Brice Prosperi
Olivier Chrétien

Rapport Java : Simple FTP

Dans un premier temps nous avons éditer la classe FTPServerMain afin de permettre le multithreading et ainsi ne pas limiter à 1 le nombre de connexions simultanées au serveur ainsi que pour pouvoir uploader plusieurs fichiers à la fois.

Tirée de FTPServerMain.java :

```
final int numberOfCores = Runtime.getRuntime().availableProcessors();
final double blockingCoefficient = 0.9;
final int poolSize = (int)(numberOfCores / (1 - blockingCoefficient));
final ExecutorService executorPool = Executors.newFixedThreadPool(poolSize);

try {
    socketaccueil = new ServerSocket(PORT);
    System.out.println("Server up and running.");
    FTPServer.setDebug(true);
    while (true) {
        FTPServer server = new FTPServer(socketaccueil.accept());
        // submit pool
        executorPool.submit(server);
```

Nous avons ensuite implémentée les méthodes sendObject() et populate() au sein de la classe FTPServer pour permettre l'envoi d'objet serializé entre le serveur et le client.

Tiré de FTPServer.java :

```
private void sendObject(File file)throws IOException{
    if (socket == null) {
        throw new IOException("FTPServer is not connected.");
    }
    populate(file);
    output.writeObject(file);
}
private void populate(File root) {
    // TODO Auto-generated method stub
    for(File it : root.listFiles()){
        @SuppressWarnings("unused")
        File f = new File(root, it.getName());
        if(it.isDirectory()){
            populate(it);
        }
    }
}
```

Puis nous nous sommes attachés à la modification des méthodes listDir() et populateTree() afin d'obtenir arborescence des fichiers du serveur FTP.

Tirée de CommandDispatcher.java :

```
private void listDir() {
    if (!alreadyConnected) {
        window.console.append(NL + "You are not connected to any server.");
        return;
    }
    try {
        String dirContent = client.ls();
        window.console.append(NL + dirContent);
        File f = client.getFileTree();
        window.root.removeAllChildren();
        DefaultMutableTreeNode userRoot = new DefaultMutableTreeNode(f.getName());
        populateTree(userRoot, f);
        window.root.add(userRoot);
    } catch (IOException e) {
        window.console.append(NL + e.getMessage());
        e.printStackTrace();
    }

}

private void populateTree(DefaultMutableTreeNode userRoot, File root) {
    // TO BE COMPLETED - WRITE THIS RECURSIVE METHOD - 7 LINES
    File[] files = root.listFiles();
    for (File f : files){
        DefaultMutableTreeNode child = new DefaultMutableTreeNode (f.getName());
        userRoot.add(child);
        if(f.isDirectory()){
            populateTree(child, f);
        }
    }
}
```

Pour la suite nous n'avons pas été en mesure de réaliser le CWD. Nous avons toutefois tenté le download en implétant une classe Downloader ainsi que les méthodes stin() et download() mais n'ayant su maitriser le CWD nous n'avons pas non plus été en mesure de modifier les dossiers source et destination pour le download. Voici tout de même les codes des méthodes implémentées.

Tirées de FTPServ.java

```
case STIN:
    stin();
    break;
```

```
private void stin(){

    try {
          dl2 = dl.accept();
          System.out.println("Accepted connection : " + dl2);

          // sendfile
          File file = new File (System.getProperty("user.dir")
                    + System.getProperty("file.separator")
                    + response.substring(5));
          byte [] buffer  = new byte [4096];
          FileInputStream input = new FileInputStream(file);
        BufferedInputStream bufferInput = new BufferedInputStream(input);
          bufferInput.read(buffer,0,buffer.length);
          OutputStream output = dl2.getOutputStream();
          System.out.println("Sending...");
          output.write(buffer,0,buffer.length);
          output.flush();
          output.close();
          input.close();
          dl.close();
          dl2.close();
          System.out.println("STIN " + response.split(" ")[1]
                    + " : File send");
          } catch (IOException e) {
                try {
                    sendLine("error");
                } catch (IOException e1) {
                    e1.printStackTrace();
                }
                e.printStackTrace();
            }
}
```

Tirées de CommandDispatcher.java :

```
case STIN:
    download();
    break;
```

```
private void download() {
    if (!alreadyConnected) {
        window.console.append(NL + "You are not connected to any server.");
        return;
    }
    final JFileChooser fc = new JFileChooser();
    fc.setFileSelectionMode(JFileChooser.FILES_ONLY);
    fc.setMultiSelectionEnabled(true);
    fc.setDragEnabled(true);
    int returnVal = fc.showDialog(window, "Download");
    if (returnVal == JFileChooser.APPROVE_OPTION) {
        File[] files = fc.getSelectedFiles();
        List<Future<Boolean>> downloaders = new ArrayList<Future<Boolean>>(
                files.length);
        for (File f : files) {
            Downloader downloader = new Downloader(f, window, server, port, user,
                    pass);
            downloaders.add(completionPool.submit(downloader));
        }
    } else {
        window.console.append(NL + "Download action cancelled.");
    }
}
```

Tirée de SimpleFTP.java :

```
public boolean stin(File file) throws IOException {
    if (file.isDirectory()) {
        throw new IOException("Client cannot download a directory.");
    }
    String filename = file.getName();
    return stin(new FileInputStream(file), filename);
}

public boolean stin(InputStream inputStream, String filename)
        throws IOException {
    sendLine("PASV");
    String response = readLine();
    if (!response.startsWith("227 ")) {
        throw new IOException("SimpleFTP could not request passive mode: "
    + response);
    }

    byte [] buffer  = new byte [4096];
    InputStream input = socket.getInputStream();
    FileOutputStream output = new FileOutputStream(filename);
    BufferedOutputStream bufferOutput = new BufferedOutputStream(output);
    int bytesRead = input.read(buffer,0,buffer.length);
    int current = bytesRead;
    do {
        bytesRead =
                input.read(buffer, current, (buffer.length-current));
        if(bytesRead >= 0) current += bytesRead;
    } while(bytesRead > -1);
    sendLine("STIN " + filename);
    bufferOutput.write(buffer, 0 , current);
    bufferOutput.flush();
    bufferOutput.close();
    socket.close();
    response = readLine();
    return response.startsWith("226 ");
}
```