3.4. Клиентские и серверные приложения на сетевых соединениях и дейтаграммах

Большинство протоколов прикладного уровня работают на основе соединений TCP/IP или дейтаграмм (UDP). Для разработки клиентских и серверных приложений, взаимодействующих с существующими системами, на Java можно использовать классы обычных и дейтаграммных сокетов. При анализе формата сообщений протокола нужно обращать внимание, на каких потоках данных они строятся – текстовых или двоичных.

Пример. Простой клиент FTP

Протокол FTP [70,71] строится на основе обмена тестовыми (одно и многострочными сообщениями) без соглашений о кодировке, т.е. допускает только использование латиницы в именах файлов и каталогов. Для передачи файла устанавливается отдельное соединение, параметрами которого (IP-адрес и порт) клиент и сервер обмениваются перед передачей файла. Имеется два режима передачи, устанавливаемые по команде клиента:

- В пассивном режиме сервер выделяет порт для установления соединения и сообщает его клиенту, ожидая на нем соединения. Клиент, получив номер порта, устанавливает к нему соединение, после чего оба участника протокола передают/принимают физический поток байтов файла. Окончание передачи обозначается закрытием соединения передающей стороной – приемная сторона получает EOF;
- В активном режиме клиент передает собственный адрес и порт, на котором он будет ожидать соединение, сервер запрашивает соединение, после чего передача данных производится аналогично. Использование активного режима возможно, если клиент имеет статический IP-адрес в сети, таковой эмулируется сетью.

Рассматриваемый простейший клиент не реализует полную систему команд протокола, в частности, не выполняет никаких операций с каталогами. Он выполняет передачу единственного файла в рамках установленного соединения. Тем не менее, он использует несколько потоков, чтобы при рассогласовании процедур обмена не происходило зависаний клиента:

- Текстовый поток – слушатель сообщений сервера. Хотя основное взаимодействие происходит по схеме запрос-ответ, его наличие является принципиальным, т.к. позволяет реагировать на любое асинхронные действия со стороны сервера;
- Передача файла по отдельному соединению также является потоком, в котором происходит установление этого соединения и сама передача;
- По завершении сеанса устанавливается тайм-аут, реализуемый отдельным потоком (следовало бы установить его на все операции);
- Основная последовательность команд клиента также реализована в потоке (уже в **main**).

```
//-----------------------------------------------------------------------------------------------------34_FTPClientPassive
public class FTPClientPassive{
   private boolean connected=false;          // Основное соединение установлено
   private boolean loggedIn=false;           // Авторизация произведена
   private Socket socket;
   private PrintWriter pw;                   // Текстовый поток передачи
   private String localIP="";
   private String remoteIP="";               // Удаленный IP для второго соединения
   private int remotePort=0;                 // Порт для второго соединения
```

```
   private ServerAnswerThread log;              // Поток, принимающий ответы сервера
   private DataThread dataThread;               // Поток, передающий файл
   private String localFolderName="";           // Имена Локального каталога и файла
   private String localFileName="";
   private boolean linux=false;
```

Вложенный класс - поток, «сидящий на приеме», принимает строки текста от сервера из текстового потока, открытого на сокете, который создан основным классом. У него есть собственное состояние **connected** и метод **interrupt**

```
//-------------------------------------------------------------------------------------------------------------34_FTPClientPassive
   public class ServerAnswerThread extends Thread{
      private boolean connected;
      public boolean isConnected(){ return connected; }
      public ServerAnswerThread(){
         connected = true;
         start();
         }
      @Override
      public void interrupt(){ super.interrupt(); connected = false; }
```

Чисто техническая процедура, разбор ответа сервера, содержащего 6 десятичных чисел – 4 числа - IP-адрес и 2 числа – значение байтов номера порта.

```
//-------------------------------------------------------------------------------------------------------------34_FTPClientPassive
      private String []getDigits(String ss){   // Распознавание цепочки чисел - IP-порт
         int k1=ss.indexOf("(");
         int k2=ss.indexOf(")");
         ss=ss.substring(k1+1,k2);
         return ss.split(",");
         }
```

Поток анализирует только 3 вида сообщений сервера – сообщения об ошибках (>400) и окончании сеанса - QUIT 221, 230 – успешной авторизации и 227 – получении IP-порта соединения для передачи файла. Сообщения изменяю данные основного класса синхронизировано со всеми остальными операциями.

```
//-------------------------------------------------------------------------------------------------------------34_FTPClientPassive
      @Override
      public void run(){
         try{
            BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            String message;
            while (connected){
               message = br.readLine();
               if (message != null){
                  log(message);
                  int code = Integer.parseInt(message.substring(0, 3));
                  synchronized (FTPClientPassive.this){
                     if (code == 221 || code >=500) { closeAll(); connected=false; }
                     if (code == 230) loggedIn = true;
                     if (code == 227){
                        String arr[] = getDigits(message.substring(3));
                        remotePort=Integer.parseInt(arr[4])*256+Integer.parseInt(arr[5]);
                        remoteIP=arr[0]+"."+arr[1]+"."+arr[2]+"."+arr[3];
                        }
                     }
                  }
               }
            br.close();
         }  catch(Exception ex){ fatal(ex.getMessage()); }
      }
```

```
}
```

Поток передачи файла имеет также собственный признак активности **isRun**, по которому ждет его завершения основной поток. Поток открывает сокет, файловый поток данных и физический поток данных на сокете, а затем копирует данные блоками по 1024 байта до появления EOF во входном потоке. Для каждого направления передачи реализован отдельный метод.

```
//-----------------------------------------------------------------------------------------------------34_FTPClientPassive
   public class DataThread extends Thread{
   private Socket dataSocket;
   private int code;
   private boolean isRun=false;

   public DataThread(int cmd) throws Exception{
      code=cmd;
      isRun=true;
      start();
      }
   public boolean isRun(){
           synchronized (FTPClientPassive.this){
           return isRun;
           }}
   @Override
   public void interrupt(){
      super.interrupt();
      if (dataSocket.isClosed()) return;
      try {
         dataSocket.close();
          } catch(IOException ex){ fatal(ex.getMessage()); }
      }
   @Override
   public void run(){
      try {
         dataSocket=new Socket();
         dataSocket.connect(new InetSocketAddress(remoteIP,remotePort),15000);
         log("Connected!!!!");
         if (!socket.isConnected()) new IOException("Ошибка соединения с сервером");
         if (code==0) download();
         if (code==1) upload();
         }
      catch(Throwable ex) { fatal(ex.getMessage()); }
      isRun=false;
      }

   private void download()throws  Throwable{
      char cc=(linux ? '/':'\\');
      FileOutputStream fos = new FileOutputStream(localFolderName + cc + localFileName);
      InputStream dis = new dataSocket.getInputStream();
      boolean success=true;
      try {
         int amount;
         byte[] tmp = new byte[1024];
         while ((amount = dis.read(tmp)) != -1)
            fos.write(tmp, 0, amount);
         } catch(EOFException ex) {success=false; }
      fos.close();
      dis.close();
      dataSocket.close();
      if (!success) new IOException("Ошибка приема данных");
   }
```

```
  private void upload()throws Throwable{
        char cc=(linux ? '/':'\\');
    FileInputStream fis = new FileInputStream(localFolderName + cc + localFileName);
    OutputStream dos = dataSocket.getOutputStream();
    int amount;
    byte[] tmp = new byte[1024];
    boolean success=true;
    try {
       while ((amount = fis.read(tmp)) != -1){
          dos.write(tmp, 0, amount);
          System.out.println(amount);
          }
       } catch(EOFException ex) { success=false; }
    dos.close();
    fis.close();
    dataSocket.close();
    if (!success) new IOException("Ошибка передачи данных");
    }
  }
```

Поток команд клиента реализован в основном классе **FTPClientPassive**. В нем имеется несколько методов, в которых проверяется, не наступили ли события – завершение авторизации (ответ сервера), получение IP-порта для второго соединения (ответ сервера), завершение потока передачи файла. Ожидание наступления каждого события реализовано в виде вызова соответствующего метода в цикле с «засыпанием» на каждом шаге. Т.к. ответы сервера и процесс передачи файла достаточно продолжительны, частота опроса может быть достаточно малой.

```
//-------------------------------------------------------------------------------------------------------------34_FTPClientPassive
  public FTPClientPassive(){
        this(false);
        }
  public FTPClientPassive(boolean lin){
     linux=lin;
    connected=false;
    loggedIn=false;
    localFileName=null;
    }

  public boolean isConnected(){
    return connected;
    }
  public boolean isLogged() throws Exception,Error{
    if (!connected) throw new IOException("Соединение разорвано");
    return loggedIn;
    }
  public boolean isRemoteIPValid() throws Exception,Error{
    if (!connected) throw new IOException("Соединение разорвано");
    return remotePort!=0;
    }

  private void waitForDataThread(){
    while (true){
       synchronized (this){
          if (!connected || !dataThread.isRun()) return;
          }
       }
    }
```

```
//-------------------------------------------------------------------------------------------------------34_FTPClientPassive
   public boolean connect(String host, int port){
      if (connected) {
         log("Выполняется операция с сервером");
         return false;
         }
      try {
         socket = new Socket(host, port);
         if (socket.isConnected()){
            connected=true;
            localIP=socket.getLocalAddress().getHostAddress();
            localIP=localIP.replace('.', ',');
            pw = new PrintWriter(socket.getOutputStream(), true);
            log = new ServerAnswerThread();
         }
         return true;
      }
      catch(Exception ex){
         connected=loggedIn=false;
         log(ex.getMessage());
         return false;
         }
      }
```

Метод разъединения **disconnect** прерывает поток передачи данных файла, посылает серверу команду **QUIT** и запускает поток тайм-аута. Как было отмечено выше, получение от сервера ответа с кодом 221 (а также любого кода ошибки) приводит к вызову метода **closeAll,** который прерывает поток тайм-аута и закрывает сокет (поток приема ответов получит исключение при ожидании ввода на потоке данных на этом сокете).

```
//-------------------------------------------------------------------------------------------------------34_FTPClientPassive
   public synchronized void disconnect(){
      try {
         if (dataThread != null) dataThread.interrupt();
         } catch(Exception ee){}
      try {
         sendCommand("quit","");              // Закрытие послед ответа на команду
         closeTimeOut=new Thread(){
            public void run(){                // Тайм-аут команды quit
               try { sleep(10000); } catch(Exception ee){}
               closeAll();                    // Принудительное закрытие
               }
            };
         closeTimeOut.start();
         } catch(Exception ee){
            log(ee.getMessage());
            closeAll();
            }
      }
   private Thread closeTimeOut=null;
   private synchronized void closeAll(){
      try {
         if(!connected) return;
         if (closeTimeOut!=null){
            closeTimeOut.interrupt();
            closeTimeOut=null;
            }
         connected=loggedIn=false;
         localFolderName = "";
         pw.close();
         socket.close();
         } catch(Exception ee){}
```

```
    }

//-------------------------------------------------------------------------------------------------------34_FTPClientPassive
  public synchronized void sendCommand(String command, String param) throws Exception{
    if (!connected)  throw new IOException("Команда для  неустановленного соединения");
    log(command+" "+param);
    pw.println(command+" "+param);
    //------------ Действия после основной команды ----------------------------
    if (!loggedIn)  return;
    if (command.equalsIgnoreCase("retr")){
      if (dataThread != null && dataThread.isRun)
        throw new IOException("Поток обмена данными уже выполняется");
      dataThread=new DataThread(0);
      return;
      }
    if (command.equalsIgnoreCase("stor")){
      if (dataThread != null && dataThread.isRun)
        throw new IOException("Поток обмена данными уже выполняется");
      dataThread=new DataThread(1);
      return;
      }
    }
```

Метод **copyFile** содержит основной поток команд клиента к серверу:

- Команда передачи логина - user;
- Команда передачи пароля – pass
- Ожидание в цикле в засыпанием подтверждения авторизации или кода ошибки, вызывающего закрытие сеанса;
- Команды передачи двоичных данных и установки директории сервера;
- Команда перехода в пассивный режим с ожиданием IP и порта, на который следует запрашивать соединение (устанавливается ответом сервера);
- Команда получения или передачи файла со стартом соответствующего потока, после которой выполянется цикл ожидания завершения потока;
- Команда quit с циклом ожидания закрытия соединения по тайм-ауту или по получению ответа от сервера.

```
//-------------------------------------------------------------------------------------------------------34_FTPClientPassive
public void copyFile(boolean up,String host,int port, String user,String pass,
    String path,String locname,String remDir,String remname)throws Throwable{
      if (!connect(host, port)) return;
      sendCommand("user",user);
      sendCommand("pass",pass);
      while(!isLogged()){                  // Цикл ожидания ответа на авторизацию
            Thread.sleep(100);
            if (!isConnected()) return;
            }
      sendCommand("type","I");          // Команда – режим двоичных данных
      sendCommand("cwd",remDir);    // Команда – установить директорию FTP-сервера
      sendCommand("pasv","");           // Команда перехода в пассивный режим
      while (!isRemoteIPValid())          // Ожидание удаленного IP-ПОРТА
            { Thread.sleep(1000); }
      localFolderName=path;
      localFileName=locname;            // Старт потока с отдельным соединением
      sendCommand((up ? "stor" : "retr"),remname);
      while (dataThread.isRun())          // Ожидание завершения потока
            { Thread.sleep(1000); }
```

```
        sendCommand("quit","");          // Команда – заверщить сеанс
        while(isConnected())             // Ожидание ответа, по которому закрывается соединение
                { Thread.sleep(1000); }
        log("Операция выполнена");
        }
```

Main запускает поток, у котором создает объект **FTPClientPassive** и вызывает в нем метод **copyFile,** аналогичным образом его следует запускать в оконных приложениях, чтобы не занимать поток GUI.

```
//-------------------------------------------------------------------------------------------------------34_FTPClientPassive
public static void main(String argv[]){
      new Thread(){
         public void run(){
            FTPClientPassive cl=new FTPClientPassive();
            try {

               cl.copyFile(false,"happy.kiev.ua", 21, "anonymous", "", "f:/", "nature.jpg",
                           "/pub/pictures","030007b.jpg");
               } catch(Throwable ee){
                  cl.fatal(ee.toString());
                  }
            }
      }.start();
   }
}
```

Ниже приведен пример трассировки команд клиента и ответов сервера. Как видим, ответы сервера запаздывают относительно передаваемых команд, т.к. не все команды дожидаются своих ответов.

```
user anonymous
pass
220 (vsFTPd 3.0.2)
331 Please specify the password.
230 Login successful.
type I
cwd /pub/pictures
pasv
200 Switching to Binary mode.
250 Directory successfully changed.
227 Entering Passive Mode (213,133,161,33,24,69).
retr 030007b.jpg
Connected!!!!
150 Opening BINARY mode data connection for 030007b.jpg (245760 bytes).
226 Transfer complete.
quit
221 Goodbye.
Операция выполнена
```