1、概述在最初的 http 协议中，没有上传文件方面的功能。 rfc1867 (<http://www.ietf.org/rfc/rfc1867.txt>) 为 http 协议添加了这个功能。客户端的浏览器，如 Microsoft IE, Mozila, Opera 等，按照此规范将用户指定的文件发送到服务器。服务器端的网页程序，如 php, asp, jsp 等，可以按照此规范，解析出用户发送来的文件。Microsoft IE, Mozila, Opera 已经支持此协议，在网页中使用一个特殊的 form 就可以发送文件。绝大部分 http server ，包括 tomcat ，已经支持此协议，可接受发送来的文件。各种网页程序，如 php, asp, jsp 中，对于上传文件已经做了很好的封装。  
  
2、上传文件的实例：用 servelet 实现（http server 为 tomcat 4.1.24）1. 在一个 html 网页中，写一个如下的form ：<form enctype="multipart/form-data" action="http://192.168.29.65/UploadFile" method=post>   
load multi files :<br>      
<input name="userfile1" type="file"><br>     
<input name="userfile2" type="file"><br>   
<input name="userfile3" type="file"><br>     <input name="userfile4" type="file"><br>     
text field :<input type="text" name="text" value="text"><br>     
<input type="submit" value="提交"><input type=reset></form>  
  
  
  
用户可以选择多个文件，填写表单其它项，点击“提交”按钮后就开始上传给 <http://192.168.29.65/upload_file/UploadFile>  
  
这是一个 servelet 程序注意 enctype="multipart/form-data", method=post, type="file" 。根据 rfc1867, 这三个属性是必须的。multipart/form-data 是新增的编码类型，以提高二进制文件的传输效率。具体的解释请参阅 rfc18672. 服务端 servelet 的编写现在第三方的 http upload file 工具库很多。Jarkata 项目本身就提供了fileupload 包<http://jakarta.apache.org/commons/fileupload/> 。  
  
文件上传、表单项处理、效率问题基本上都考虑到了。在 Struts 中就使用了这个包，不过是用 Struts 的方式另行封装了一次。这里我们直接使用 fileupload 包。至于Struts 中的用法，请参阅 Struts 相关文档。这个处理文件上传的 servelet 主要代码如下：  
  
  
public void doPost( HttpServletRequest request, HttpServletResponse response )  
{     
     DiskFileUpload diskFileUpload = new DiskFileUpload();     // 允许文件最大长度  
     diskFileUpload.setSizeMax( 100\*1024\*1024 );     // 设置内存缓冲大小   
     diskFileUpload.setSizeThreshold( 4096 );     // 设置临时目录     
     diskFileUpload.setRepositoryPath( "c:/tmp" );   
     List fileItems = diskFileUpload.parseRequest( request );   
     Iterator iter = fileItems.iterator();     for( ; iter.hasNext(); )  
     {      
     FileItem fileItem = (FileItem) iter.next();     
       if( fileItem.isFormField() ) {          // 当前是一个表单项      
     out.println( "form field : " + fileItem.getFieldName() + ", " + fileItem.getString() );      
       } else {         
     // 当前是一个上传的文件         
     String fileName = fileItem.getName();   
     fileItem.write( new File("c:/uploads/"+fileName) );       
       }  
  
}}  
  
为简略起见，异常处理，文件重命名等细节没有写出。  
  
3、 客户端发送内容构造假设接受文件的网页程序位于 <http://192.168.29.65/upload_file/UploadFile.>假设我们要发送一个二进制文件、一个文本框表单项、一个密码框表单项。文件名为 E:\s ，其内容如下：（其中的XXX代表二进制数据，如 01 02 03）abbXXXccc 客户端应该向 192.168.29.65 发送如下内容：  
  
  
POST /upload\_file/UploadFile HTTP/1.1  
Accept: text/plain, \*/\*  
Accept-Language: zh-cn  
Host: 192.168.29.65:80  
Content-Type:multipart/form-data;boundary=---------------------------7d33a816d302b6  
User-Agent: Mozilla/4.0 (compatible; OpenOffice.org)  
Content-Length: 424  
Connection: Keep-Alive -----------------------------7d33a816d302b6  
Content-Disposition:form-data;   
name="userfile1";   
filename="E:\s"Content-Type:  
application/octet-stream abbXXXccc  
-----------------------------7d33a816d302b6  
  
Content-Disposition: form-data;  
  
name="text1" foo  
  
-----------------------------7d33a816d302b6  
  
Content-Disposition: form-data;  
  
name="password1" bar  
  
-----------------------------7d33a816d302b6--  
  
  
(上面有一个回车)此内容必须一字不差，包括最后的回车。  
  
注意：Content-Length: 424 这里的424是红色内容的总长度（包括最后的回车）  
注意这一行：Content-Type: multipart/form-data; boundary=---------------------------7d33a816d302b6  
  
根据 rfc1867, multipart/form-data是必须的.---------------------------7d33a816d302b6 是分隔符，分隔多个文件、表单项。  
  
其中33a816d302b6 是即时生成的一个数字，用以确保整个分隔符不会在文件或表单项的内容中出现。前面的 ---------------------------7d 是 IE 特有的标志。  
  
Mozila 为---------------------------71用手工发送这个例子，在上述的 servlet 中检验通过。  
  
  
  
使用POST发送数据  
  
　　以POST方式发送数据主要是为了向服务器发送较大量的客户端的数据，它不受URL的长度限制。POST请求将数据以URL编码的形式放在HTTP正文中，字段形式为fieldname=value，用&分隔每个字段。注意所有的字段都被作为字符串处理。实际上我们要做的就是模拟浏览器POST一个表单。以下是IE发送一个登陆表单的POST请求：  
  
  
POST <http://127.0.0.1/login.do> HTTP/1.0  
Accept: image/gif, image/jpeg, image/pjpeg, \*/\*  
Accept-Language: en-us,zh-cn;q=0.5  
Content-Type: application/x-www-form-urlencoded  
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)  
Content-Length: 28  
\r\n  
username=admin&password=1234  
  
  
　　要在MIDP应用程序中模拟浏览器发送这个POST请求，首先设置HttpConnection的请求方式为POST：  
  
  
hc.setRequestMethod(HttpConnection.POST);  
  
  
　　然后构造出HTTP正文：  
  
  
byte[] data = "username=admin&password=1234".getBytes();  
  
　　并计算正文长度，填入Content-Type和Content-Length：  
  
hc.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");  
hc.setRequestProperty("Content-Length", String.valueOf(data.length));  
  
　　然后打开OutputStream将正文写入：  
  
OutputStream output = hc.openOutputStream();  
output.write(data);  
  
  
　　需要注意的是，数据仍需要以URL编码格式编码，由于MIDP库中没有J2SE中与之对应的URLEncoder类，因此，需要自己动手编写这个encode()方法，可以参考java.net.URLEncoder.java的源码。剩下的便是读取服务器响应，代码与GET一致，这里就不再详述。  
  
　　使用multipart/form-data发送文件  
  
　　如果要在MIDP客户端向服务器上传文件，我们就必须模拟一个POST multipart/form-data类型的请求，Content-Type必须是multipart/form-data。  
  
　　以multipart/form-data编码的POST请求格式与application/x-www-form-urlencoded完全不同，multipart/form-data需要首先在HTTP请求头设置一个分隔符，例如ABCD：  
  
  
hc.setRequestProperty("Content-Type", "multipart/form-data; boundary=ABCD");  
  
　　然后，将每个字段用“--分隔符”分隔，最后一个“--分隔符--”表示结束。例如，要上传一个title字段"Today"和一个文件C:\1.txt，HTTP正文如下：  
  
  
--ABCD  
Content-Disposition: form-data; name="title"  
\r\n  
Today  
--ABCD  
Content-Disposition: form-data; name="1.txt"; filename="C:\1.txt"  
Content-Type: text/plain  
\r\n  
<这里是1.txt文件的内容>  
--ABCD--  
\r\n  
  
  
　　请注意，每一行都必须以\r\n结束，包括最后一行。如果用Sniffer程序检测IE发送的POST请求，可以发现IE的分隔符类似于---------------------------7d4a6d158c9，这是IE产生的一个随机数，目的是防止上传文件中出现分隔符导致服务器无法正确识别文件起始位置。我们可以写一个固定的分隔符，只要足够复杂即可。  
  
　　发送文件的POST代码如下：  
  
  
String[] props = ... // 字段名  
String[] values = ... // 字段值  
byte[] file = ... // 文件内容  
String BOUNDARY = "---------------------------7d4a6d158c9"; // 分隔符  
StringBuffer sb = new StringBuffer();  
// 发送每个字段:  
for(int i=0; i  
sb = sb.append("--");  
sb = sb.append(BOUNDARY);  
sb = sb.append("\r\n");  
sb = sb.append("Content-Disposition: form-data; name=\""+ props*+ "\"\r\n\r\n");  
sb = sb.append(URLEncoder.encode(values));  
sb = sb.append("\r\n");  
}  
// 发送文件:  
sb = sb.append("--");  
sb = sb.append(BOUNDARY);  
sb = sb.append("\r\n");  
sb = sb.append("Content-Disposition: form-data; name=\"1\"; filename=\"1.txt\"\r\n");  
sb = sb.append("Content-Type: application/octet-stream\r\n\r\n");  
byte[] data = sb.toString().getBytes();  
byte[] end\_data = ("\r\n--" + BOUNDARY + "--\r\n").getBytes();  
// 设置HTTP头:  
hc.setRequestProperty("Content-Type", MULTIPART\_FORM\_DATA + "; boundary=" + BOUNDARY);  
hc.setRequestProperty("Content-Length", String.valueOf(data.length + file.length + end\_data.length));  
// 输出:  
output = hc.openOutputStream();  
output.write(data);  
output.write(file);  
output.write(end\_data);  
// 读取服务器响应：  
// TODO...*