# apache ftpserver文档

## 1、使用apache ftpserver

<http://mina.apache.org/ftpserver-project/downloads.html>中下载

启动命令

切换到bin目录后ftpd.bat res/conf/ftpd-typical.xml

## 2、使用ftplet进行业务扩展

文档：<http://mina.apache.org/ftpserver-project/ftplet.html#ftplet>

conf/ftpd-full.xml是配置说明的文档，我们启动的时候使用ftpd-typical.xml。

可以看到里面

<!--

Use this section to define your Ftplets, they are configured like

regular Spring beans

-->

<ftplets>

<ftplet name="ftplet1">

<beans:bean class="org.apache.ftpserver.examples.MyFtplet">

<beans:property name="foo" value="123" />

</beans:bean>

</ftplet>

</ftplets>

我们把它复制放进conf/typical.xml中，修改为这样

<ftplets>

<ftplet name="ftplet1">

<beans:bean class="com.liuyc.test.MyFtplet">

<!--<beans:property name="foo" value="123" />-->

</beans:bean>

</ftplet>

</ftplets>

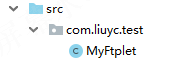
class="com.liuyc.test.MyFtplet"是jar包名，现在需要生成这个jar包并放在common/lib中。

参考文档：

<https://blog.csdn.net/juy19901128/article/details/68955810>

<https://blog.csdn.net/bigbigtangcsdn/article/details/81989204>

1、新建一个project，然后在src上新建package和class

名字和配置对应就可以了。

2、编写类，示例：

**package** com.liuyc.test;  
**import** org.apache.ftpserver.ftplet.\*;   
**import** java.io.\*;   
  
**public class** MyFtplet **extends** DefaultFtplet{  
  
 @Override  
 **public** FtpletResult onConnect(FtpSession session)  
 **throws** FtpException, IOException {  
 System.***out***.println(**"liuyc log: onConnect"**);  
 **return super**.onConnect(session);  
 }  
  
 @Override  
 **public** FtpletResult onDisconnect(FtpSession session)  
 **throws** FtpException, IOException {  
 System.***out***.println(**"liuyc log: onDisconnect"**);  
 **return super**.onDisconnect(session);  
 }  
  
 @Override  
 **public** FtpletResult beforeCommand(FtpSession session, FtpRequest request)  
 **throws** FtpException, IOException {  
 System.***out***.println(**"liuyc log: beforeCommand"**);  
 **return super**.beforeCommand(session, request);  
 }  
  
 @Override  
 **public** FtpletResult afterCommand(FtpSession session, FtpRequest request, FtpReply reply)  
 **throws** FtpException, IOException {  
 System.***out***.println(**"liuyc log: afterCommand"**);  
 **return super**.afterCommand(session, request, reply);  
 }  
}

需要继承DefaultFtplet，然后重写方法实现自定义的内容，可以重写的方法在[官方文档](http://mina.apache.org/ftpserver-project/ftplet.html#ftplet)中都有展示。然后就按参考文档中的两种方法之一进行打包。把生成的jar拷贝到apache-ftpserver-1.1.1/common/lib目录中，重新运行ftpserver。然后通过浏览器或者cmd命令连接ftpserver，会发现执行了自定义的内容。

命令行执行示例：open ip port，输入用户名，密码，显示成功就可以执行所有的ftp命令了。



## 3、监听器说明

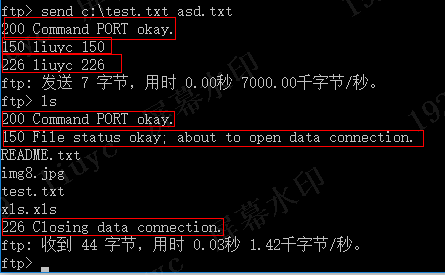
**onUploadStart**：

This method will be called before the file upload. The file name can be get from the request argument. We can get the data input stream from request. This will be called before the permission check. This is called during STOR command. If the method returns SKIP, it has to send responses before and after processing. For example, before opening the data input stream, the method has to notify the client with a response code 150. Similarly, after the data transfer, the method has to notify the client with a response code 226. In case of any error, the method should send different response codes like 450, 425, 426, 551.

通过下面的方式来获取文件的数据，然后进行自定义的处理，可以把获取的数据保存为文件。但我不建议这样做，下面是通过打开连接来获取数据，然后在执行STOR的时候再次返回150状态码来告诉客户端准备读取数据，但已经无法读取了Can't open data connection.，已经被自定义的代码中进行过处理，就会无法保存文件，所以要是自己编写与客户端的直接操作，就把文件保存也实现，把完整的上传过程实现出来，然后**return** FtpletResult.***SKIP***;（个人理解，可能有其他的解决方法）。按说明，这个函数是上传之前进行的处理（start-stor-end），但现在这样获取数据就已经破坏了工作顺序了，变为在start中执行stor的内容。

session.write(**new** DefaultFtpReply(150, **"liuyc 150"**));  
 OutputStream outputStream = **new** ByteArrayOutputStream();  
 DataConnectionFactory connectionFactory = session.getDataConnection();  
 **try** {  
 DataConnection dataConnection = connectionFactory.openConnection();  
 dataConnection.transferFromClient(session, outputStream);  
 *// now outputstream contains the uploaded file and you could  
 // store it in S3 if you wish* ByteArrayOutputStream baos = **new** ByteArrayOutputStream();  
 baos = (ByteArrayOutputStream) outputStream;  
 **byte** b [] = baos.toByteArray();  
 **for**(**int** x= 0 ; x < b.**length**; x++) {  
 *// 打印字符* System.***out***.print((**char**)b[x]);  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 } **finally** {  
 connectionFactory.closeDataConnection();  
 session.write(**new** DefaultFtpReply(226, **"liuyc 226"**));  
 }

可以看到下图，我通过上面的代码自定义返回消息，来告诉客户端，服务端处理成功了，但服务端内部其实是只执行了打印，然后就返回SKIP，所以没有保存文件，在ls的时候也会发现没有asd.txt文件。



而服务端打印以下内容，说明在onUploadStart中收到文件数据，但返回了SKIP，所以没有执行STOR和onUploadEnd的内容了。



**onUploadEnd**：

FtpFile t = session.getFileSystemView().getFile(**request.**getArgument());

//根据文件名可以获得ftp存储目录下的文件

File f = (File)t.getPhysicalFile();

//这样就获得文件对象，可以进行各种操作，如写入读取移动等

由于在onUploadStart中就返回SKIP，如果有某些处理写在onUploadEnd的也要把代码提前到start中，即start中包含start-stor-end的所有工作才能返回skip。

**参数说明：**

在文档<http://mina.apache.org/ftpserver-project/ftplet.html#defaultftplet>中，所有方法都是有FtpSession session, FtpRequest request参数，

session常用的如下：

session.getConnectionTime() 获取用户连接的时间

session.getFileSystemView() 获取当前服务器的文件结构，返回类型为FtpFile

session.getFileSystemView().getHomeDirectory() 获取服务器根目录路径，返回类型为FtpFile

session.getFileSystemView().getWorkingDirectory() 获取用户当前连接的目录路径，返回类型为FtpFile

session.getFileSystemView().getWorkingDirectory().getName() 获取用户当前连接的目录名字，返回类型为String

session.getFileSystemView().getWorkingDirectory().getAbsolutePath() 获取用户当前连接的目录的完整路径名字，返回类型为String（返回之后最后是没有斜杠的）。用这个可以获取用户文件的存放路径。

session.getFileSystemView().changeWorkingDirectory(**"**folder**"**); 切换路径，相对workingDirectory路径，试过../../但无法超出根路径，最终还是在根路径/里

session.getLoginTime() 获取用户登录时间

session.write() 发送消息

另外还可以获取以下属性或方法：

getClientAddress

getServerAddress

getDataConnection

getClientCertificates

getFailedLogins

getLastAccessTime

getMaxIdleTime

setMaxIdleTime

getUser

getUserArgument

getLanguage

isLoggedIn

getFileOffset

getRenameFrom

getDataType

getStructure

getAttribute

setAttribute

removeAttribute

isSecure

getSessionId

-------------------------------------------------------------------------

request常用的如下：

request.getArgument() 获取用户提交的命令中的参数，比如存储命令，这个参数就是文件名

request.getCommand() 获取用户提交的命令中的命令。比如存储的命令是"STOR"

request.getRequestLine() 获取用户提交的命令中的命令+参数

还有：

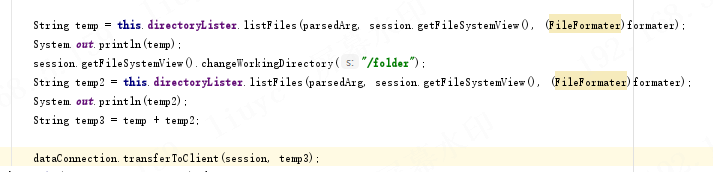
hasArgument

getReceivedTime

## 4、多路径

网上关于ftp有虚拟目录（别名）这个说法，大概就是D:/f1作为主目录，里面新建一个文件夹如alias，F:/f2作为虚拟目录（别名），然后设置f2的符号链接到f1的alias，在访问ftp的时候打开alias会看到f2的内容。这个是网上对ftp配置多路径的方法，但始终是一一对应的关系，ftp网页上看到的列表必须在主目录中存在，不能设置另一个目录的内容也同时显示在主目录，但显示在子目录就可以，即作为虚拟目录。

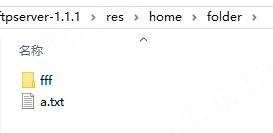
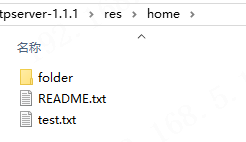
如果需要把两个甚至多个目录的文件都显示在ls命令下，则需要对多个命令进行修改。系统没有关于ls命令的start和end的监听处理器，所以我们能进行处理的只能在beforeCommand和afterCommand，而大多命令都在beforeCommand和afterCommand之间进行了dataConnection的数据传递了，所以要修改只能在beforeCommand中进行修改。



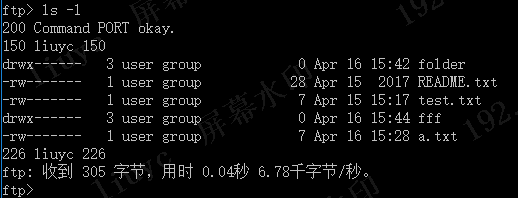
打开dataConnection，获取完主目录的数据，然后切换目录（只能基于主目录进行切换）再获取，然后作为temp3一起返回，最后要返回SKIP，否则在NLST命令中再次打开dataConnection又报Can't open data connection。最后客户端就会收到：



而ftp的实际文件系统情况

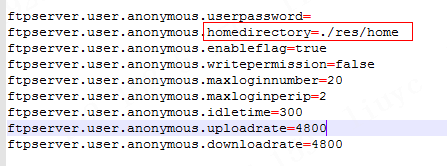


这样对于客户端来说是同一个文件夹，但实际在服务端是不同的文件夹都显示出来，但现在未能把folder也隐藏起来，因为它里面的内容已经显示了出来，所以folder这个文件夹是不应该显示的。通过代码可以看到数据是字符串格式，通过过滤应该是可以去掉的，但还有一种命令ls -l，比较难处理，需要一种不管什么参数，都能进行处理的方法。同时也存在一个问题，如果两个不同的文件夹有相同的文件名文件，在显示的时候就会相同了，下载的时候服务器是到哪个目录下进行下载也无法判断。



而且folder既然是想隐藏的，那么ls folder不应该返回folder文件夹下的信息，所以参数越多的命令需要修改的内容就越多。浏览器访问触发的是LIST –l的命令，也需要把LIST进行同样的处理，dir也一样。

以上是通过代码来进行处理的，通过配置没有这种功能，没找到可以配置多个路径的方法。



## 5、下载

**问题描述：**假如下载test.txt（1K），系统对文件进行处理，假如处理后文件变为apache-tomcat-8.5.37.exe（10M，为了测试断点续传），下载过程中断开连接，再次下载，那么是否可以不再把txt处理为exe而直接把exe发送过来，并且是知道偏移（跳过长度）的发送？

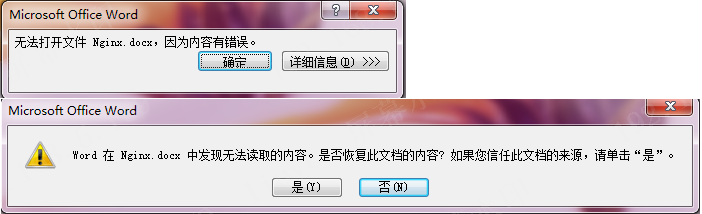
**1、第一个问题**，解决方法判断文件是否存在。例如在其他盘，专门存放处理后的文件（otherhome目录），假如目录结构和文件名都与ftp目录（home目录）相同，然后下载的时候，判断otherhome/file是否存在，因为路径是可以确定的，所以可以这样判断文件存在，不存在就重新处理，存在就获取它的数据。

cmd过程：下载已存在的Nginx.docx（源11K，处理后60K）和不存在的test.txt，通过下面两图展示过程：

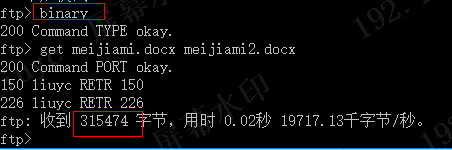


（这图下载进度是错误的，下面第6大点会进行说明）

关于cmd下载后的文件报错误：

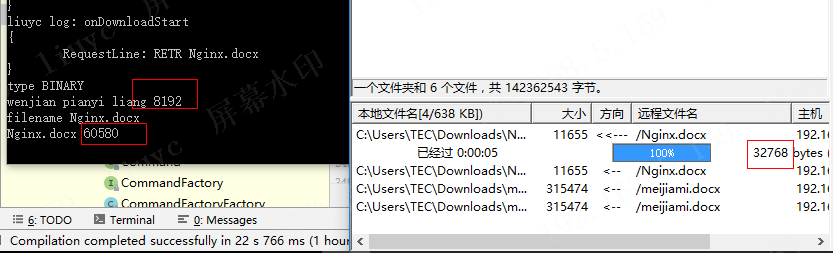


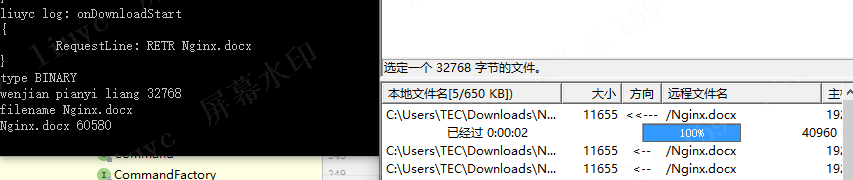
通过浏览器下载的能打开，通过cmd下载的报上图，点击是也无法打开，再显示有错误。通过查看文件大小，发现在cmd端接收到的数据大了，原因是使用了ascii模式传输，换行符改变了，解决方法：



**2、第二个问题**，文件跳过长度，通过session.getFileOffset()来获取，这个不需要进行修改。

下面两图，可以看出，图一下载到32768的时候断开连接，然后续传下载，在图二中能获取到35768这个值，从这个大小开始进行接收数据合并到文件中，两个图片的经过时间可以看出是两次下载请求。





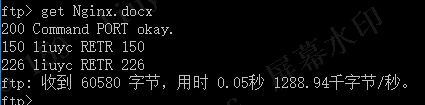
## 6、关于下载时文件大小的处理：

### 6.1、浏览器端解决，ftp工具端未解决

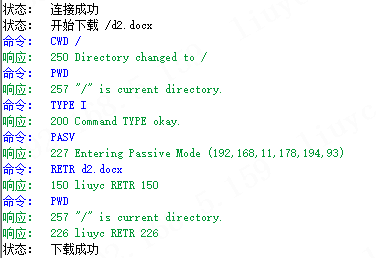
1、先说明在浏览器端，因为是能成功处理的，通过下图看到，浏览器会发送一个size命令，获得文件的大小，所以我在beforeCommand中对SIZE命令进行修改，把文件改为处理后的文件，这样返回给客户端的文件大小就是加密后的文件大小，下载就能显示正常的进度。



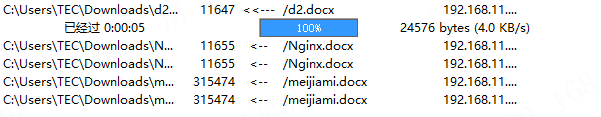
2、对于命令行端，不知道怎么监测进度，所以看不出有什么问题。

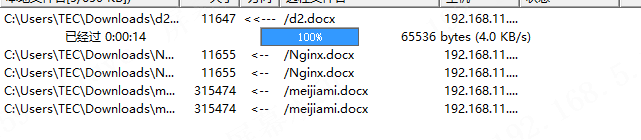


3、对于ftp工具，就有问题了。下图，它并没有发送size命令



下载过程如下图，达到了100%，但还在继续接收数据（源文件11K，处理后文件105K）





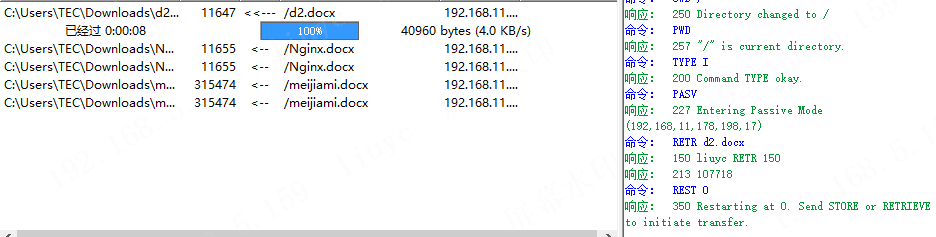
### 6.2、个人对返回码213（文件状态回复）的理解：

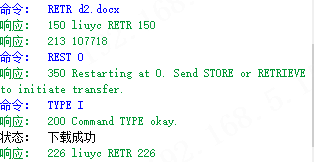
上面6.1是修改SIZE命令来达到效果，下面不对SIZE命令进行修改，通过服务端返回213告诉客户端文件大小的方式进行实验。

1、如果在onDownloadStart返回150之前返回213，浏览器无法显示下载进度，浏览器看到网页在等待，一段时候之后，就是文件完全下载完成的时候，直接显示100%然后就下载完成。

同样在ftp工具上测试，，会捕捉到异常426 Socket exception during data transferSocket closed，无法正常下载，本地生成一个0K的文件。

2、如果213在150与226之间，打开dataConnect之前进行返回，浏览器效果与上面第1点相同，而ftp工具能进行下载了，但仍是使用源文件的大小来作为100%时的值



而它收到一个350（文件行为暂停）不清楚是什么原因。

3、如果213在150与226之间，打开dataConnect之后，进行返回，ftp工具同第2点，浏览器没有了等待的过程，直接显示下载中，但获取的是源文件大小11K作为分母值，而不是我返回213的文件大小作为分母值。下图显示超出100%时的下载过程



4、如果213在226之后进行返回，ftp工具同第2点，浏览器同第3点。

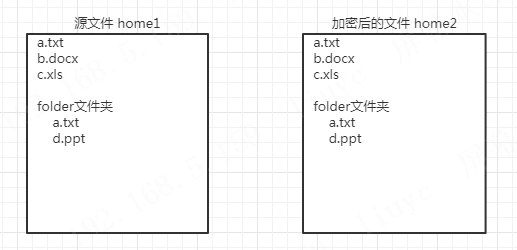
### 6.3总结

通过以上，就是说明服务端单独的返回213，没有响应到客户端的什么内容，需要客户端发送size命令，服务器返回213返回码才是正常的响应过程。

在chrome浏览器，只要修改SIZE命令，返回处理后文件的大小就能解决。而对于ftp工具（8uftp），它并没有发送size命令，它怎么获取文件大小来作为分母值我也不知道，可能是在LIST的时候就获取了保存在数组中，甚至是软件通过字符串读取来获取，如果是这样的话，那就没有办法显示正确的进度百分比，需要知道多种ftp工具是以什么方法来获取文件大小作为分母值来计算进度百分比才能继续进行修改。

## 7、文件维护方案

由于在第5大点中使用两个目录结构相同的方式来达到下载文件的时候，去寻找另一目录下的文件来发送给客户端，示意图



假如原本有3个文件和一个目录folder，里面还有2个文件

另一个专门存放加密后的文件的目录保持同样的结构和相同的文件名

在下载的时候，如get folder/d.ppt

程序就到path/home2/中拼接上参数folder/d.ppt，就能找到对应的加密文档进行下载

上传文件，在onUploadEnd中把上传到home1的文件进行加密并保存到home2中

删除文件，在onDeleteStart中对home2的文件也进行删除

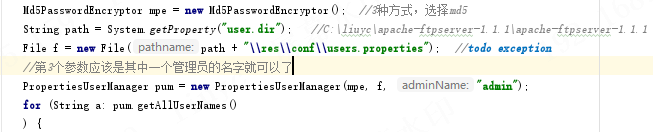
创建文件夹，在onMkdirStart中给home2也同样创建一个文件夹

等等，这样是通过命令来维护文件，但如果文件长期存在，就相当于保存两份文档会比较占空间。于是也需要一个定时器来进行管理。

写一个定时器，每隔一定长度的时间运行一次，大概的处理内容就是获取所有用户，通过用户可以获取他们的home1，然后把home1通过某些特定的规则转为home2，先遍历用户，判断他们是否在使用ftp服务，如果没在使用，就查看用户目录下的文件修改日期超过某个固定时间的就删除。

问题1：怎么获取所有用户？

解决：用户管理有两种方式，一种保存在user.properties文件中，另一种是使用数据库进行管理，需要配置（未实践过不清楚是什么效果）。这里我使用文件的方式，下图是获取所有用户的代码：



问题2：怎么判断用户是否在使用ftp服务？

解决：使用一个HashMap全局变量，记录key-value，key为用户名，value为正在进行操作的时候加1，当为0或null的时候才对该用户进行清理文件工作。如两个终端A和B都是同一用户anonymous，A进行下载操作，这个时候map(anonymous,1)，B也开始下载操作，这时map(anonymous,2)，到了准备清理文件的时间，判断用户admin，value为null，所以开始执行清理，然后下一个用户anonymous，value为2，不进行清理，然后A下载完毕，map(anonymous,1)，还是不能进行清理，B完成任务，value变为0，可以进行清理，等下一个清理周期。

这样也会存在一个情况，可能用户处于空闲状态，而刚好到了定时器的执行时间，就会有可能在删除文件的时候，用户准备去下载文件，所以还需要一个标志，如果正在执行清理操作，就返回用户稍后再使用之类的信息，最后把清理设置在凌晨，就是通过用户使用标志和清理任务标志加上分析最少人使用的时间段，基本上就能避免使用冲突。

如果用户的确在某个时间，下载一个大文件需要比较长的时间，而这个时间也是需要清理文件的时间，则清理任务知道用户在使用，那先清除其他没在使用的用户的文件，该用户等下一次再进行清理。最坏的情况，该用户就是每天都在准备清理的时候进行操作，就会导致该用户一直都没法清理文件。这个时候我们可以增加一个全局变量，标记用户在处理哪些文件，在清理任务中忽略这些文件，对其他如超过3天的文件进行删除。

存在的问题是浏览器对返回码固定了处理，例如在下载时返回某个返回码想告诉用户稍后再试，浏览器只会显示：无法访问此网站 网址为 ftp://192.168.11.178:2121/d2.docx 的网页可能暂时无法连接，或者它已永久性地移动到了新网址。而ftp工具（8uftp）能显示自定义返回的消息，这样用户就能知道发生了什么情况。