**ftplet**：用来监听命令，监听命令前后的动作，我们可以在这些监听方法中实现自己的逻辑。我们可以叫它为监听器、插件等。

## 1、在ftplet中，假如我们没有写过任何内容，只有普通的输出到console的日志，这个时候，测试上传：

（1）上传直至完成

beforeCommand：request：STOR filename

onUploadStart：request：STOR filename

执行STOR默认的操作

//afterCommand特殊，多一个reply变量

afterCommand：request：STOR filename，reply：226 Transfer complete

onUploadEnd：STOR filename

（2）上传-中止-继续-中止

beforeCommand：request：STOR filename

onUploadStart：request：STOR filename

执行STOR默认的操作

afterCommand：request：STOR filename，reply：226 Transfer complete

onUploadEnd：STOR filename

beforeCommand：ABOR

执行ABOR默认的操作

afterCommand：ABOR 226 ABOR successful

beforeCommand：APPE filename

onAppendStart：APPE filename

执行APPE默认的操作

afterCommand：APPE filename，reply：226 Transfer complete

onAppendEnd：APPE filename

beforeCommand：ABOR

执行ABOR默认的操作

afterCommand：ABOR 226 ABOR successful

上面的两种情况，说明完整上传和中止上传时会产生一样的结果，所以服务器是不知道如何区分两者

<https://stackoverflow.com/questions/17948452/apache-mina-ftplet-move-uploaded-file-after-succesfull-upload>，有人提出了同样的问题，但没有答案

## 2、能否通过ftplet增加逻辑来实现区分完整上传和中止上传

我们先看一下STOR的源码：

**try** {  
 **long** skipLen = session.getFileOffset();  
 String fileName = request.getArgument();  
 **if** (fileName == **null**) {  
 session.write(LocalizedDataTransferFtpReply.translate(session, request, context, 501, **"STOR"**, (String)**null**, (FtpFile)**null**));  
 } **else** {  
 DataConnectionFactory connFactory = session.getDataConnection();  
 **if** (connFactory **instanceof** IODataConnectionFactory) {  
 InetAddress address = ((IODataConnectionFactory)connFactory).getInetAddress();  
 **if** (address == **null**) {  
 session.write(**new** DefaultFtpReply(503, **"PORT or PASV must be issued first"**));  
 **return**;  
 }  
 }  
  
 FtpFile file = **null**;  
  
 **try** {  
 file = session.getFileSystemView().getFile(fileName);  
 } **catch** (Exception var32) {  
 **this**.LOG.debug(**"Exception getting file object"**, var32);  
 }  
  
 **if** (file == **null**) {  
 session.write(LocalizedDataTransferFtpReply.translate(session, request, context, 550, **"STOR.invalid"**, fileName, file));  
 } **else** {  
 fileName = file.getAbsolutePath();  
 **if** (!file.isWritable()) {  
 session.write(LocalizedDataTransferFtpReply.translate(session, request, context, 550, **"STOR.permission"**, fileName, file));  
 } **else** {  
 session.write(LocalizedFtpReply.translate(session, request, context, 150, **"STOR"**, fileName)).awaitUninterruptibly(10000L);  
  
 DataConnection dataConnection;  
 **try** {  
 dataConnection = session.getDataConnection().openConnection();  
 } **catch** (Exception var33) {  
 **this**.LOG.debug(**"Exception getting the input data stream"**, var33);  
 session.write(LocalizedDataTransferFtpReply.translate(session, request, context, 425, **"STOR"**, fileName, file));  
 **return**;  
 }  
  
 **boolean** failure = **false**;  
 OutputStream outStream = **null**;  
 **long** transSz = 0L;  
  
 **try** {  
 outStream = file.createOutputStream(skipLen);  
 transSz = dataConnection.transferFromClient(session.getFtpletSession(), outStream);  
 **if** (outStream != **null**) {  
 outStream.close();  
 }  
  
 **this**.LOG.info(**"File uploaded {}"**, fileName);  
 ServerFtpStatistics ftpStat = (ServerFtpStatistics)context.getFtpStatistics();  
 ftpStat.setUpload(session, file, transSz);  
 } **catch** (SocketException var29) {  
 **this**.LOG.debug(**"Socket exception during data transfer"**, var29);  
 failure = **true**;  
 session.write(LocalizedDataTransferFtpReply.translate(session, request, context, 426, **"STOR"**, fileName, file));  
 } **catch** (IOException var30) {  
 **this**.LOG.debug(**"IOException during data transfer"**, var30);  
 failure = **true**;  
 session.write(LocalizedDataTransferFtpReply.translate(session, request, context, 551, **"STOR"**, fileName, file));  
 } **finally** {  
 IoUtils.close(outStream);  
 }  
  
 **if** (!failure) {  
 session.write(LocalizedDataTransferFtpReply.translate(session, request, context, 226, **"STOR"**, fileName, file, transSz));  
 }  
  
 }  
 }  
 }  
} **finally** {  
 session.resetState();  
 session.getDataConnection().closeDataConnection();  
}

上边标蓝色的就是整个命令的核心，先创建一个file，然后发送150告诉客户端打开连接，然后通过transferFromClient接收数据到file，在接收过程中一直都没有错的话，就发送226告诉客户端上传成功，最后关闭连接。

当时我认为ABOR会触发transferFromClient的socket异常，这样就可以知道是否用户终止了，因为我在测试下载的时候，ABOR的确会触发socket异常然后告诉客户端426的错误，但实际测试上传的时候，却发现始终走的是没有错误的代码即226状态，亦即第一点提到的stackoverflow链接 的内容。

STOR命令中始终无法找出完整上传和中止上传的不同点，就尝试在其他监听命令中分析两者是否会存在区别，但仍然无果。

## 3、总结

在没对ftplet进行编写的情况下测试8uftp、FlashFXP、FileZilla在上传然后中止的时候触发的命令

### 3.1、8uftp：

beforeCommand：request：STOR filename

onUploadStart：request：STOR filename

执行STOR默认的操作

afterCommand：request：STOR filename，reply：226 Transfer complete

onUploadEnd：STOR filename

beforeCommand：ABOR

执行ABOR默认的操作

afterCommand：ABOR 226 ABOR successful

==========================================

### 3.2、FlashFXP：

beforeCommand：request：STOR filename

onUploadStart：request：STOR filename

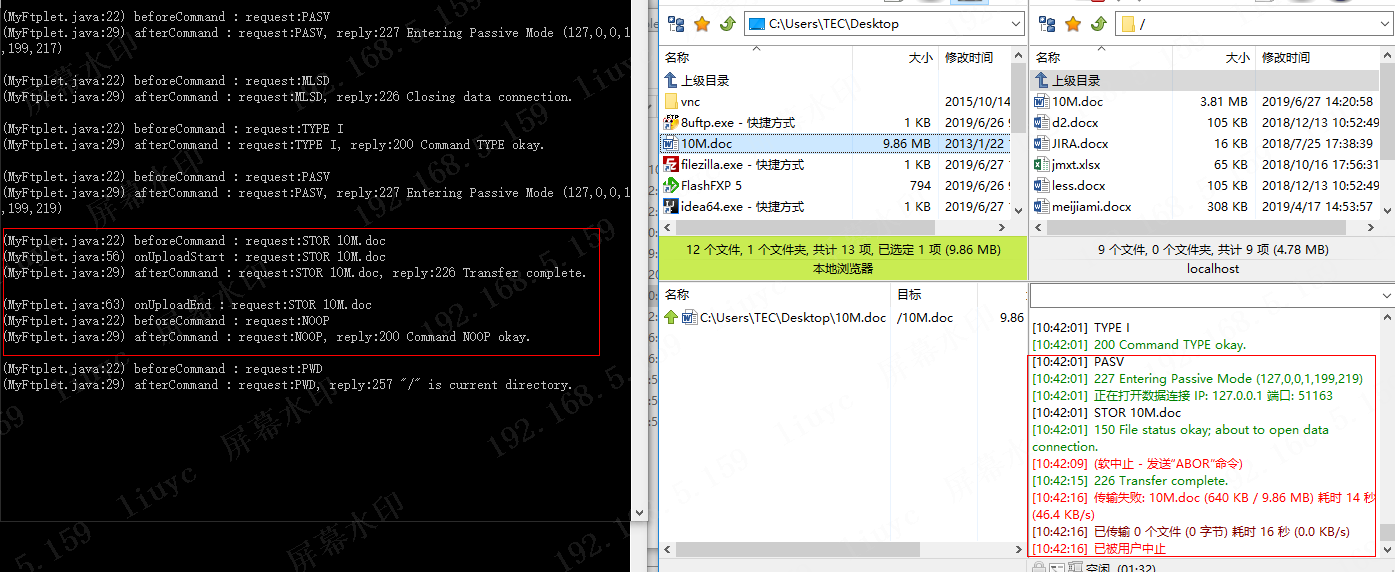
执行STOR默认的操作

afterCommand：request：STOR filename，reply：226 Transfer complete

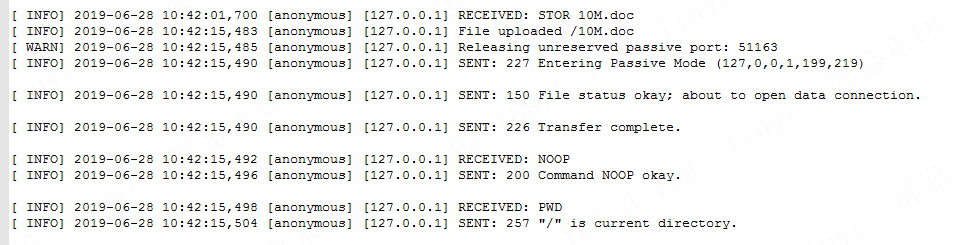
onUploadEnd：request：STOR filename

beforeCommand：request：NOOP

afterCommand：request：NOOP，reply：200 Command NOOP okay



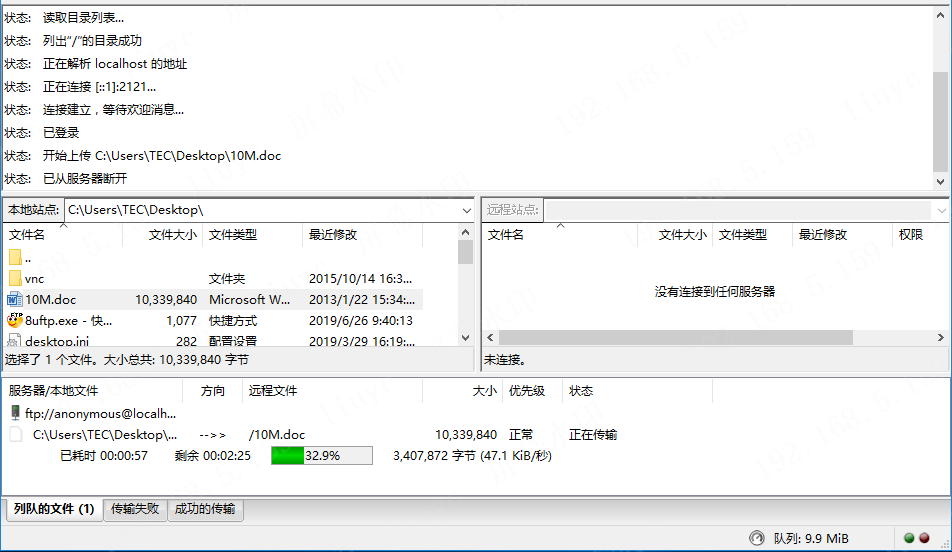
日志：



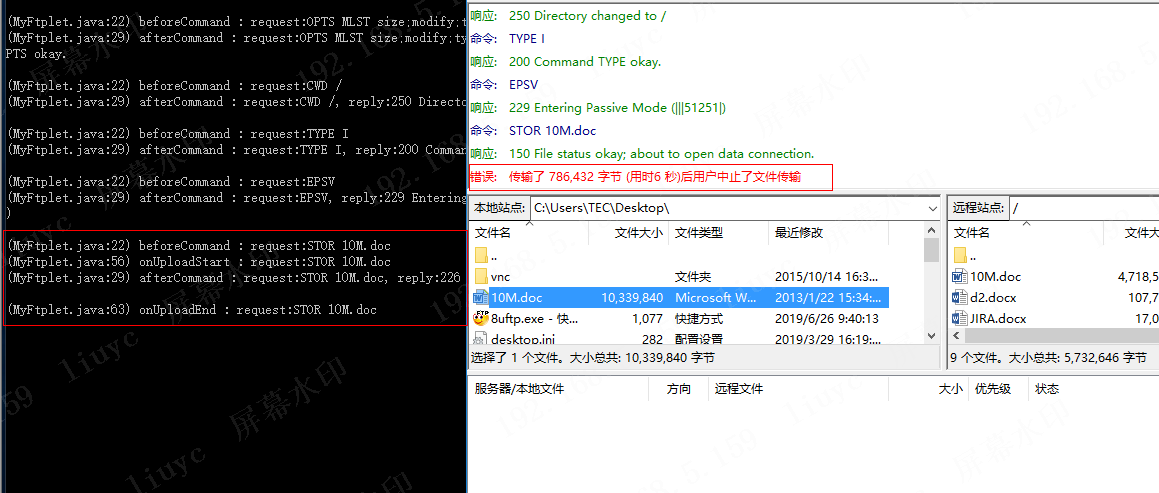
它并没有收到ABOR命令，而是NOOP

================================================

### 3.3、FileZilla：

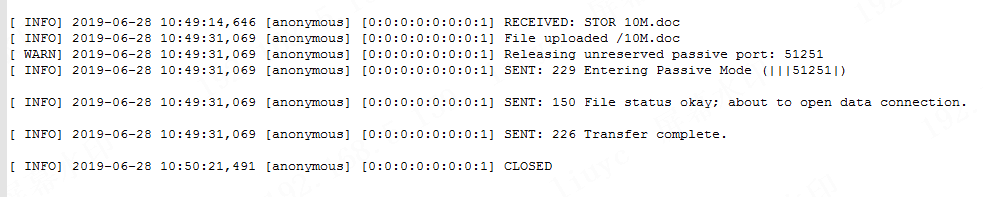


提示：断开与服务器的链接，仍会继续上传，只能删除队列才能中止：



并没有发送ABOR的命令

日志：



===============================================

### 3.4、综上3种客户端的执行结果：

对于FileZilla，因为它并没有发送ABOR命令或者其他的命令（也有可能是我自己操作不当才没有ABOR命令），所以我们在ftpserver永远都不知道是完整上传了还是用户中止上传了，因为它的执行始终是：

beforeCommand：request：STOR filename

onUploadStart：request：STOR filename

执行STOR默认的操作

afterCommand：request：STOR filename，reply：226 Transfer complete

onUploadEnd：STOR filename

就算上面第2大点中提到的ABOR能触发异常来区分完整上传和中止上传的例子成功了，那FileZilla会因为没有发送ABOR导致仍然无法判断。

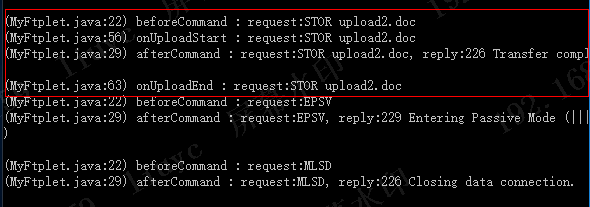
对于FlashFXP，客户端是提示了发送ABOR命令，但在服务端并没有看到beforeCommand ABOR相关的输出，反而有个NOOP的命令，猜测是ABOR命令由于没有触发，就触发了NOOP命令代表“无动作，除了来自服务器上的承认”来回应客户端你的命令处理了但无动作。

对于8uftp：会有ABOR的消息，对于ABOR命令，解析是：如果没有在传输，执行了也没什么更改，会返回执行成功，如果有传输，就会中断数据连接，同样返回执行成功。该消息多次测试都在afterCommand STOR/APPE之后，所以对于8uftp，我们可以在afterCommand STOR/APPE时就在数据库设置该文件已经上传完毕，然后在beforeCommand ABOR的时候，判断两个命令相隔了多长的时间，如果是在短时间内这两个命令相连的，就是用户中止了上传，就在beforeCommand ABOR中把数据库该文件设置会未上传完。主要是这个时间间隔，要短，并且是afterCommand STOR/APPE和beforeCommand ABOR相连，才能代表用户中止了传输，不然只能代表abor没变化。

再简单总结：FileZilla好像是永远都没办法区分完整上传还是中止上传，8uftp通过一种妥协的方法来实现控制，FlashFXP收到的是NOOP，如果模仿8uftp的处理又似乎不太合适。所以如果8uftp的方法也不能作为文件状态的判断，那就没有方法了。然后网上有一个关于客户端如何区分服务端的文件是否完整：<https://stackoverflow.com/questions/37418455/how-to-know-if-the-file-is-completely-uploaded-in-ftp>，里面提到的也是一种妥协的方法，通过判断文件大小是否变化来判断文件是否上传完成，这个方法就不用区分3中客户端的差异。

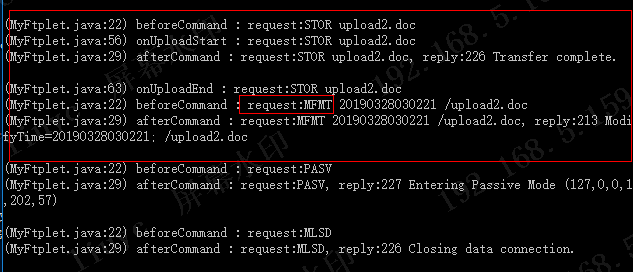
附完整上传时的命令：

FileZilla：



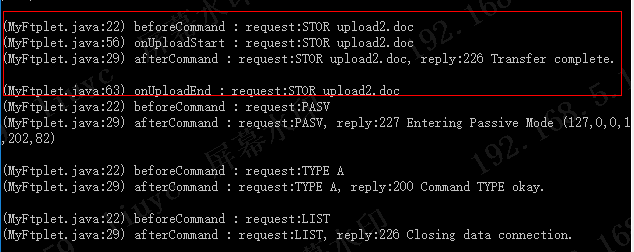
与中止没分别。

FlashFXP：



会比终止多了MFMT命令，可以作为完成标识

8uftp：



与中止上传没有区别

3个客户端，最好就是在中止的时候，服务端能产生异常，不中止就没有异常，这样才能控制是否上传完整。