclipse与解压后的Tomcat进行关联，也就是完成服务器的配置。

**步骤一：为MyEclipse添加服务器配置**

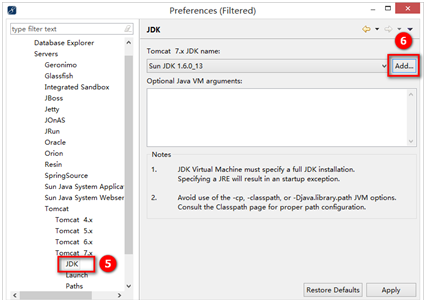
依据图-41至图-45所示步骤完成服务器配置



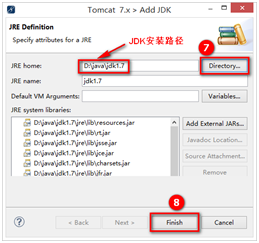
图– 41



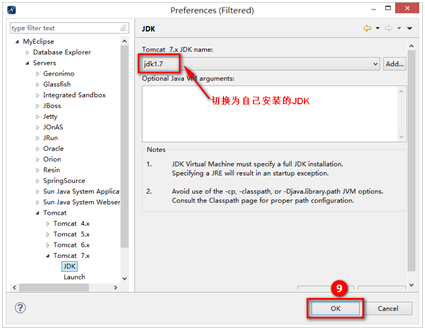
图– 42



图– 43

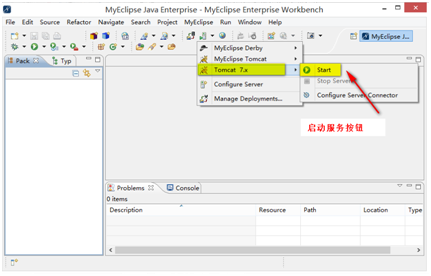


图– 44



图– 45

配置成功后结果如图-46所示。



图– 46

可以通过点击“Start”按钮启动关联到的Tomcat，观察Console输出窗口会出现“Server startup in XXXX ms”的提示信息，代表服务器配置成功并能够正确启动。通过点击“Stop Server”按钮可以停止服务。

**步骤二：新建Web工程**

通过点击菜单“File”(“New”(“Web Project”新建一个Web工程，填写工程名信息如图-47后点击确定，生成工程。

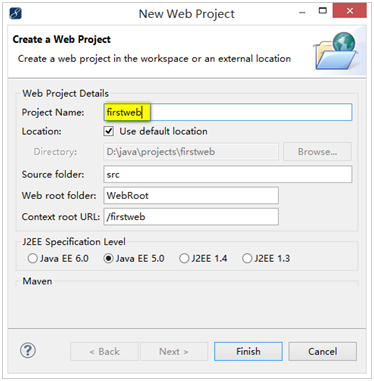
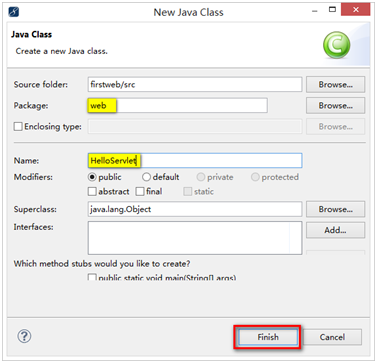


图 - 47

**步骤三：创建java类**

在工程文件的“src”上右键，new一个class文件，填写包名“web”，填写类名“HelloServlet”后点击“Finish”，如图-48所示。

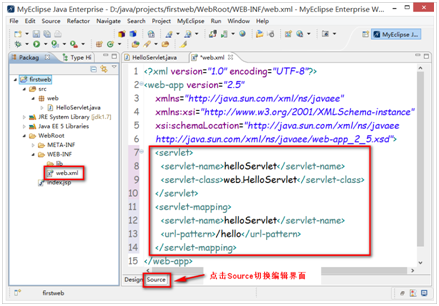


图– 48

类文件内容参考图-33。

**步骤四：配置web.xml文件**

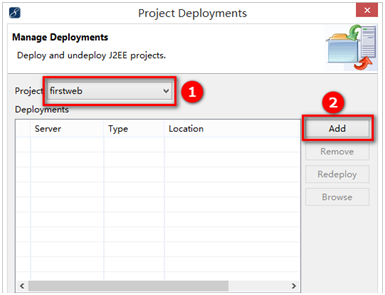
点击工程中“WEB-INF”节点下的web.xml文件，点击屏幕下方的“Source”按钮切换编辑视图，改写文件内容如图-49所示。



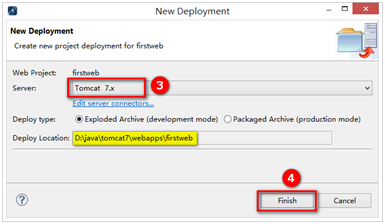
图– 49

**步骤五：部署工程到Tomcat**

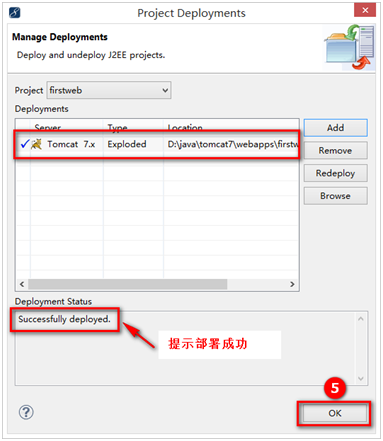
点击工具栏中的/图标进入到部署工程的界面，按照图-50至图-52所示顺序依次选择要部署的项目、部署到哪个服务器之后点击OK按钮完成部署。



图– 50



图– 51



图– 52

**步骤六：启动服务器查看运行结果**

点击工具栏中的/服务器图标下面“Tomcat7.X”对应的“Start”按钮启动服务器，观察控制台输出窗口中显示成功启动的信息后，打开浏览器，输入“http://localhost:8080/firstweb/hello“，查看运行结果。同图-40显示一样。

**3.3 完整代码**

与“手动开发第一个Web项目“的完整代码相同。

***使用MyEclipse开发Web项目的过程与手动编写完成的步骤是一样的，只是将一些步骤变成了自动完成，如编译、打包***

* 编写java源文件 手动
* 编译 自动
* 打包 自动
* 部署 手动
* 启动服务，查看 手动

**4 输出请求数据包中的内容**

**4.1 问题**

获取请求数据包中的消息头中各键值对的名称和值，提交方式，协议，请求资源路径等信息，并输出到控制台中。

**4.2 方案**

客户端发来的请求数据包到达服务器端以后会被解释为HttpServletRequest类型的对象，数据包中的数据会被存储到该类型的对象之中，所以借助于该对象提供的getHeaderNames(),getHeader(),getMethod(),getRequestURI()等方法即可获取这些信息。

**4.3 步骤**

实现此案例按如下步骤进行。

**步骤一：新建web工程**

点击菜单“文件”(“新建”(“Web Project”，填写信息如图-53所示，一般填写完工程名以后其他信息默认不用修改，部署的应用名会与工程名保持一致，但如果修改了部署名称以后，那么部署之后访问时注意资源路径要与修改后的名称保持一致。

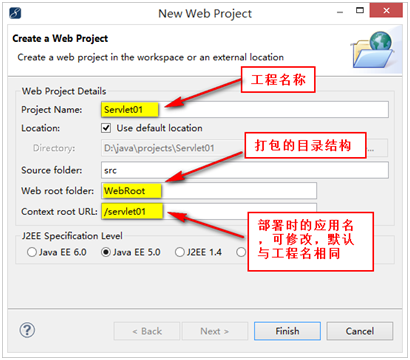


图 - 53

**步骤二：新建类RequestInfo.java**

新建一个位于web包下面的RequestInfo.java的类，使得该类继承自HttpServlet抽象类，重写service（）方法，代码如下。

1. package web**;**
2. import java**.**io**.**IOException**;**
3. import javax**.**servlet**.**ServletException**;**
4. import javax**.**servlet**.**http**.**HttpServlet**;**
5. import javax**.**servlet**.**http**.**HttpServletRequest**;**
6. import javax**.**servlet**.**http**.**HttpServletResponse**;**
7. **public** **class** RequestInfo **extends** HttpServlet**{**
8. **public** void service**(**HttpServletRequest request**,**HttpServletResponse response**)**
9. **throws** ServletException**,**IOException**{**

12. **}**
13. **}**

**步骤三：调用request对象的方法获取消息头信息**

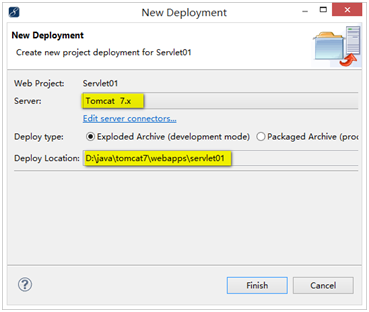
1. package web**;**
2. import java**.**io**.**IOException**;**
3. import java**.**util**.**Enumeration**;**
4. import javax**.**servlet**.**ServletException**;**
5. import javax**.**servlet**.**http**.**HttpServlet**;**
6. import javax**.**servlet**.**http**.**HttpServletRequest**;**
7. import javax**.**servlet**.**http**.**HttpServletResponse**;**
8. **public** **class** ReqInfoServlet **extends** HttpServlet**{**
9. **public** void service**(**HttpServletRequest request**,**HttpServletResponse response**)**
10. **throws** ServletException**,**IOException**{**
11. //获取所有消息头的名称
12. Enumeration e **=** request**.**getHeaderNames**();**
13. **while(**e**.**hasMoreElements**()){**
14. //遍历Enumeration获取每一个消息头的名称
15. String headerName **=** e**.**nextElement**().**toString**();**
16. //输出消息头的名-值对儿信息
17. System**.**out**.**println**(**headerName**+**":"**+**request**.**getHeader**(**headerName**));**
18. **}**
19. System**.**out**.**println**(**"请求方式："**+**request**.**getMethod**());**
20. System**.**out**.**println**(**"请求的协议种类："**+**request**.**getProtocol**());**
21. System**.**out**.**println**(**"请求资源路径："**+**request**.**getRequestURI**());**
22. System**.**out**.**println**(**"请求的路径信息："**+**request**.**getRequestURL**());**
23. System**.**out**.**println**(**"请求的Servlet路径："**+**request**.**getServletPath**());**
24. **}**
25. **}**

**步骤四：修改web.xml文件**

1. **<?**xml version**=**"1.0" encoding**=**"UTF-8"**?>**
2. **<**web**-**app version**=**"2.5"
3. xmlns**=**"http://java.sun.com/xml/ns/javaee"
4. xmlns**:**xsi**=**"http://www.w3.org/2001/XMLSchema-instance"
5. xsi**:**schemaLocation**=**"http://java.sun.com/xml/ns/javaee
6. http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"**>**
7. **<**servlet**>**
8. **<**servlet**-**name**>**reqInfoServlet**</**servlet**-**name**>**
9. **<!--** 类名要写完整，包含包名，注意大小写 **-->**
10. **<**servlet**-class>**web**.**ReqInfoServlet**</**servlet**-class>**
11. **</**servlet**>**
12. **<**servlet**-**mapping**>**
13. **<**servlet**-**name**>**reqInfoServlet**</**servlet**-**name**>**
14. **<**url**-**pattern**>**/reqInfo</url**-**pattern**>**
15. **</**servlet**-**mapping**>**
16. **</**web**-**app**>**

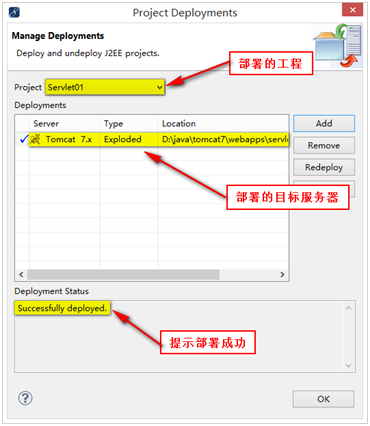
**步骤五：部署**

点击工具栏上的“部署”图标，选择要部署的工程及“Add”按钮，选择要部署到的目标服务器后点击Finish。如图-54：



图– 54

提示部署成功后点击OK，如图-55：



图– 55

**步骤六：启动服务，访问应用**

通过点击工具栏中服务器图标，选择Tomcat7.X，点击Start启动服务器。打开浏览器在地址栏中输入“http://localhost:8080/servlet01/reqInfo”查看结果。如图-56：

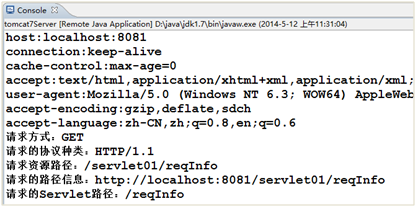


图 - 56

**4.4 完整代码**

ReqInfoServlet.java完整代码如下：

1. package web**;**
2. import java**.**io**.**IOException**;**
3. import java**.**util**.**Enumeration**;**
4. import javax**.**servlet**.**ServletException**;**
5. import javax**.**servlet**.**http**.**HttpServlet**;**
6. import javax**.**servlet**.**http**.**HttpServletRequest**;**
7. import javax**.**servlet**.**http**.**HttpServletResponse**;**
8. **public** **class** ReqInfoServlet **extends** HttpServlet**{**
9. **public** void service**(**HttpServletRequest request**,**HttpServletResponse response**)**
10. **throws** ServletException**,**IOException**{**
12. //获取所有消息头的名称
13. Enumeration e **=** request**.**getHeaderNames**();**
14. **while(**e**.**hasMoreElements**()){**
15. //遍历Enumeration获取每一个消息头的名称
16. String headerName **=** e**.**nextElement**().**toString**();**
17. //输出消息头的名-值对儿信息
18. System**.**out**.**println**(**headerName**+**":"**+**request**.**getHeader**(**headerName**));**
19. **}**
21. System**.**out**.**println**(**"请求方式："**+**request**.**getMethod**());**
22. System**.**out**.**println**(**"请求的协议种类："**+**request**.**getProtocol**());**
23. System**.**out**.**println**(**"请求资源路径："**+**request**.**getRequestURI**());**
24. System**.**out**.**println**(**"请求的路径信息："**+**request**.**getRequestURL**());**
25. System**.**out**.**println**(**"请求的Servlet路径："**+**request**.**getServletPath**());**
26. **}**
27. **}**

web.xml文件完成代码如下：

1. **<?**xml version**=**"1.0" encoding**=**"UTF-8"**?>**
2. **<**web**-**app version**=**"2.5"
3. xmlns**=**"http://java.sun.com/xml/ns/javaee"
4. xmlns**:**xsi**=**"http://www.w3.org/2001/XMLSchema-instance"
5. xsi**:**schemaLocation**=**"http://java.sun.com/xml/ns/javaee
6. http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"**>**
7. **<**servlet**>**
8. **<**servlet**-**name**>**reqInfoServlet**</**servlet**-**name**>**
9. **<!--** 类名要写完整，包含包名，注意大小写 **-->**
10. **<**servlet**-class>**web**.**ReqInfoServlet**</**servlet**-class>**
11. **</**servlet**>**
12. **<**servlet**-**mapping**>**
13. **<**servlet**-**name**>**reqInfoServlet**</**servlet**-**name**>**
14. **<**url**-**pattern**>**/reqInfo</url**-**pattern**>**
15. **</**servlet**-**mapping**>**
16. **</**web**-**app**>**

**5 向客户端输出中文**

**5.1 问题**

实现访问Servlet之后，服务器端能够返回带有中文信息的页面。

**5.2 方案**

通过response对象修改响应数据包中“content-type”消息头，从而设置浏览器的解码方式为支持中文信息的方式。

**5.3 步骤**

**步骤一：新建类ResInfoServlet.java**

在web包下新建类ResInfoServlet.java,在service方法中添加设置响应头的代码。如下所示：

1. package web**;**
2. import java**.**io**.**IOException**;**
3. import java**.**io**.**PrintWriter**;**
4. import javax**.**servlet**.**ServletException**;**
5. import javax**.**servlet**.**http**.**HttpServlet**;**
6. import javax**.**servlet**.**http**.**HttpServletRequest**;**
7. import javax**.**servlet**.**http**.**HttpServletResponse**;**
8. **public** **class** ResInfoServlet **extends** HttpServlet**{**
9. **public** void service**(**HttpServletRequest request**,**HttpServletResponse response**)**
10. **throws** ServletException**,**IOException**{**
11. response**.**setContentType**(**"text/html;charset=utf-8"**);**
12. PrintWriter out **=** response**.**getWriter**();**
13. out**.**print**(**"能够输出中文信息的Servlet"**);**
14. out**.**close**();**
15. **}**
16. **}**

**步骤二：修改web.xml，配置Servlet**

在web.xml文件中增加对ResInfoServlet的配置。代码如下所示：

1. **<**servlet**>**
2. **<**servlet**-**name**>**resInfoServlet**</**servlet**-**name**>**
3. **<**servlet**-class>**web**.**ResInfoServlet**</**servlet**-class>**
4. **</**servlet**>**
5. **<**servlet**-**mapping**>**
6. **<**servlet**-**name**>**resInfoServlet**</**servlet**-**name**>**
7. **<**url**-**pattern**>**/resInfo</url**-**pattern**>**
8. **</**servlet**-**mapping**>**

**步骤三：重新部署工程，访问应用**

重新部署工程后，在浏览器中输入合法地址查看页面结果如下：

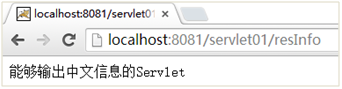


图 - 57

**5.4 完整代码**

ResInfoServlet.java文件的完整代码如下：

1. package web**;**
2. import java**.**io**.**IOException**;**
3. import java**.**io**.**PrintWriter**;**
4. import javax**.**servlet**.**ServletException**;**
5. import javax**.**servlet**.**http**.**HttpServlet**;**
6. import javax**.**servlet**.**http**.**HttpServletRequest**;**
7. import javax**.**servlet**.**http**.**HttpServletResponse**;**
8. **public** **class** ResInfoServlet **extends** HttpServlet**{**
9. **public** void service**(**HttpServletRequest request**,**HttpServletResponse response**)**
10. **throws** ServletException**,**IOException**{**
11. response**.**setContentType**(**"text/html;charset=utf-8"**);**
12. PrintWriter out **=** response**.**getWriter**();**
13. out**.**print**(**"能够输出中文信息的Servlet"**);**
14. out**.**close**();**
15. **}**
16. **}**

web.xml文件的完整代码如下：

1. **<?**xml version**=**"1.0" encoding**=**"UTF-8"**?>**
2. **<**web**-**app version**=**"2.5"
3. xmlns**=**"http://java.sun.com/xml/ns/javaee"
4. xmlns**:**xsi**=**"http://www.w3.org/2001/XMLSchema-instance"
5. xsi**:**schemaLocation**=**"http://java.sun.com/xml/ns/javaee
6. http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"**>**
7. **<**servlet**>**
8. **<**servlet**-**name**>**reqInfoServlet**</**servlet**-**name**>**
9. **<!--** 类名要写完整，包含包名，注意大小写 **-->**
10. **<**servlet**-class>**web**.**ReqInfoServlet**</**servlet**-class>**
11. **</**servlet**>**
12. **<**servlet**>**
13. **<**servlet**-**name**>**resInfoServlet**</**servlet**-**name**>**
14. **<**servlet**-class>**web**.**ResInfoServlet**</**servlet**-class>**
15. **</**servlet**>**
16. **<**servlet**-**mapping**>**
17. **<**servlet**-**name**>**reqInfoServlet**</**servlet**-**name**>**
18. **<**url**-**pattern**>**/reqInfo</url**-**pattern**>**
19. **</**servlet**-**mapping**>**
20. **<**servlet**-**mapping**>**
21. **<**servlet**-**name**>**resInfoServlet**</**servlet**-**name**>**
22. **<**url**-**pattern**>**/resInfo</url**-**pattern**>**
23. **</**servlet**-**mapping**>**
24. **</**web**-**app**>**