相信大多Web开发者对Tomcat是非常熟悉的，众所周知Tomcat是一款非常好用的开源Servlet容器，您一定对这个最流行的Servlet容器充满好奇，虽然它并不像一个黑盒子那样让人无法触摸但是Tomcat的源码的确让人看起来头疼。笔者就在这里和大家共同分析一个简单的Web服务器是如何工作。

**Web服务器**

Web服务器是一个复杂的系统，**一个Web服务器要为一个Servlet的请求提供服务，需要做三件事：**

1、**创建一个request对象并填充那些有可能被所引用的Servlet使用的信息，如参数、头部、cookies、查询字符串等等。**一个request对象是javax.servlet.ServletRequest或javax.servlet.http.ServletRequest接口的一个实例

2、**创建一个response对象，所引用的servlet使用它来给客户端发送响应。**一个response对象是javax.servlet.ServletRequest或javax.servlet.http.ServletRequest接口的一个实例。

3、**调用servlet的service方法，并传入request和response对象。这里servlet会从request对象取值，给response写值。**

在正式展示代码之前还需要了解一些必须额HTTP的知识（如果您对此非常熟悉您可以直接看下面分析代码）

**HTTP**

HTTP的定义不知道的童鞋可以自己去度娘，这里主要要说的就是HTTP协议的格式

HTTP请求包括三部分

1、方法、统一资源标识符（URI）、协议/版本

2、请求的头部

3、主题内容

下面是一个HTTP请求的例子

**[plain]** [view plaincopyprint?](http://blog.csdn.net/beijiguangyong/article/details/7439352)

1. POST /examples/default.jsp HTTP/1.1
2. Accept: text/plain; text/html
3. Accept-Language: en-gb
4. Connection: Keep-Alive
5. Host: localhost
6. User-Agent: Mozilla/4.0 (compatible; MSIE 4.01; Windows 98)
7. Content-Length: 33
8. Content-Type: application/x-www-form-urlencoded
9. Accept-Encoding: gzip, deflate
11. lastName=Franks&firstName=Michael

第一行表明这是POST请求方法，/examples/default.jsp是URI，HTTP/1.1是协议以及版本。其中URI指明了一个互联网资源，这里通常是相对服务器根目录解释的，也就是说这个HTTP请求就是告诉服务器我需要这个文件目录如下：根目录/ examples/default.jsp。

最后一行是HTTP的主题内容，Servlet会处理请求的主题内容，然后返回给客户端HTTP响应。

类似于HTTP请求，一个HTTP响应也包括上面三个部分。

1、方法、统一资源标识符（URI）、协议/版本

2、响应的头部

3、主题内容

下面是一个HTTP响应的例子

**[plain]** [view plaincopyprint?](http://blog.csdn.net/beijiguangyong/article/details/7439352)

1. HTTP/1.1 200 OK
2. Server: Microsoft-IIS/4.0
3. Date: Mon, 5 Jan 2004 13:13:33 GMT
4. Content-Type: text/html
5. Last-Modified: Mon, 5 Jan 2004 13:13:12 GMT
6. Content-Length: 112
8. <html>
9. <head>
10. <title>HTTP Response Example</title>
11. </head>
12. <body>
13. Welcome to Brainy Software
14. </body>
15. </html>

第一行告诉协议版本，以及请求成功（200表示成功）

响应头部和请求头部一样，一些有用的信息。响应的主体就是响应本身HTML内容。

好了基本知识介绍完毕，下面开始解释代码

**部分相关代码**

**[java]** [view plaincopyprint?](http://blog.csdn.net/beijiguangyong/article/details/7439352)

1. import java.net.Socket;
2. import java.net.ServerSocket;
3. import java.net.InetAddress;
4. import java.io.InputStream;
5. import java.io.OutputStream;
6. import java.io.IOException;
7. import java.io.File;
9. public class HttpServer {
11. public static final String WEB\_ROOT = System.getProperty("user.dir")
12. + File.separator + "webroot";
14. private static final String SHUTDOWN\_COMMAND = "/SHUTDOWN";
16. private boolean shutdown = false;
18. public static void main(String[] args) {
19. HttpServer server = new HttpServer();
20. server.await();
21. }
23. public void await() {
24. ServerSocket serverSocket = null;
25. int port = 8080;
26. try {
27. **serverSocket = new ServerSocket(port, 1,**
28. **InetAddress.getByName("127.0.0.1"));**
29. } catch (IOException e) {
30. e.printStackTrace();
31. System.exit(1);
32. }
34. **while (!shutdown)**{
35. Socket socket = null;
36. InputStream input = null;
37. OutputStream output = null;
38. try {
39. **socket = serverSocket.accept();**
40. input = socket.getInputStream();
41. output = socket.getOutputStream();
43. **Request request = new Request(input);**
44. **request.parse();**
46. **Response response = new Response(output);**
47. **response.setRequest(request);**
48. **response.sendStaticResource();**
50. socket.close();
52. **shutdown = request.getUri().equals(SHUTDOWN\_COMMAND);**
53. } catch (Exception e) {
54. e.printStackTrace();
55. continue;
56. }
57. }
58. }
59. }

HttpServer类代表一个web服务器。首先提供一个WEB\_ROOT所在的目录和它下面所有的子目录下静态资源。其次定义了一个中止服务的命令，也就是说当得到的请求后面跟/shutdown的时候停止服务，默认是把服务设置为开启。下面就是进入main函数了，首先实例化一个HttpServer类，然后就是通过await方法等待客户端发来的请求。如果客户端输入的URL不是http://localhost:8080/SHUTDOWN则表示不停止服务器，然后就是**继续执行await方法中的内容，**在await方法中最重要的就是定义两个对象，一个是request一个是response，下面就来说说Request和Response类。

**[java]** [view plaincopyprint?](http://blog.csdn.net/beijiguangyong/article/details/7439352)

1. import java.io.InputStream;
2. import java.io.IOException;
4. public class Request {
6. private InputStream input;
7. private String uri;
9. public Request(InputStream input) {
10. this.input = input;
11. }
13. public void parse() {
15. StringBuffer request = new StringBuffer(2048);
16. int i;
17. byte[] buffer = new byte[2048];
18. try {
19. i = input.read(buffer);
20. } catch (IOException e) {
21. e.printStackTrace();
22. i = -1;
23. }
24. for (int j = 0; j < i; j++) {
25. request.append((char) buffer[j]);
26. }
27. System.out.print(request.toString());
28. **uri = parseUri(request.toString());**
29. }
31. private String parseUri(String requestString) {
32. int index1, index2;
33. index1 = requestString.indexOf(' ');
34. if (index1 != -1) {
35. index2 = requestString.indexOf(' ', index1 + 1);
36. if (index2 > index1)
37. return requestString.substring(index1 + 1, index2);
38. }
39. return null;
40. }
42. public String getUri() {
43. return uri;
44. }
46. }

首先调用InputStream对象中的**read方法获取HTTP请求的原始数据，**然后在**parseUri方法中获得uri也就是要请求的静态资源。**说白了**Request类的主要作用就是告诉服务器用户要的是什么**也就是在**http://localhost:8080后面出现的东西。(也就是那些URI，比如userid，productid等)**

**[java]** [view plaincopyprint?](http://blog.csdn.net/beijiguangyong/article/details/7439352)

1. import java.io.OutputStream;
2. import java.io.IOException;
3. import java.io.FileInputStream;
4. import java.io.File;
6. public class Response {
8. private static final int BUFFER\_SIZE = 1024;
9. Request request;
10. OutputStream output;
12. public Response(OutputStream output) {
13. this.output = output;
14. }
16. public void setRequest(Request request) {
17. this.request = request;
18. }
20. public void sendStaticResource() throws IOException {
21. byte[] bytes = new byte[BUFFER\_SIZE];
22. FileInputStream fis = null;
23. try {
24. **File file = new File(HttpServer.WEB\_ROOT, request.getUri());**
25. if (file.exists()) {  //如果file存在，就把里面file的内容给传出去。
26. **fis = new FileInputStream(file);**
27. **int ch = fis.read(bytes, 0, BUFFER\_SIZE);**
28. **while (ch != -1) {**
29. **output.write(bytes, 0, ch);**
30. **ch = fis.read(bytes, 0, BUFFER\_SIZE);**
31. **}**
32. } else {
34. String errorMessage = "HTTP/1.1 404 File Not Found\r\n"
35. + "Content-Type: text/html\r\n"
36. + "Content-Length: 23\r\n" + "\r\n"
37. + "<h1>File Not Found</h1>";
38. output.write(errorMessage.getBytes());
39. }
40. } catch (Exception e) {
42. System.out.println(e.toString());
43. } finally {
44. if (fis != null)
45. fis.close();
46. }
47. }
48. }

**Response类代表一个HTTP响应。首先Response接收一个OutputStream对象，然后通过sendStaticResource方法对接收的Request进行处理，整个处理过程就是根据请求在服务器端进行寻找对应静态资源的过程。找到所需要的资源后发送给客户端然后让客户端显示出来。**

**运行程序**

运行上面的HttpServer类，然后在浏览器的地址栏中键入下面的地址：http:localhost:8080/index.jsp，然后你会在浏览器中看到index.jsp页面。



在控制台可以看到类似于下面的HTTP请求

**[plain]** [view plaincopyprint?](http://blog.csdn.net/beijiguangyong/article/details/7439352)

1. GET /index.jsp HTTP/1.1
2. Host: localhost:8080
3. Connection: keep-alive
4. Cache-Control: max-age=0
5. User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.7 (KHTML, like Gecko) Chrome/16.0.912.75 Safari/535.7 360EE
6. Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8
7. Accept-Encoding: gzip,deflate,sdch
8. Accept-Language: zh-CN,zh;q=0.8
9. Accept-Charset: GBK,utf-8;q=0.7,\*;q=0.3

**小结**

上面自己动手写的这个所谓的服务器仅仅有三个类组成，从功能上来说他只能显示一些静态的资源，并不是全部功能。一个优秀的服务器还有很多细节要做，但是出于学习的目的大家现在有这些了解就足够了