[**HTTP头部详解及使用Java套接字处理HTTP请求**](http://blog.csdn.net/overmaker/article/details/2194921)

标签： [java](http://www.csdn.net/tag/java)[http服务器](http://www.csdn.net/tag/http%e6%9c%8d%e5%8a%a1%e5%99%a8)[浏览器](http://www.csdn.net/tag/%e6%b5%8f%e8%a7%88%e5%99%a8)[服务器](http://www.csdn.net/tag/%e6%9c%8d%e5%8a%a1%e5%99%a8)[socket](http://www.csdn.net/tag/socket)[url](http://www.csdn.net/tag/url)

2008-03-18 17:39 9572人阅读 [评论](http://blog.csdn.net/overmaker/article/details/2194921#comments)(1) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/overmaker/article/details/2194921#report)

分类：



java*（7）*



[作者同类文章](http://blog.csdn.net/overmaker/article/category/364859)*X*

进行Web开发关键是要了解超文本传输协议（HTTP），该协议用来传输网页、图像以及因特网上在浏览器与服务器间传输的其他类型文件。只要你在浏览器上输入一个URL，最前面的http://就表示使用HTTP来访问指定位置的信息。（大部分浏览器还支持其他一些不同的协议，其中FTP就是一个典型例子。）

本文从HTTP协议的结构上初步探讨HTTP协议的工作原理和请求响应格式,并最后通过一个使用Java编写的小HTTP服务器验证了如何处理和响应HTTP请求

HTTP由两部分组成：请求和响应。当你在Web浏览器中输入一个URL时，浏览器将根据你的要求创建并发送请求，该请求包含所输入的URL以及一些与浏览器本身相关的信息。当服务器收到这个请求时将返回一个响应，该响应包括与该请求相关的信息以及位于指定URL（如果有的话）的数据。直到浏览器解析该响应并显示出网页（或其他资源）为止。

**HTTP请求**

HTTP请求的格式如下所示：

＜request-line＞

＜headers＞

＜blank line＞

[＜request-body＞]

在HTTP请求中，第一行必须是一个请求行（request line），用来说明请求类型、要访问的资源以及使用的HTTP版本。紧接着是一个首部（header）小节，用来说明服务器要使用的附加信息。在首部之后是一个空行，再此之后可以添加任意的其他数据[称之为主体（body）]。

在HTTP中，定义了大量的请求类型，不过Ajax开发人员关心的只有GET请求和POST请求。只要在Web浏览器上输入一个URL，浏览器就将基于该URL向服务器发送一个GET请求，以告诉服务器获取并返回什么资源。对于www.wrox.com的GET请求如下所示：

GET / HTTP/1.1

Host: www.wrox.com

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)

Gecko/20050225 Firefox/1.0.1

Connection: Keep-Alive

请求行的第一部分说明了该请求是GET请求。该行的第二部分是一个斜杠（/），用来说明请求的是该域名的根目录。该行的最后一部分说明使用的是HTTP 1.1版本（另一个可选项是1.0）。那么请求发到哪里去呢？这就是第二行的内容。

第2行是请求的第一个首部，HOST。首部HOST将指出请求的目的地。结合HOST和上一行中的斜杠（/），可以通知服务器请求的是www.wrox.com/（HTTP 1.1才需要使用首部HOST，而原来的1.0版本则不需要使用）。第三行中包含的是首部User-Agent，服务器端和客户端脚本都能够访问它，它是浏览器类型检测逻辑的重要基础。该信息由你使用的浏览器来定义（在本例中是Firefox 1.0.1），并且在每个请求中将自动发送。最后一行是首部Connection，通常将浏览器操作设置为Keep-Alive（当然也可以设置为其他值，但这已经超出了本书讨论的范围）。注意，在最后一个首部之后有一个空行。即使不存在请求主体，这个空行也是必需的。

如果要获取一个诸如http://www.wrox.com/books的www.wrox.com域内的页面，那么该请求可能类似于：

GET /books/ HTTP/1.1

Host: www.wrox.com

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)

Gecko/20050225 Firefox/1.0.1

Connection: Keep-Alive

注意只有第一行的内容发生了变化，它只包含URL中www.wrox.com后面的部分。

要发送GET请求的参数，则必须将这些额外的信息附在URL本身的后面。其格式类似于：

URL ? name1=value1&name2=value2&..&nameN=valueN

该信息称之为查询字符串（query string），它将会复制在HTTP请求的请求行中，如下所示：

GET /books/?name=Professional%20Ajax HTTP/1.1

Host: www.wrox.com

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)

Gecko/20050225 Firefox/1.0.1

Connection: Keep-Alive

注意，为了将文本“Professional Ajax”作为URL的参数，需要编码处理其内容，将空格替换成%20，这称为URL编码（URL encoding），常用于HTTP的许多地方（JavaScript提供了内建的函数来处理URL编码和解码，这些将在本章中的后续部分中说明）。“名称—值”（name—value）对用 & 隔开。绝大部分的服务器端技术能够自动对请求主体进行解码，并为这些值的访问提供一些逻辑方式。当然，如何使用这些数据还是由服务器决定的。

浏览器发送的首部，通常比本文中所讨论的要多得多。为了简单起见，这里的例子尽可能简短。

另一方面，POST请求在请求主体中为服务器提供了一些附加的信息。通常，当填写一个在线表单并提交它时，这些填入的数据将以POST请求的方式发送给服务器。

以下就是一个典型的POST请求：

POST / HTTP/1.1

Host: www.wrox.com

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)

Gecko/20050225 Firefox/1.0.1

Content-Type: application/x-www-form-urlencoded

Content-Length: 40

Connection: Keep-Alive

name=Professional%20Ajax&publisher=Wiley

从上面可以发现， POST请求和GET请求之间有一些区别。首先，请求行开始处的GET改为了POST，以表示不同的请求类型。你会发现首部Host和User-Agent仍然存在，在后面有两个新行。其中首部Content-Type说明了请求主体的内容是如何编码的。浏览器始终以application/ x-www-form- urlencoded的格式编码来传送数据，这是针对简单URL编码的MIME类型。首部Content-Length说明了请求主体的字节数。在首部Connection后是一个空行，再后面就是请求主体。与大多数浏览器的POST请求一样，这是以简单的“名称—值”对的形式给出的，其中name是Professional Ajax，publisher是Wiley。你可以以同样的格式来组织URL的查询字符串参数。

正如前面所提到的，还有其他的HTTP请求类型，它们遵从的基本格式与GET请求和POST请求相同。下一步我们来看看服务器将对HTTP请求发送什么响应。

**HTTP响应**

如下所示，HTTP响应的格式与请求的格式十分类似：

＜status-line＞

＜headers＞

＜blank line＞

[＜response-body＞]

正如你所见，在响应中唯一真正的区别在于第一行中用状态信息代替了请求信息。状态行（status line）通过提供一个状态码来说明所请求的资源情况。以下就是一个HTTP响应的例子：

HTTP/1.1 200 OK

Date: Sat, 31 Dec 2005 23:59:59 GMT

Content-Type: text/html;charset=ISO-8859-1

Content-Length: 122

＜html＞

＜head＞

＜title＞Wrox Homepage＜/title＞

＜/head＞

＜body＞

＜!-- body goes here --＞

＜/body＞

＜/html＞

在本例中，状态行给出的HTTP状态代码是200，以及消息OK。状态行始终包含的是状态码和相应的简短消息，以避免混乱。最常用的状态码有：

◆200 (OK): 找到了该资源，并且一切正常。

◆304 (NOT MODIFIED): 该资源在上次请求之后没有任何修改。这通常用于浏览器的缓存机制。

◆401 (UNAUTHORIZED):客户端无权访问该资源。这通常会使得浏览器要求用户输入用户名和密码，以登录到服务器。

◆403 (FORBIDDEN):客户端未能获得授权。这通常是在401之后输入了不正确的用户名或密码。

◆404 (NOT FOUND):在指定的位置不存在所申请的资源。

在状态行之后是一些首部。通常，服务器会返回一个名为Data的首部，用来说明响应生成的日期和时间（服务器通常还会返回一些关于其自身的信息，尽管并非是必需的）。接下来的两个首部大家应该熟悉，就是与POST请求中一样的Content-Type和Content-Length。在本例中，首部Content-Type指定了MIME类型HTML（text/html），其编码类型是ISO-8859-1（这是针对美国英语资源的编码标准）。响应主体所包含的就是所请求资源的HTML源文件（尽管还可能包含纯文本或其他资源类型的二进制数据）。浏览器将把这些数据显示给用户。

注意，这里并没有指明针对该响应的请求类型，不过这对于服务器并不重要。客户端知道每种类型的请求将返回什么类型的数据，并决定如何使用这些数据。

**附录:使用Java套接字实现一个可以处理get和post请求的小HTTP服务器程序**

/\*\* \*//\*\*  
 \* SimpleHttpServer.java  
 \*/  
  
import java.io.\*;  
import java.net.\*;  
import java.util.StringTokenizer;  
  
/\*\* \*//\*\*  
 \* 一个简单的用 Java Socket 编写的 HTTP 服务器应用, 演示了请求和应答的协议通信内容以及  
 \* 给客户端返回 HTML 文本和二进制数据文件(一个图片), 同时展示了 404, 200 等状态码.  
 \* 首先运行这个程序,然后打开Web浏览器,键入http://localhost,则这个程序能够显示出浏览器发送了那些信息  
 \* 并且向浏览器返回一个网页和一副图片, 并测试同浏览器对话.  
 \* 当浏览器看到 HTML 中带有图片地址时, 则会发出第二次连接来请求图片等资源.  
 \* 这个例子可以帮您理解 Java 的 HTTP 服务器软件是基于 J2SE 的 Socket 等软件编写的概念, 并熟悉  
 \* HTTP 协议.  
 \* 相反的用 Telnet 连接到已有的服务器则可以帮忙理解浏览器的运行过程和服务器端的返回内容.  
 \*  
 \* <pre>  
 \*       当用户在Web浏览器地址栏中输入一个带有http://前缀的URL并按下Enter后,或者在Web页面中某个以http://开头的超链接上单击鼠标,HTTP事务处理的第一个阶段--建立连接阶段就开始了.HTTP的默认端口是80.  
 \*    随着连接的建立,HTTP就进入了客户向服务器发送请求的阶段.客户向服务器发送的请求是一个有特定格式的ASCII消息,其语法规则为:  
 \* < Method > < URL > < HTTP Version > < >  
 \* { <Header>:<Value> < >}\*  
 \* < >  
 \* { Entity Body }  
 \*    请求消息的顶端是请求行,用于指定方法,URL和HTTP协议的版本,请求行的最后是回车换行.方法有GET,POST,HEAD,PUT,DELETE等.  
 \* 在请求行之后是若干个报头(Header)行.每个报头行都是由一个报头和一个取值构成的二元对,报头和取值之间以":"分隔;报头行的最后是回车换行.常见的报头有Accept(指定MIME媒体类型),Accept\_Charset(响应消息的编码方式),Accept\_Encoding(响应消息的字符集),User\_Agent(用户的浏览器信息)等.  
 \*    在请求消息的报头行之后是一个回车换行,表明请求消息的报头部分结束.在这个之后是请求消息的消息实体(Entity Body).具体的例子参看httpRequest.txt.  
 \*     Web服务器在收到客户请求并作出处理之后,要向客户发送应答消息.与请求消息一样,应答消息的语法规则为:  
 \* < HTTP Version> <Status Code> [<Message>]< >  
 \* { <Header>:<Value> < > } \*  
 \* < >  
 \* { Entity Body }  
 \*    应答消息的第一行为状态行,其中包括了HTTP版本号,状态码和对状态码进行简短解释的消息;状态行的最后是回车换行.状态码由3位数字组成,有5类:   
 \* 参看:HTTP应答码及其意义   
 \*   
 \* 1XX 保留   
 \* 2XX 表示成功   
 \* 3XX 表示URL已经被移走   
 \* 4XX 表示客户错误   
 \* 5XX 表示服务器错误   
 \* 例如:415,表示不支持改媒体类型;503,表示服务器不能访问.最常见的是200,表示成功.常见的报头有:Last\_Modified(最后修改时间),Content\_Type(消息内容的MIME类型),Content\_Length(内容长度)等.  
 \*    在报头行之后也是一个回车换行,用以表示应答消息的报头部分的结束,以及应答消息实体的开始.  
 \*    下面是一个应答消息的例子:  
 \* HTTP/1.0 200 OK  
 \* Date: Moday,07-Apr-97 21:13:02 GMT  
 \* Server:NCSA/1.1  
 \* MIME\_Version:1.0  
 \* Content\_Type:text/html  
 \* Last\_Modified:Thu Dec 5 09:28:01 1996  
 \* Coentent\_Length:3107  
 \*   
 \* <HTML><HEAD><TITLE></HTML>  
 \*   
 \* 在用Java语言实现HTTP服务器时,首先启动一个java.net.ServerSocket在提供服务的端口上监听连接.向客户返回文本时,可以用PrintWriter,但是如果返回二进制数据,则必须使用OutputStream.write(byte[])方法,返回的应答消息字符串可以使用String.getBytes()方法转换为字节数组返回,或者使用PrintStream的print()方法写入文本,用write(byte[])方法写入二进制数据.  
 \*   
 \* </pre>  
 \* @author 刘长炯  
 \* @version 1.0 2007-07-24 Sunday  
 \*/  
public class SimpleHttpServer implements Runnable ...{  
    /\*\* \*//\*\*  
     \*   
     \*/  
    ServerSocket serverSocket;//服务器Socket  
      
    /\*\* \*//\*\*  
     \* 服务器监听端口, 默认为 80.  
     \*/  
    public static int PORT=80;//标准HTTP端口  
      
    /\*\* \*//\*\*  
     \* 开始服务器 Socket 线程.  
     \*/  
    public SimpleHttpServer() ...{  
        try ...{  
            serverSocket=new ServerSocket(PORT);  
        } catch(Exception e) ...{  
            System.out.println("无法启动HTTP服务器:"+e.getLocalizedMessage());  
        }  
        if(serverSocket==null)  System.exit(1);//无法开始服务器  
        new Thread(this).start();  
        System.out.println("HTTP服务器正在运行,端口:"+PORT);  
    }  
      
    /\*\* \*//\*\*  
     \* 运行服务器主线程, 监听客户端请求并返回响应.  
     \*/  
    public void run() ...{  
        while(true) ...{  
            try ...{  
                Socket client=null;//客户Socket  
                client=serverSocket.accept();//客户机(这里是 IE 等浏览器)已经连接到当前服务器  
                if(client!=null) ...{  
                    System.out.println("连接到服务器的用户:"+client);  
                    try ...{  
                        // 第一阶段: 打开输入流  
                        BufferedReader in=new BufferedReader(new InputStreamReader(  
                                client.getInputStream()));  
                          
                        System.out.println("客户端发送的请求信息: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
                        // 读取第一行, 请求地址  
                        String line=in.readLine();  
                        System.out.println(line);  
                        String resource=line.substring(line.indexOf('/'),line.lastIndexOf('/')-5);  
                        //获得请求的资源的地址  
                        resource=URLDecoder.decode(resource, "UTF-8");//反编码 URL 地址  
                        String method = new StringTokenizer(line).nextElement().toString();// 获取请求方法, GET 或者 POST  
  
                        // 读取所有浏览器发送过来的请求参数头部信息  
                        while( (line = in.readLine()) != null) ...{  
                            System.out.println(line);  
                              
                            if(line.equals("")) break;  
                        }  
                          
                        // 显示 POST 表单提交的内容, 这个内容位于请求的主体部分  
                        if("POST".equalsIgnoreCase(method)) ...{  
                            System.out.println(in.readLine());  
                        }  
                          
                        System.out.println("请求信息结束 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
                        System.out.println("用户请求的资源是:"+resource);  
                        System.out.println("请求的类型是: " + method);  
  
                        // GIF 图片就读取一个真实的图片数据并返回给客户端  
                        if(resource.endsWith(".gif")) ...{  
                            fileService("images/test.gif", client);  
                            closeSocket(client);  
                            continue;  
                        }  
                          
                        // 请求 JPG 格式就报错 404  
                        if(resource.endsWith(".jpg")) ...{  
                                                    PrintWriter out=new PrintWriter(client.getOutputStream(),true);  
                        out.println("HTTP/1.0 404 Not found");//返回应答消息,并结束应答  
                        out.println();// 根据 HTTP 协议, 空行将结束头信息  
                        out.close();  
                        closeSocket(client);  
                        continue;  
                        } else ...{  
                            // 用 writer 对客户端 socket 输出一段 HTML 代码  
                            PrintWriter out=new PrintWriter(client.getOutputStream(),true);  
                            out.println("HTTP/1.0 200 OK");//返回应答消息,并结束应答  
                            out.println("Content-Type:text/html;charset=GBK");  
                            out.println();// 根据 HTTP 协议, 空行将结束头信息  
  
                            out.println("<h1> Hello Http Server</h1>");  
                            out.println("你好, 这是一个 Java HTTP 服务器 demo 应用.<br>");  
                            out.println("您请求的路径是: " + resource + "<br>");  
                            out.println("这是一个支持虚拟路径的图片:<img src='abc.gif'><br>" +  
                                    "<a href='abc.gif'>点击打开abc.gif, 是个服务器虚拟路径的图片文件.</a>");  
                            out.println("<br>这是个会反馈 404 错误的的图片:<img src='test.jpg'><br><a href='test.jpg'>点击打开test.jpg</a><br>");  
                            out.println("<form method=post action='/'>POST 表单 <input name=username value='用户'> <input name=submit type=submit value=submit></form>");  
                            out.close();  
  
                            closeSocket(client);  
                        }  
                    } catch(Exception e) ...{  
                        System.out.println("HTTP服务器错误:"+e.getLocalizedMessage());  
                    }  
                }  
                //System.out.println(client+"连接到HTTP服务器");//如果加入这一句,服务器响应速度会很慢  
            } catch(Exception e) ...{  
                System.out.println("HTTP服务器错误:"+e.getLocalizedMessage());  
            }  
        }  
    }  
      
    /\*\* \*//\*\*  
     \* 关闭客户端 socket 并打印一条调试信息.  
     \* @param socket 客户端 socket.  
     \*/  
    void closeSocket(Socket socket) ...{  
        try ...{  
            socket.close();  
        } catch (IOException ex) ...{  
            ex.printStackTrace();  
        }  
                            System.out.println(socket + "离开了HTTP服务器");          
    }  
      
    /\*\* \*//\*\*  
     \* 读取一个文件的内容并返回给浏览器端.  
     \* @param fileName 文件名  
     \* @param socket 客户端 socket.  
     \*/  
        void fileService(String fileName, Socket socket)  
    ...{  
              
        try  
        ...{  
            PrintStream out = new PrintStream(socket.getOutputStream(), true);  
            File fileToSend = new File(fileName);  
            if(fileToSend.exists() && !fileToSend.isDirectory())  
            ...{  
                out.println("HTTP/1.0 200 OK");//返回应答消息,并结束应答  
                out.println("Content-Type:application/binary");  
                out.println("Content-Length:" + fileToSend.length());// 返回内容字节数  
                out.println();// 根据 HTTP 协议, 空行将结束头信息  
                  
                FileInputStream fis = new FileInputStream(fileToSend);  
                byte data[] = new byte[fis.available()];  
                fis.read(data);  
                out.write(data);  
                out.close();  
                fis.close();  
            }  
        }  
        catch(Exception e)  
        ...{  
            System.out.println("传送文件时出错:" + e.getLocalizedMessage());  
        }  
    }  
      
    /\*\* \*//\*\*  
     \* 打印用途说明.  
     \*/  
    private static void usage() ...{  
        System.out.println("Usage: java HTTPServer <port> Default port is 80.");  
    }  
      
      
    /\*\* \*//\*\*  
     \* 启动简易 HTTP 服务器  
     \* @param args   
     \*/  
    public static void main(String[] args) ...{  
        try ...{  
            if(args.length != 1) ...{  
                usage();  
            } else if(args.length == 1) ...{  
                PORT = Integer.parseInt(args[0]);  
            }  
        } catch (Exception ex) ...{  
            System.err.println("Invalid port arguments. It must be a integer that greater than 0");  
        }  
          
        new SimpleHttpServer();  
    }  
      
}

