# 实验一Windows平台简单套接字编程

1. 实验目的

掌握Windows平台上简单的客户端与服务端的套接字编程，了解TCP与UDP的区别，掌握简单的协议设计与实现。

1. 实验内容  
   （1）在Windows上编写客户端程序和服务端程序。要求客户端程序能向服务端程序发送信息，服务端程序接收到信息后显示。

（2）分别使用面向连接的TCP协议与面向非连接的UDP协议验证“实验内容（1）”中功能，分析二者区别。

（3）在Windows上编写客户端程序和服务端程序。要求客户端程序能向服务端程序发送请求，服务端程序收到客户端请求后，对请求内容进行判断，如果是时间请求，则发送本机时间给客户端程序，否则发送欢迎信息给客户端程序。

1. 实验环境  
   （1）微机；  
   （2）网络设备（网卡，交换机，路由器）；  
   （3）Windows操作系统；

（4）VC++6.0集成开发环境。

1. 实验原理
2. TCP/UDP赋予每个服务一个唯一的协议端口号。服务器程序通过协议端口号来指定它所提供的服务，然后被动地等待通信。客户端在发送连接请求时，必须说明服务器程序运行主机的IP地址和协议端口号来指定它所希望的服务。
3. 网络编程套接字接口如表1所示。对于UDP这种无连接的网络通信，如果使用recv进行数据接收，在需要回复信息时我们无法知道回复地址，因此套接字还提供了一个recvfrom接口，它比recv多了两个参数，可以用来接收对端的地址信息，有助于UDP连接方便地进行回复。TCP由于是面向连接的，没必要每次接收记录接收地址，在accept时记录就可以了。

表1 套节字API主要接口

|  |  |
| --- | --- |
| **接口** | **解释** |
| socket  bind  listen  accept  connect  send  recv  close | 创建一个新的套接字  给服务器绑定一个传输层地址  将服务器设为被动模式  接收客户的一个请求  客户向服务器发起连接  向一个连接的套接字发送数据  从一个连接的套接字接收数据  终止一个连接 |

1. 使用面向非连接的套接字编程（UDP），套接字的工作过程如下：服务器首先启动，通过调用socket()建立一个套接字，然后调用bind()将该套接字和本地网络地址关联在一起，之后就可以调用recv()接收数据。待数据传送结束后，双方调用close()关闭套接字。
2. 使用面向连接的套接字编程（TCP），套接字工作过程如下：服务器首先启动，通过调用socket()建立一个套接字，然后调用bind()将该套接字和本地网络地址关联在一起，再调用listen()使套接字做好侦听的准备，并规定它的请求队列的长度，之后就调用accept()来接收连接。客户在建立套接字后就可调用connect()和服务器建立连接，连接一旦建立，客户机和服务器之间就可以通过调用send()和recv()来发送和接收数据。最后，待数据传送结束后，双方调用close()关闭套接字。图1显示了面向连接的套接字通信时序，TCP客户端与服务端通信的框架代码如图2、图3所示。

服务端

socket( )

　bind( )

listen( )

阻塞，等待客户数据

处理服务请求

socket( )

connect( )

send( )

客户端

请求数据

应答数据

recv( )

recv( )

accept( )

send( )

建立连接

close( )

close( )

图1.面向连接的套接字通信程序时序图

1. 实验注意事项
2. 使用ping 命令或其他工具获得服务端与客户端ip，保证ip配置无误；
3. 保证客户端请求服务的端口号与服务端设置的端口号一致，且不要与系统默认端口号或其他已设置的服务端端口号冲突，建议端口号设为6000及以上，且保证同时运行的服务端程序不会发生端口号冲突。
4. 实验设计方案（流程图或代码测试过程）

**客户端**

////////////////////////////////////////////////////////////////////////////////////////////

package NetLab;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.net.\*;

import java.util.Calendar;

/\*\*

\* @创建人：黄强

\* @时间 ：2019/8/27 10:54

\* @描述 ：TCP/UDP发送端

\*/

public class BasicFunctionSend {

public static void main(String[] args) {

//初始化发送日期消息

Calendar calendar=Calendar.getInstance();

String msg=""+calendar.get(Calendar.YEAR)+"-"+calendar.get(Calendar.MONTH)+"-"+calendar.get(Calendar.DATE)+":"+calendar.get(Calendar.HOUR)+":"+calendar.get(Calendar.MINUTE)+":"+calendar.get(Calendar.SECOND);

//发送普通消息

msg="sdghs";

TCPSend(msg);

//UDPSend(msg);

}

private static void TCPSend(String msg){

/\*\*

\*@auothor:huangqiang

\*@date:19-8-27 上午9:36

\*@parms:[msg]

\*@returns:void

\*@description:TCP消息发送

\*/

System.out.println("Sent msg:"+msg);

Socket clientSocket=null;

byte data[]=new byte[1024];

try {

//初始化服务器地址和端口

byte [] serverIP={127,0,0,1};

clientSocket = new Socket(InetAddress.getByAddress(serverIP), 23333);

//向服务器发消息

OutputStream outputStream=clientSocket.getOutputStream();

outputStream.write(msg.getBytes());

//获得服务器回复消息

InputStream inputStream=clientSocket.getInputStream();

int length=inputStream.read(data);

System.out.println("Back MSG:"+new String(data,0,length)+" from :Server IP:"+clientSocket.getInetAddress().getHostAddress());

} catch (UnknownHostException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}finally {

//关闭资源

try {

clientSocket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

private static void UDPSend(String msg){

/\*\*

\*@auothor:huangqiang

\*@date:19-8-27 上午10:40

\*@parms:[msg]

\*@returns:void

\*@description:UDP发送消息

\*/

System.out.println("Sent msg:"+msg);

//客户端套接字

DatagramSocket datagramSocket =null;

try {

//初始化收方IP

byte IPAddress []={(byte)127,0,0,1};

InetAddress inetAddress=InetAddress.getByAddress(IPAddress);

byte[] reciveBuffer=new byte[1024];

//实例化socket对象。

datagramSocket=new DatagramSocket();

//构建数据。

DatagramPacket sendPacket=new DatagramPacket(msg.getBytes(),msg.length(),inetAddress,23333);

//发送。

datagramSocket.send(sendPacket);

//构建接受Packet。

DatagramPacket recivePacket=new DatagramPacket(reciveBuffer,reciveBuffer.length);

//接受消息。

datagramSocket.receive(recivePacket);

System.out.println("back MSG:"+new String(reciveBuffer,0,recivePacket.getLength()));

} catch (SocketException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}finally {

//关闭套接字

datagramSocket.close();

}

}

}

**服务器端**

////////////////////////////////////////////////////////////////////////////////////////////

package NetLab;

import java.io.IOException;

import java.net.\*;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.Date;

import java.util.concurrent.\*;

/\*\*

\* @创建人：黄强

\* @时间 ：2018/8/27 11:54

\* @描述 ：TCO/UDP接收端。

\*/

public class BasicFunctionRec {

public static void main(String[] args) {

TCPRec();

}

/\*\*

\*@auothor:huangqiang

\*@date:19-8-27 上午9:37

\*@parms:[]

\*@returns:void

\*@description:TCP多线程处理请求

\*/

private static void TCPRec() {

Socket clientSocket=null;

//初始化服务器套接字

ServerSocket serverSocket = null;

try {

serverSocket = new ServerSocket(23333);

} catch (IOException e) {

e.printStackTrace();

}

//创建线程池，核心池：5，最大线程数：10，最大空闲存活时间：200，最大空闲存活时间单位，阻塞线程容器大小：5

ThreadPoolExecutor executor = new ThreadPoolExecutor(5, 10, 200, TimeUnit.MILLISECONDS, new ArrayBlockingQueue<>(5));

while (true) {

try {

//接受请求，获得客户端套接字

clientSocket = serverSocket.accept();

//把线程请求交给线程池，并发处理请求。

executor.execute(new TCPThreadServer(clientSocket));

System.out.println("线程池中线程数目："+executor.getPoolSize()+"，队列中等待执行的任务数目："+

executor.getQueue().size()+"，已执行玩别的任务数目："+executor.getCompletedTaskCount());

} catch (IOException e) {

e.printStackTrace();

}

}

} private static void UDPRec(){

/\*\*

\*@auothor:huangqiang

\*@date:19-8-27 上午10:39

\*@parms:[]

\*@returns:void

\*@description:UDP接收消息。

\*/

while (true) {

//服务器套接字

DatagramSocket datagramSocket = null;

try {

//初始化套接字信息。

byte message[] = new byte[1024];

datagramSocket = new DatagramSocket(23333);

DatagramPacket datagramPacket = new DatagramPacket(message, message.length);

String backMsg = "";

//接受客户端套接字

datagramSocket.receive(datagramPacket);

//获得客户端相关信息

InetAddress IP = datagramPacket.getAddress();

int Port = datagramPacket.getPort();

//打印相关消息

System.out.println("Rec Msg:" + new String(message, 0, datagramPacket.getLength()) + " from UDP client"+" \tsrc IP:" + IP + "\tsrc Port:" + Port);

SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd:HH:mm:ss");

//通过类型转换判断发送的是否是日期消息

try {

Date date = simpleDateFormat.parse(new String(message, 0, datagramPacket.getLength()));

Calendar calendar = Calendar.getInstance();

backMsg = calendar.get(Calendar.YEAR) + "-" + calendar.get(Calendar.MONTH) + "-" + calendar.get(Calendar.DATE) + ":" + calendar.get(Calendar.HOUR) + ":" + calendar.get(Calendar.MINUTE) + ":" + calendar.get(Calendar.SECOND)+" from UDP Server";

} catch (ParseException e) {

backMsg = "hello"+" from UDP Server";

}

//构建发送packet。

datagramPacket = new DatagramPacket(backMsg.getBytes(), backMsg.length(), IP, Port);

datagramSocket.send(datagramPacket);

} catch (SocketException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} finally {

//关闭资源

datagramSocket.close();

}

}

}

}

**TCP多线程处理实现**

////////////////////////////////////////////////////////////////////////////////////////////

package NetLab;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.net.Socket;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.Date;

/\*\*

\* @创建人：黄强

\* @时间 ：2018/8/27 11:34

\* @描述 ：TCP多线程处理实现

\*/

public class TCPThreadServer implements Runnable{

Socket socket;

public TCPThreadServer(Socket clientSocket){

this.socket =clientSocket;

}

@Override

/\*\*

\*@auothor:huangqiang

\*@date:19-8-27 上午9:39

\*@parms:[]

\*@returns:void

\*@description:重写run方法，用于处理链接处理。

\*/

public void run(){

//回显消息

String backMSG = "";

String name="";

try {

//Thread.sleep(3000);

//Thread.activeCount();

Thread t = Thread.currentThread();

name = t.getName();

//获得客户端请求消息

InputStream inputStream = socket.getInputStream();

byte[] data = new byte[1024];

int length = inputStream.read(data);

System.out.println("Rec MSG:" + new String(data, 0, length) + " from UDP client"+ " from Client IP:" + socket.getInetAddress().getHostAddress()+" 处理线程："+name);

//通过String 到Date类型的转换来判断发送消息是否是日期消息。

SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd:HH:mm:ss");

Date date = simpleDateFormat.parse(new String(data, 0, length));

//回显消息

Calendar calendar = Calendar.getInstance();

backMSG = calendar.get(Calendar.YEAR) + "-" + calendar.get(Calendar.MONTH) + "-" + calendar.get(Calendar.DATE) + ":" + calendar.get(Calendar.HOUR) + ":" + calendar.get(Calendar.MINUTE) + ":" + calendar.get(Calendar.SECOND)+" from TCP Server"+" 处理线程："+name;

} catch (IOException e) {

e.printStackTrace();

} catch (ParseException e) {

//发送非日期消息时的回显消息

backMSG = "Hello"+" from TCP Server"+" 处理线程："+name;

} finally {

//关闭资源

try {

OutputStream outputStream = socket.getOutputStream();

outputStream.write(backMSG.getBytes());

socket.close();

} catch (IOException e) {

e.printStackTrace();

}

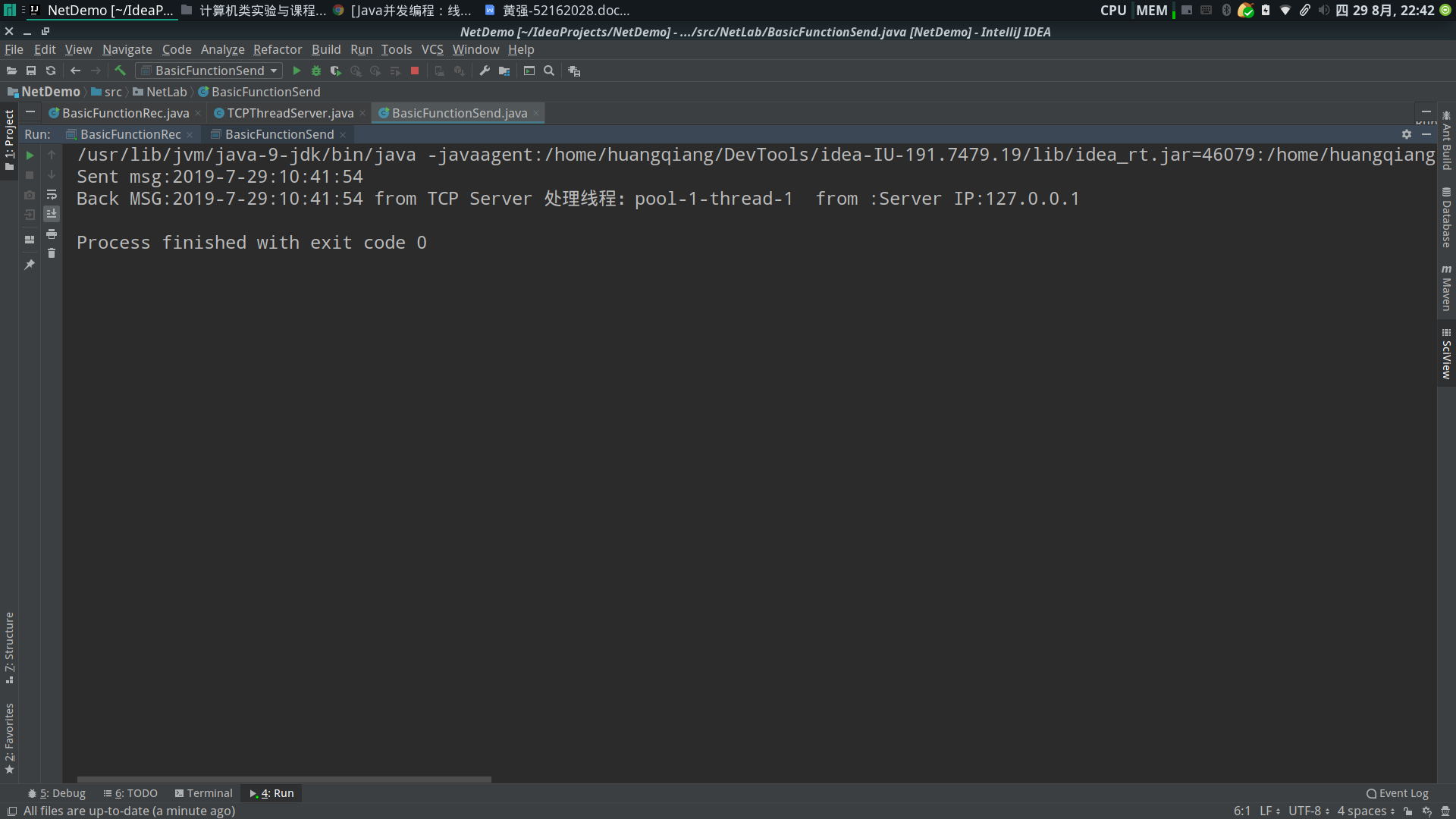
}

}

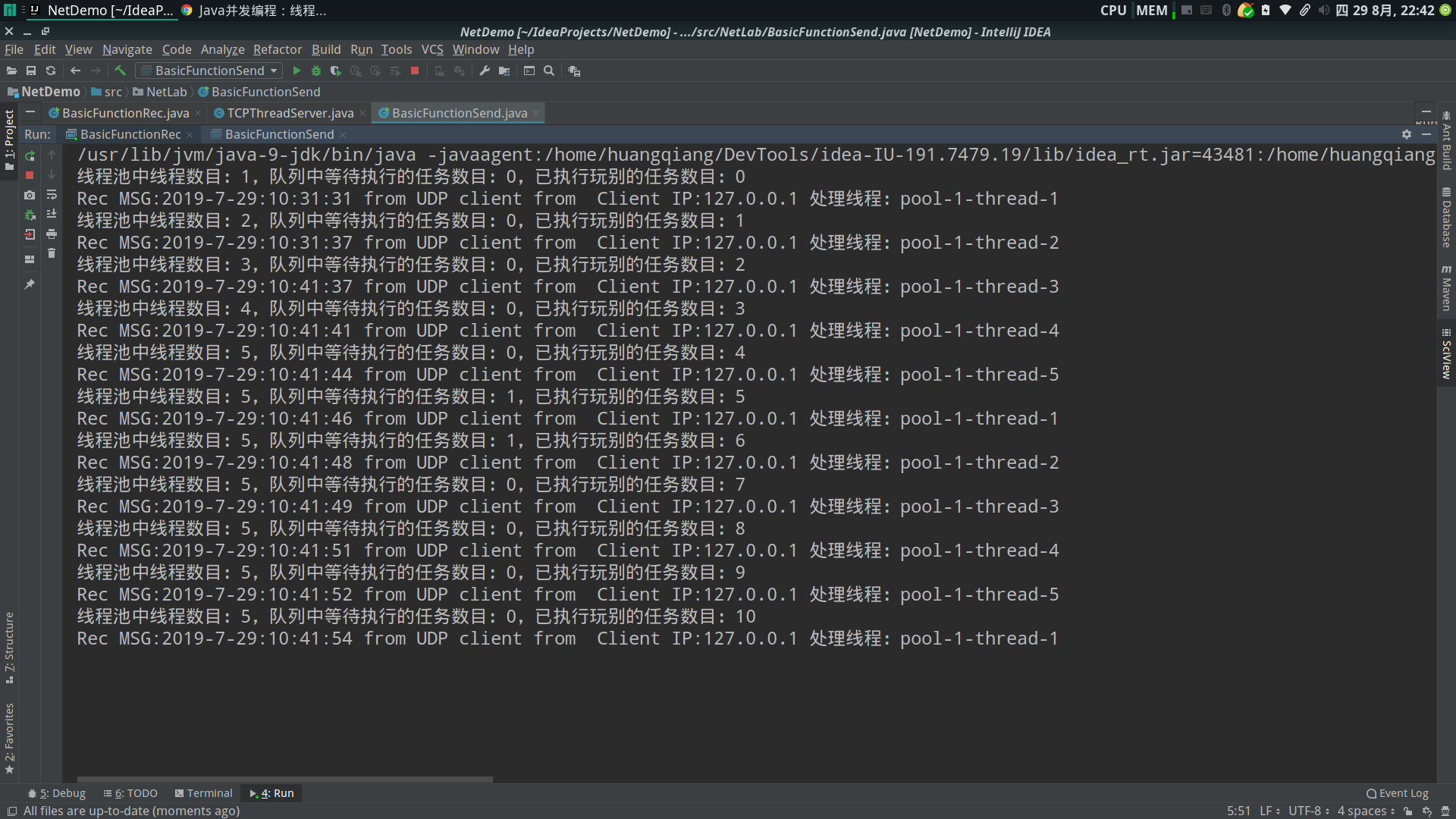
}

七： 实验数据及结论

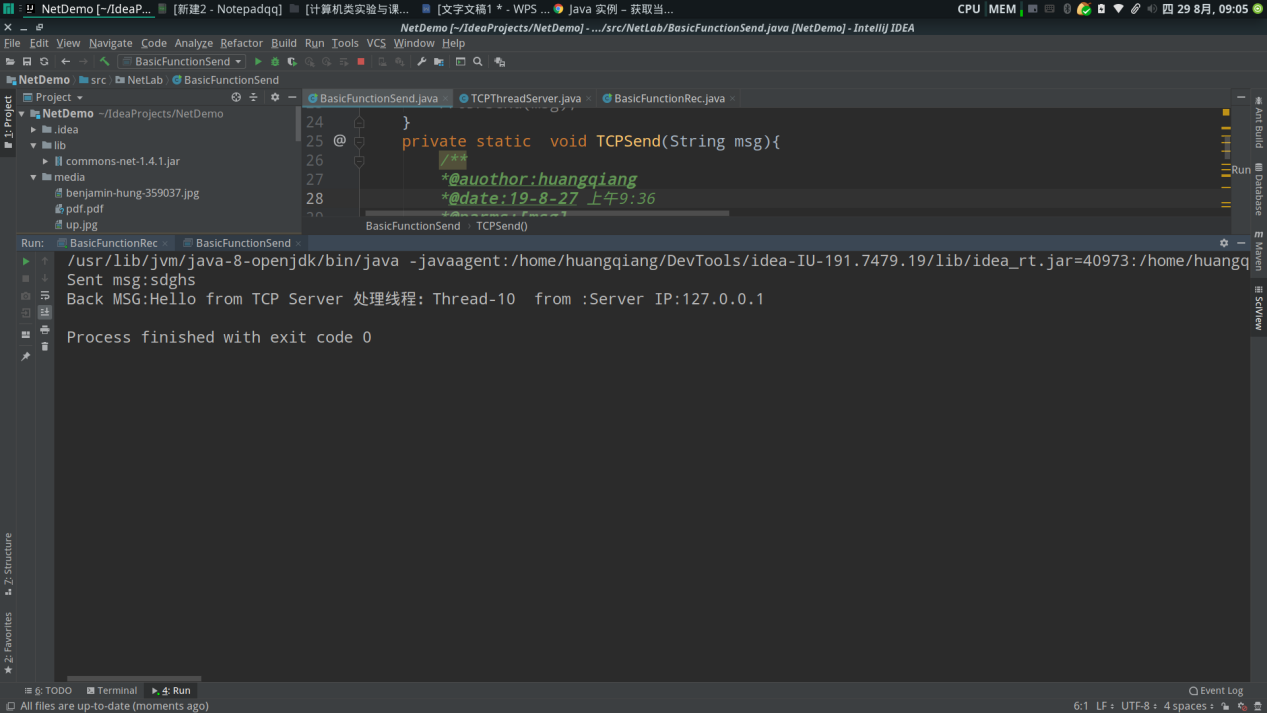
1：客户端用TCP发送日期消息时候的显示



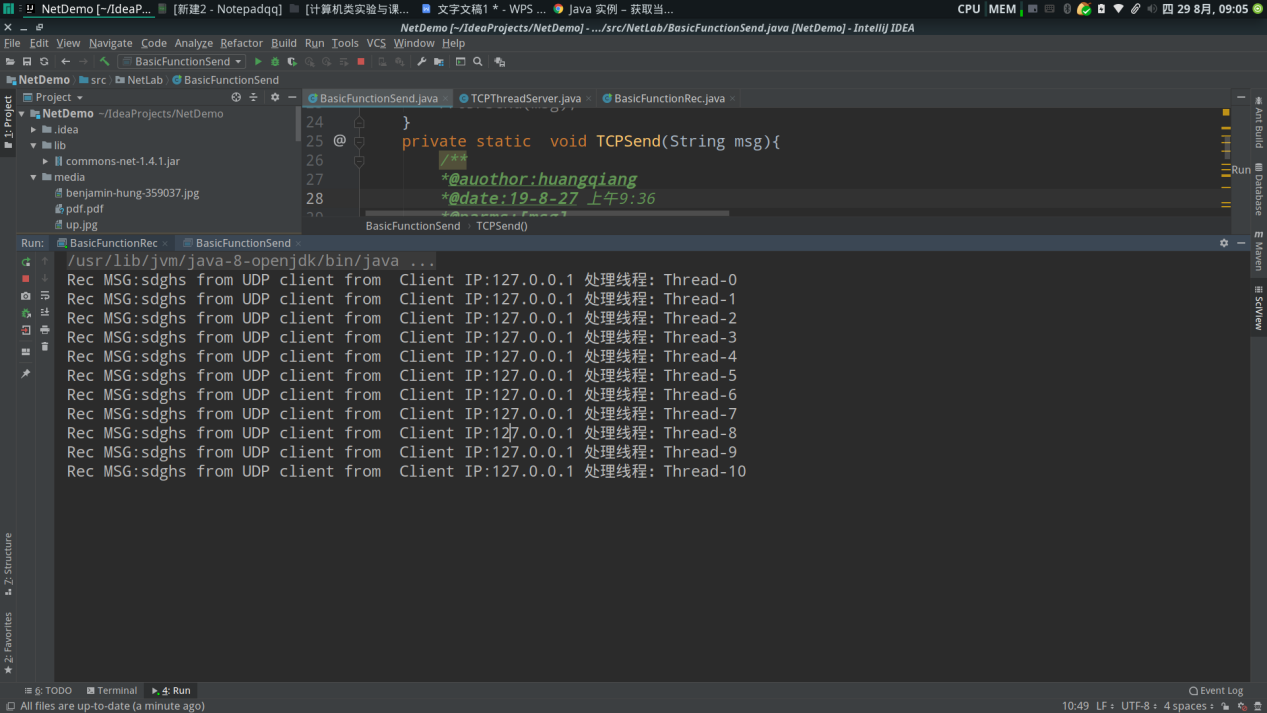
2：服务器端在多线程下用TCP接收日期消息时候的显示



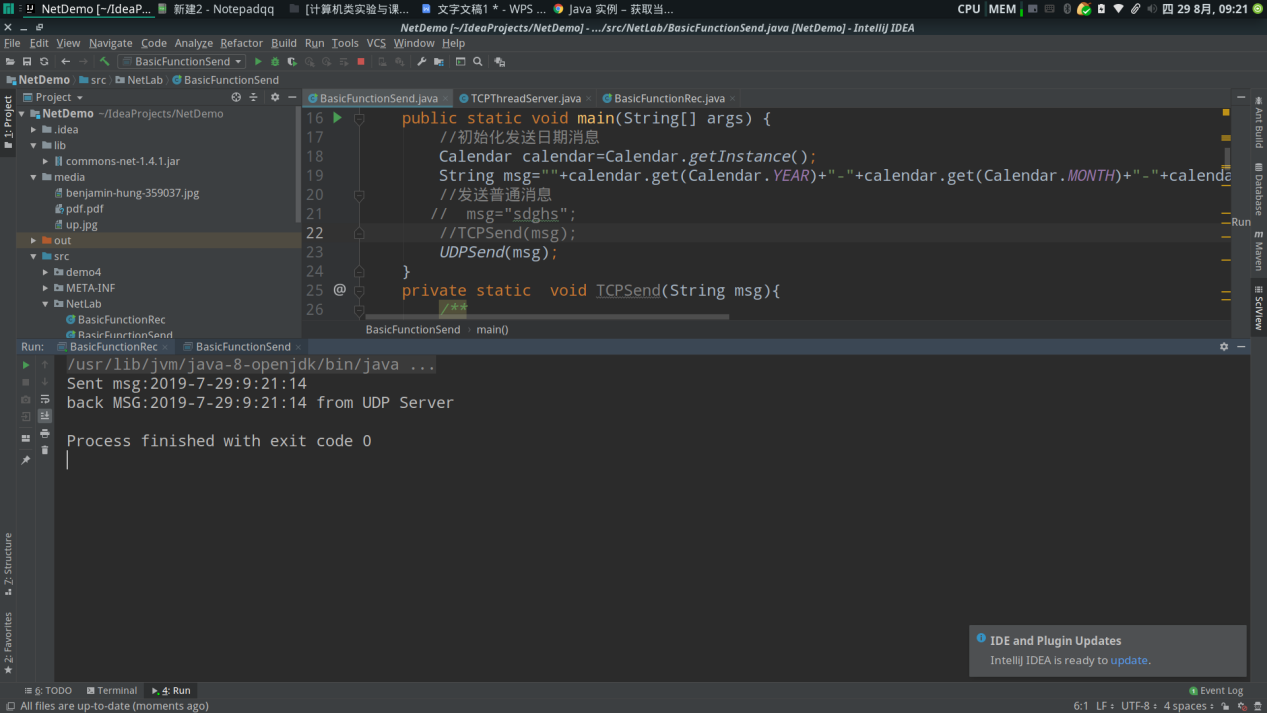
3：客户端用TCP发送非日期消息时候的显示



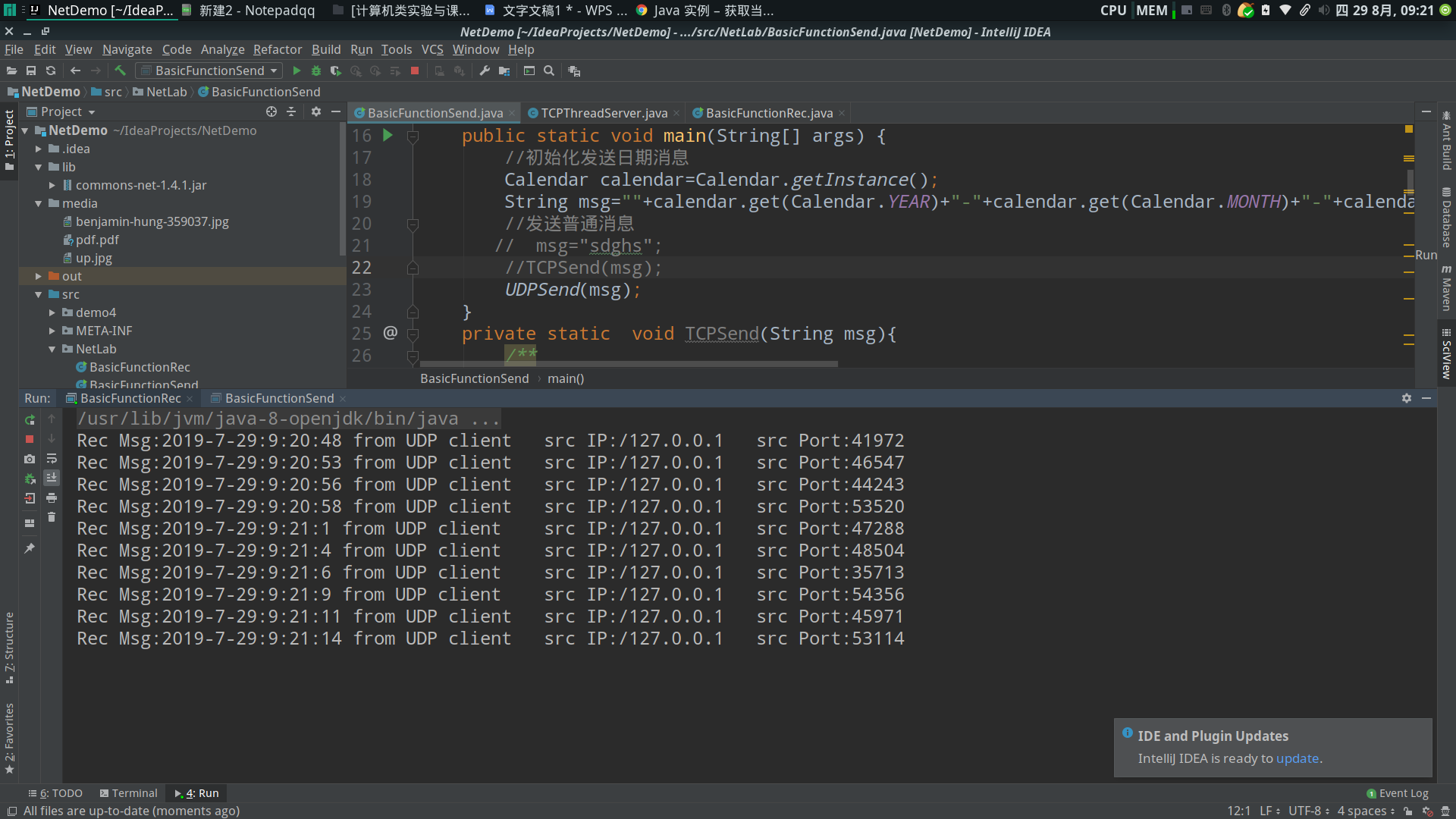
4：服务器端在多线程下用TCP接收非日期消息时候的显示



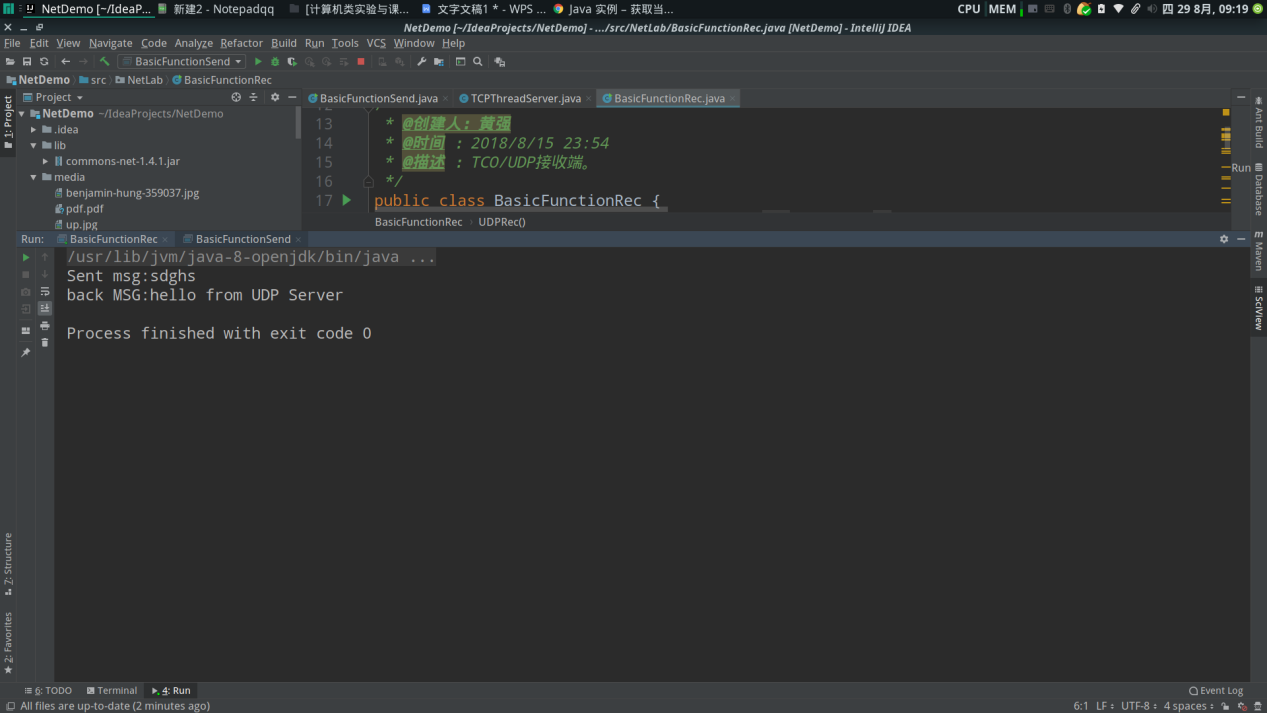
5：客户端用UDP发送日期消息时候的显示



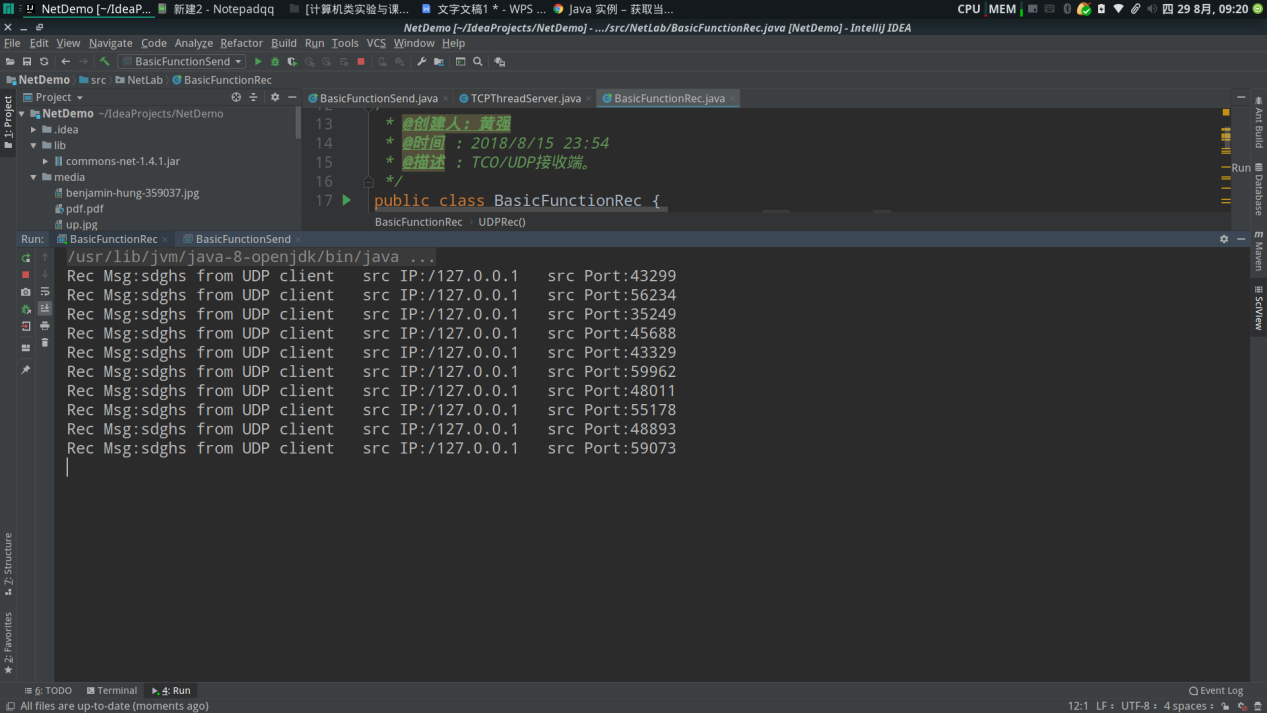
6：服务器端用UDP接收日期消息时候的显示



7：客户端用UDP发送非日期消息时候的显示



8：服务器端用UDP接收非日期消息时候的显示



FTP/UDP间区别：

UDP 与 TCP 的主要区别在于 UDP 不一定提供可靠的数据传输。事实上，该协议不能保证数据准确无误地到达目的地。UDP 在许多方面非常有效。当某个程序的目标是尽快地传输尽可能多的信息时（其中任意给定数据的重要性相对较低），可使用 UDP。ICQ 短消息使用 UDP 协议发送消息。 许多程序将使用单独的TCP连接和单独的UDP连接。重要的状态信息随可靠的TCP连接发送，而主数据流通过UDP发送。TCP： TCP的目的是提供可靠的数据传输，并在相互进行通信的设备或服务之间保持一个虚拟连接。TCP在数据包接收无序、丢失或在交付期间被破坏时，负责数据恢复。它通过为其发送的每个数据包提供一个序号来完成此恢复。记住，较低的网络层会将每个数据包视为一个独立的单元，因此，数据包可以沿完全不同的路径发送，即使它们都是同一消息的组成部分。这种路由与网络层处理分段和重新组装数据包的方式非常相似，只是级别更高而已。 为确保正确地接收数据，TCP要求在目标计算机成功收到数据时发回一个确认（即 ACK）。如果在某个时限内未收到相应的 ACK，将重新传送数据包。如果网络拥塞，这种重新传送将导致发送的数据包重复。但是，接收计算机可使用数据包的序号来确定它是否为重复数据包，并在必要时丢弃它。TCP与UDP区别总结：1、TCP面向连接（如打电话要先拨号建立连接）;UDP是无连接的，即发送数据之前不需要建立连接2、TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP尽最大努力交付，即不保证可靠交付3、TCP面向字节流，实际上是TCP把数据看成一连串无结构的字节流;UDP是面向报文的UDP没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如IP电话，实时视频会议等）4、每一条TCP连接只能是点到点的;UDP支持一对一，一对多，多对一和多对多的交互通信5、TCP首部开销20字节;UDP的首部开销小，只有8个字节6、TCP的逻辑通信信道是全双工的可靠信道，UDP则是不可靠信道

# 实验四 FTP综合应用编程（选做）

1. 实验目的

掌握FTP应用编程。

1. 实验内容  
   （1）编写ftp客户端程序，能够从指定ftp服务器中下载文件；  
   （2）设计并实现一个支持多个客户端的文件传输服务器，客户端从键盘输入文件名称，发送给服务器，服务器找到该文件并发送回客户端，客户端接收完文件后将文件保存在本地，否则提示客户端文件不存在。
2. 实验环境  
   （1）微机；  
   （2）网络设备（网卡，交换机，路由器）；  
   （3）Windows操作系统，安装FileZilla\_Server-0\_8\_4；

（4）VC++6.0集成开发环境。

1. 实验原理

开发FTP客户端程序首先要与服务端建立TCP连接，之后调用send（）接口向已建立的TCP连接发送FTP命令实现文件传输相关的功能，服务端返回数据通过recv（）接口接收。FTP协议的命令列表与响应码如表2、表3所示。

1. 实验步骤
2. 创建Win32 Console Application 工程；
3. 将ws2\_32.lib 加入到工程设置link 模块中；
4. 设计编码；
5. 联调，测试。
6. 实验注意事项
7. 使用ping 命令或其他工具获得服务端与客户端ip，保证ip配置无误；
8. ftp的默认端口号为21，如ftp服务器更改端口号，客户端程序需使用更改后的端口号连接服务器。
9. 实验设计方案（流程图或代码测试过程）

FTP工具类

////////////////////////////////////////////////////////////////////////////////////////////

package NetLab;

/\*\*

\*@author:huangqiang

\*@date:19-8-27

\*@description:ftp工具类,实现文件的上传与下载

\*/

import org.apache.commons.net.ftp.FTPClient;

import org.apache.commons.net.ftp.FTPFile;

import org.apache.commons.net.ftp.FTPReply;

import java.io.\*;

import java.net.SocketException;

import java.util.ArrayList;

import java.util.Formatter;

import java.util.Scanner;

public class FtpUtil {

//文件夹深度

static int deepth = 0;

//格式化输出

static Formatter formatter = new Formatter(System.out);

static ArrayList<FileBean> fileBeans=new ArrayList<>();

static ConnectionPool connectionPool=new ConnectionPool(5);

public static boolean downloadFtpFile( String ftpPath, String localPath,String targetFileName) {

/\*\*

\*@auothor:huangqiang

\*@date:19-8-28 上午10:10

\*@parms:[ftpHost, ftpUserName, ftpPassword, ftpPort, ftpPath, localPath, fileName]

\*@returns:void

\*@description:FTP下载文件

\*/

FTPClient ftpClient = null;

boolean result = false;

Scanner scanner = new Scanner(System.in);

try {

//获得连接

ftpClient=connectionPool.getConnection();

//基础设置

ftpClient.setControlEncoding("UTF-8");

ftpClient.setFileType(FTPClient.BINARY\_FILE\_TYPE);

ftpClient.enterLocalPassiveMode();

ftpClient.changeWorkingDirectory(ftpPath);

//下载文件

System.out.print("输入本地存储的文件名：");

String localFileName = scanner.nextLine();

File localFile = new File(localPath + File.separatorChar + localFileName);

//判断文件是否已经存在。

while (true) {

if (localFile.exists()) {

//覆盖原文件

System.out.println("文件已经存在，是否覆盖？ y/n");

String choose = scanner.nextLine();

// System.out.println(choose);

if (choose.equals("y")) {

break;

} else {

//重命名新文件

if (choose.equals("n")) {

System.out.println("请输入新的文件名：");

localFileName = scanner.nextLine();

localFile = new File(localPath + File.separatorChar + localFileName);

}

}

} else {

break;

}

}

//存储文件

OutputStream os = new FileOutputStream(localFile);

result = ftpClient.retrieveFile(targetFileName, os);

os.close();

connectionPool.closeClient(ftpClient);

} catch (FileNotFoundException e) {

System.out.println("没有找到" + ftpPath + "文件");

e.printStackTrace();

} catch (SocketException e) {

System.out.println("连接FTP失败.");

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

System.out.println("文件读取错误。");

e.printStackTrace();

}

if (result){

System.out.println("下载成功！");

}

return result;

}

public static boolean uploadFile( String ftpPath, String fileName, InputStream input) {

/\*\*

\*@auothor:huangqiang

\*@date:19-8-28 上午10:10

\*@parms:[ftpHost, ftpUserName, ftpPassword, ftpPort, ftpPath, fileName, input]

\*@returns:boolean

\*@description:FTP上传文件

\*/

boolean success = false;

FTPClient ftpClient = null;

try {

//获得连接

int reply;

ftpClient=connectionPool.getConnection();

reply = ftpClient.getReplyCode();

if (!FTPReply.isPositiveCompletion(reply)) {

ftpClient.disconnect();

return success;

}

//基础设置

ftpClient.setControlEncoding("UTF-8");

ftpClient.setFileType(FTPClient.BINARY\_FILE\_TYPE);

ftpClient.enterLocalPassiveMode();

ftpClient.changeWorkingDirectory(ftpPath);

//上传文件

ftpClient.storeFile(fileName, input);

input.close();

connectionPool.closeClient(ftpClient);

success = true;

} catch (IOException e) {

e.printStackTrace();

} finally {

if (ftpClient.isConnected()) {

try {

ftpClient.disconnect();

} catch (IOException ioe) {

}

}

}

return success;

}

public static boolean deleteFile( String pathname, String filename) {

/\*\*

\*@auothor:huangqiang

\*@date:19-8-29 上午9:40

\*@parms:[ftpHost, ftpUserName, ftpPassword, ftpPort, pathname, filename]

\*@returns:boolean

\*@description:删除文件

\*/

boolean success = false;

FTPClient ftpClient = null;

try {

//获得连接

int reply;

ftpClient=connectionPool.getConnection();

reply = ftpClient.getReplyCode();

if (!FTPReply.isPositiveCompletion(reply)) {

ftpClient.disconnect();

return success;

}

//基础设置，开始删除文件

ftpClient.changeWorkingDirectory(pathname);

ftpClient.dele(filename);

connectionPool.closeClient(ftpClient);

success = true;

if (success){

System.out.println("删除文件成功");

}

} catch (IOException ex) {

ex.printStackTrace();

}

return success;

}

public static void browser(String ftpHost, String ftpUserName, String ftpPassword, int ftpPort, String pathname, String localPath) {

/\*\*

\*@auothor:huangqiang

\*@date:19-8-29 下午5:36

\*@parms:[ftpHost, ftpUserName, ftpPassword, ftpPort, pathname, localPath]

\*@returns:void

\*@description:查询文件

\*/

FTPClient myFTPClient = null;

//获得连接

int reply;

try {

myFTPClient=connectionPool.getConnection();

reply = myFTPClient.getReplyCode();

if (!FTPReply.isPositiveCompletion(reply)) {

myFTPClient.disconnect();

}

//基础设置。

myFTPClient.enterLocalPassiveMode();

myFTPClient.setFileType(FTPClient.BINARY\_FILE\_TYPE);

myFTPClient.setControlEncoding("UTF-8");

myFTPClient.changeWorkingDirectory(pathname);

FTPFile[] files = myFTPClient.listFiles();

String blocks = "";

for (int i = 0; i < deepth; ++i) {

blocks += "---";

}

System.out.println("Path:"+pathname);

//显示当前文件路径下文件。

System.out.print(blocks);

formatter.format("%-5s %-17s %-10s %-10s\n", "编号", "文件名", "文件类型","大小");

// System.out.println(blocks+" "+"编号\t\t文件名\t\t\t文件类型");

blocks += ">";

for (int i = 0; i < files.length; i++) {

if (files[i].isFile()) {

System.out.print(blocks + " " + i);

formatter.format("\t %-20s %-10s %-10s\n", files[i].getName(), "文件",files[i].getSize()/2014+"KB");

} else {

System.out.print(blocks + " " + i);

formatter.format("\t %-20s %-10s %-10s\n", files[i].getName(), "文件夹",files[i].getSize()/2014+"KB");

}

}

connectionPool.closeClient(myFTPClient);

//下一步路径选择

System.out.print("输入要浏览文件夹或者文件编号(-1 返回上一层,-2 结束访问)：");

Scanner scanner = new Scanner(System.in);

int index = Integer.parseInt(scanner.nextLine());

if (index == -1) {

//返回上一级

deepth--;

if (deepth >= 0) {

int sepIndex = pathname.lastIndexOf(File.separator);

pathname = pathname.substring(0, sepIndex);

// System.out.println(pathname);

} else {

//根目录下无法再返回

System.err.println("已在根目录，无法继续返回上一层！");

}

//返回上一级

browser(ftpHost, ftpUserName,

ftpPassword, ftpPort, pathname, localPath);

} else {

//推出查询

if (index == -2) {

System.exit(0);

} else {

//查寻下一级文件夹

if (files[index].isDirectory()) {

deepth++;

browser(ftpHost, ftpUserName,

ftpPassword, ftpPort, pathname + File.separator + files[index].getName(), localPath);

} else {

//文件操作

System.out.print("选择下载或者删除文件 0：下载，1：删除 ");

int choose = Integer.parseInt(scanner.nextLine());

//下载

if (choose == 0) {

downloadFtpFile( pathname, localPath,files[index].getName());

} else {

//删除

deleteFile( pathname, files[index].getName());

}

//显示当前目录文件列表

browser(ftpHost, ftpUserName, ftpPassword, ftpPort, pathname, localPath);

}

}

}

} catch (IOException e) {

e.printStackTrace();

}

}

private static void listFiles(String pathname) {

/\*\*

\*@auothor:huangqiang

\*@date:19-8-29 下午8:50

\*@parms:[ftpHost, ftpUserName, ftpPassword, ftpPort, pathname]

\*@returns:void

\*@description:获得所有文件列表。

\*/

FTPClient ftpClient = null;

//获得连接

int reply;

try {

ftpClient=connectionPool.getConnection();

reply = ftpClient.getReplyCode();

if (!FTPReply.isPositiveCompletion(reply)) {

ftpClient.disconnect();

}

//基础设置。

ftpClient.enterLocalPassiveMode();

ftpClient.setFileType(FTPClient.BINARY\_FILE\_TYPE);// 设置传输的模式，以二进制流的方式读取

ftpClient.setControlEncoding("UTF-8");

System.out.println(ftpClient.printWorkingDirectory());

ftpClient.changeWorkingDirectory(pathname);

//获得当前路径下文件列表

FTPFile[] files = ftpClient.listFiles();

connectionPool.closeClient(ftpClient);

//存储文件的信息，包括名称、大小、路径、文件类型

for (int i = 0; i < files.length; i++) {

if (files[i].isFile()) {

String name=files[i].getName();

//文件类型判断

String type;

if (name.lastIndexOf(".")==-1){

type=name;

}else {

type=name.substring(name.lastIndexOf(".")+1);

}

//存储文件了信息。

FileBean fileBean=new FileBean(name,pathname,type,files[i].getSize()/1024);

fileBeans.add(fileBean);

}else {

listFiles(pathname+File.separator+files[i].getName());

}

}

} catch (IOException e) {

e.printStackTrace();

}

}

public static void search(String ftpHost, String ftpUserName, String ftpPassword, int ftpPort, String pathname,String localPath){

/\*\*

\*@auothor:huangqiang

\*@date:19-8-29 下午8:53

\*@parms:[ftpHost, ftpUserName, ftpPassword, ftpPort, pathname, localPath]

\*@returns:void

\*@description:查询文件。

\*/

listFiles(pathname);

//显示全部文件

formatter.format("%-3s %-39s %-6s %-54s %-5s\n", "编号", "文件名", "文件类型","路径","文件大小");

for (int i=0;i<fileBeans.size();++i){

FileBean fileBean=fileBeans.get(i);

formatter.format("%-5s %-40s %-10s %-55s %-10s\n", i, fileBean.getName(), fileBean.getType(),fileBean.getPath(),fileBean.getSize()+"KB");

}

Scanner scanner=new Scanner(System.in);

System.out.print("要查找的文件名(输入 -1 退出)：");

String name= scanner.nextLine();

try {

if(Integer.parseInt(name)==-1){

return;

}

}catch (Exception e){

//Null

}

formatter.format("%-3s %-39s %-6s %-54s %-5s\n", "编号", "文件名", "文件类型","路径","文件大小");

//查找文件

int count=0;

for (int i=0;i<fileBeans.size();++i){

FileBean fileBean=fileBeans.get(i);

if (fileBean.getName().equals(name)){

count++;

formatter.format("%-5s %-40s %-10s %-55s %-10s\n", i, fileBean.getName(), fileBean.getType(),fileBean.getPath(),fileBean.getSize()+"KB");

}

}

//查找结果处理

if (count==0){

System.out.println("未找到该文件！");

System.out.println("是否重新查找：y/n");

String choose = scanner.nextLine();

// 重新查找

if (choose.equals("y")) {

fileBeans=new ArrayList<>();

search(ftpHost,ftpUserName,ftpPassword,ftpPort,pathname,localPath);

} else {

//结束查找

if (choose.equals("n")) {

return;

}

}

}else {

//选择目标文件

System.out.print("选择文件：");

int index=Integer.parseInt(scanner.nextLine());

FileBean target=fileBeans.get(index);

//目标文件操作

System.out.print("选择下载或者删除文件 0：下载，1：删除 ");

int choose = Integer.parseInt(scanner.nextLine());

//下载

if (choose == 0) {

downloadFtpFile( target.getPath(), localPath,target.getName());

} else {

//删除

deleteFile(target.getPath(), target.getName());

}

fileBeans=new ArrayList<>();

search(ftpHost,ftpUserName,ftpPassword,ftpPort,pathname,localPath);

}

}

}

FTP连接池类

////////////////////////////////////////////////////////////////////////////////////////////

package NetLab;

import org.apache.commons.net.ftp.FTPClient;

import org.apache.commons.net.ftp.FTPFile;

import org.apache.commons.net.ftp.FTPReply;

import java.io.IOException;

import java.net.SocketException;

import java.util.ArrayList;

public class ConnectionPool {

/\*\*

\*@author:huangqiang

\*@date:19-8-30

\*@description:连接池

\*/

ArrayList<FTPClient> ftpClients=new ArrayList<>();

//池总大小-1

int topIndex;

public ConnectionPool(int maxConnection) {

topIndex =maxConnection-1;

//配置信息

String ftpHost = "127.0.0.1";

String ftpUserName = "huangqiang";

String ftpPassword = "huangqiang";

int ftpPort = 21;

//获得多个连接

for (int i = 0; i< topIndex +1; ++i){

FTPClient ftpClient = new FTPClient();

try {

ftpClient = new FTPClient();

// 连接FTP服务器

ftpClient.connect(ftpHost, ftpPort);

// 登陆FTP服务器

ftpClient.login(ftpUserName, ftpPassword);

//状态查询

if (!FTPReply.isPositiveCompletion(ftpClient.getReplyCode())) {

System.out.println("未连接到FTP，用户名或密码错误。");

ftpClient.disconnect();

} else {

System.out.println("FTP连接成功。");

}

} catch (SocketException e) {

e.printStackTrace();

System.out.println("FTP的IP地址可能错误，请正确配置。");

} catch (IOException e) {

e.printStackTrace();

System.out.println("FTP的端口错误,请正确配置。");

}

ftpClients.add(ftpClient);

}

}

public FTPClient getConnection(){

/\*\*

\*@auothor:huangqiang

\*@date:19-8-30 上午10:43

\*@parms:[]

\*@returns:org.apache.commons.net.ftp.FTPClient

\*@description:获得连接

\*/

if (topIndex >=0){

topIndex--;

return ftpClients.remove(topIndex +1);

}

System.out.println("连接池已空");

return null;

}

public void closeClient(FTPClient ftpClient){

/\*\*

\*@auothor:huangqiang

\*@date:19-8-30 上午10:43

\*@parms:[]

\*@returns:void

\*@description:归还连接

\*/

try {

ftpClient.changeWorkingDirectory("/");

} catch (IOException e) {

e.printStackTrace();

}

ftpClients.add(ftpClient);

topIndex++;

}

public void destoryAll(){

/\*\*

\*@auothor:huangqiang

\*@date:19-8-30 上午10:43

\*@parms:[]

\*@returns:void

\*@description:摧毁所有连接

\*/

for (int i=0;i<ftpClients.size();++i){

try {

FTPClient f =ftpClients.get(i);

f.logout();

f.disconnect();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

FTP测试类

////////////////////////////////////////////////////////////////////////////////////////////

package NetLab;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

public class FtpTest {

public static void main(String[] args) {

//配置信息

String ftpHost = "127.0.0.1";

String ftpUserName = "huangqiang";

String ftpPassword = "huangqiang";

int ftpPort = 21;

//ftp服务器上文件夹路径

String ftpPath = "data";

//本地文件路径。上传：全路径，带文件名。下载：要存储的文件夹路径。

String localPath = "/home/huangqiang/IdeaProjects/NetDemo/media/";

//表示上传ftp服务器端名称。

String targetFileName = "pink.jpg";

//上传一个文件

try{

FileInputStream in=new FileInputStream(new File(localPath));

boolean test = FtpUtil.uploadFile( ftpPath, targetFileName,in);

if(test){

System.out.println("上传成功");

}else {

System.out.println("上传失败");

}

} catch (FileNotFoundException e){

e.printStackTrace();

System.out.println(e);

}

// 下载一个文件

boolean result=FtpUtil.downloadFtpFile( ftpPath, localPath,targetFileName);

if (result){

System.out.println("下载成功！");

}else {

System.out.println("下载失败！");

}

//删除文件

FtpUtil.deleteFile( ftpPath,targetFileName);

//浏览目录

FtpUtil.browser(ftpHost,ftpUserName,ftpPassword,ftpPort,ftpPath,localPath);

//文件查找

FtpUtil.search(ftpHost,ftpUserName,ftpPassword,ftpPort,ftpPath,localPath);

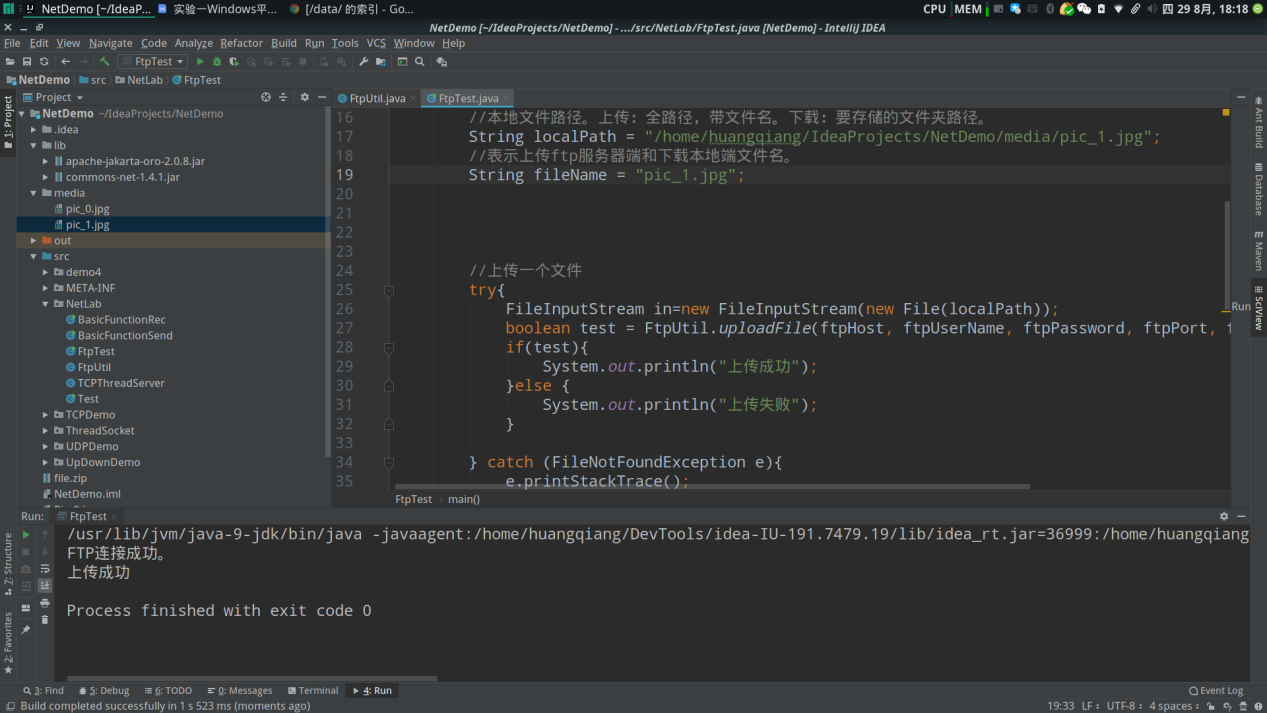
}

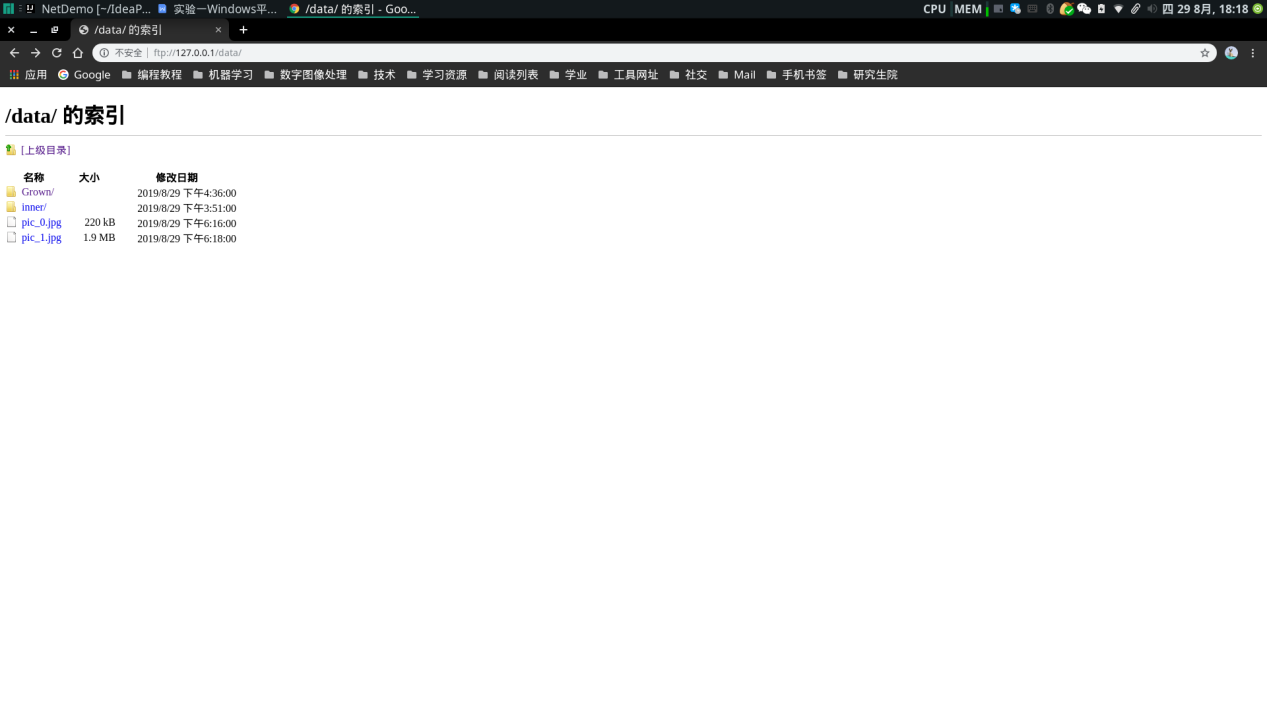
}八：实验数据及结论

1：原始文件目录结构

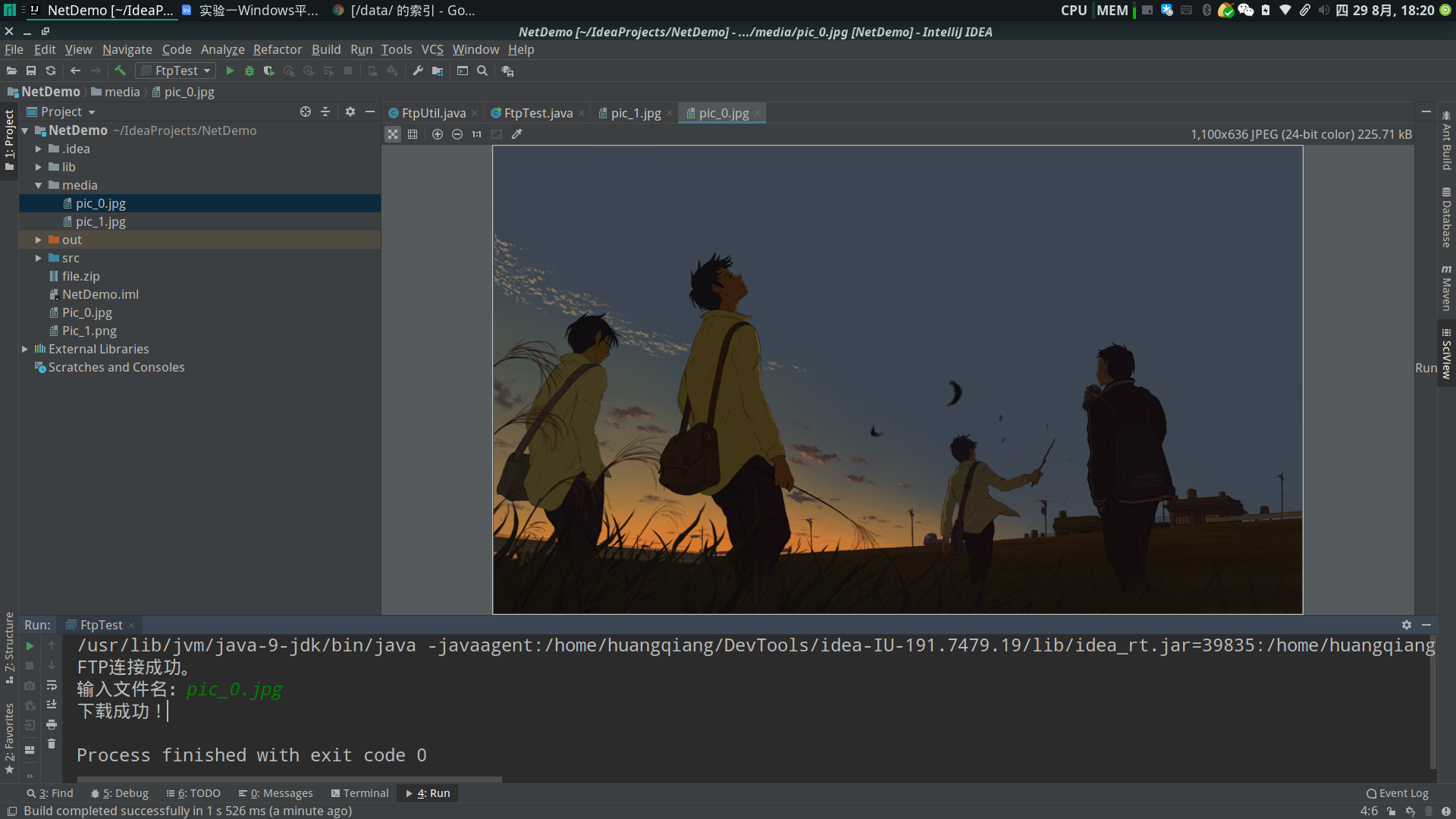


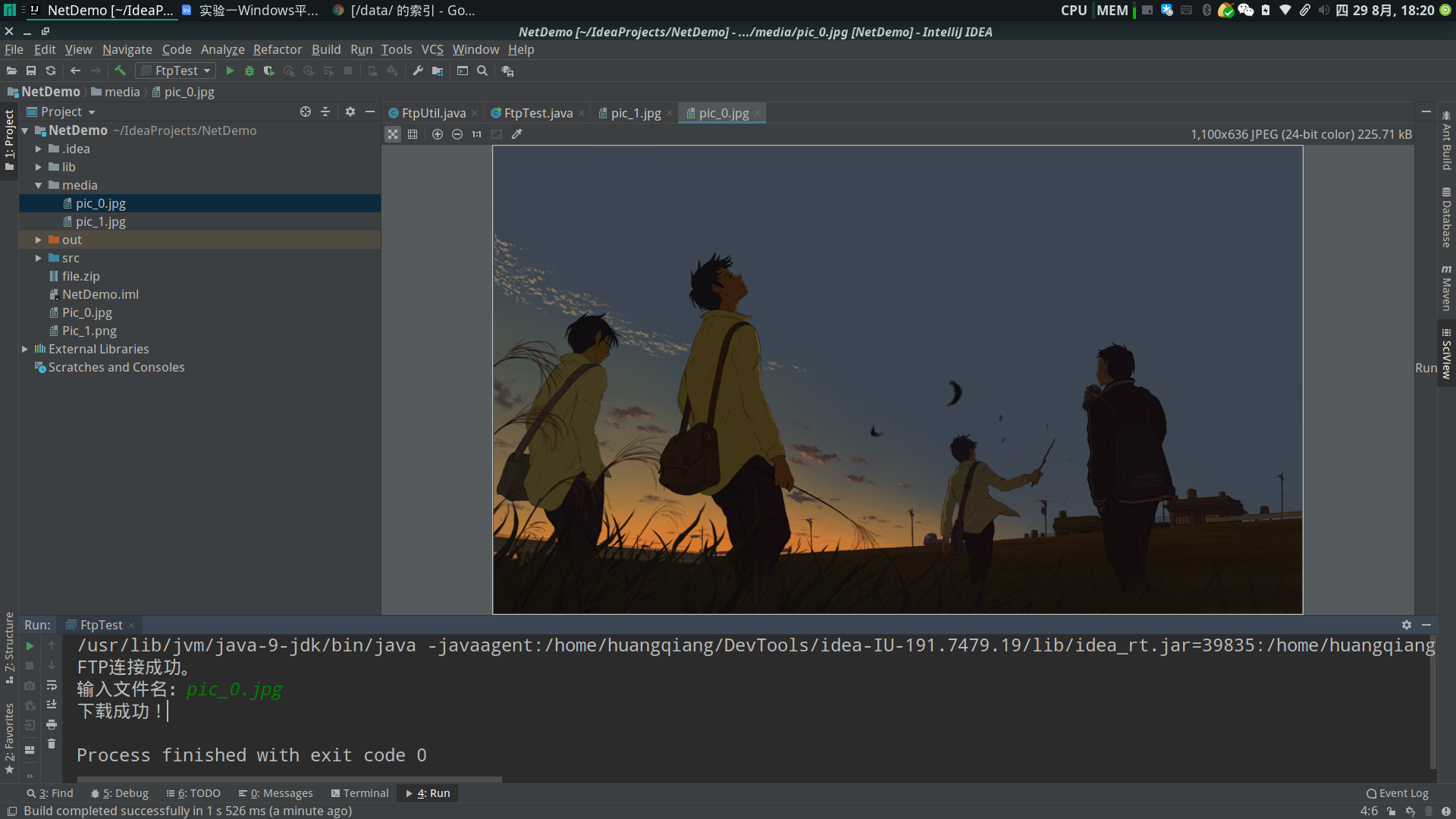
2：上传文件 pic\_1.jpg



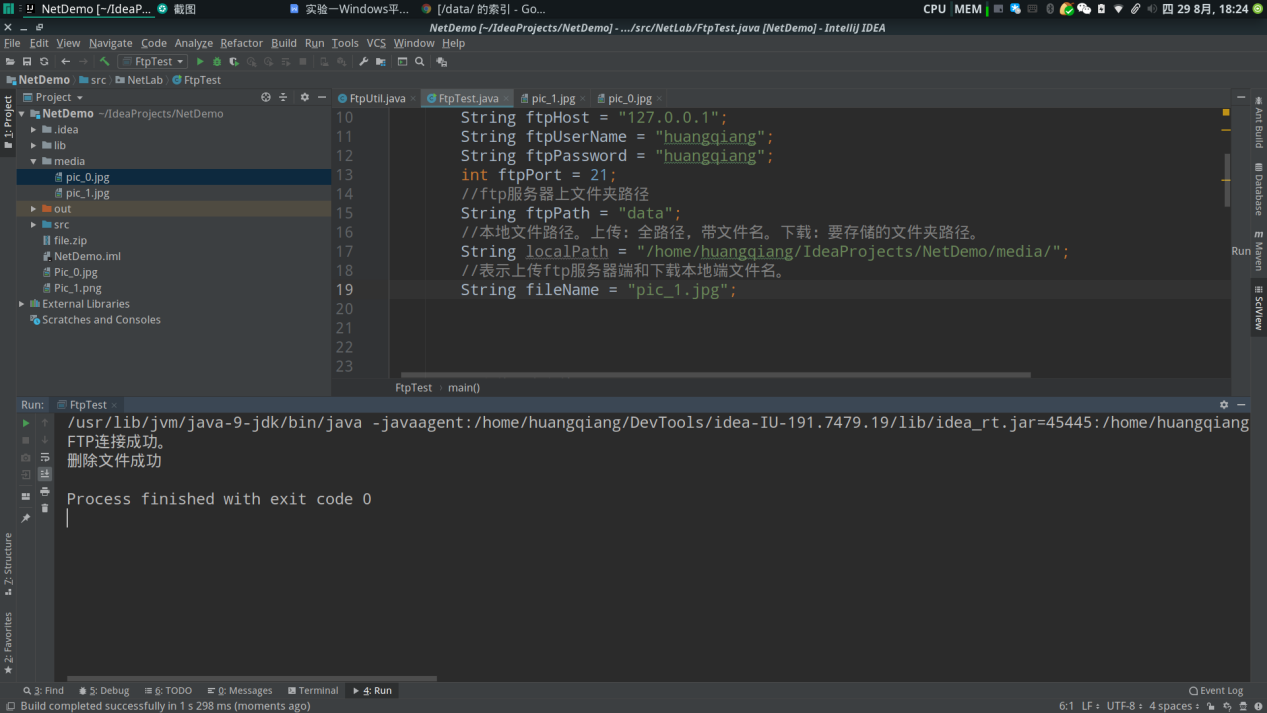


3：下载文件 pic\_0.jpg



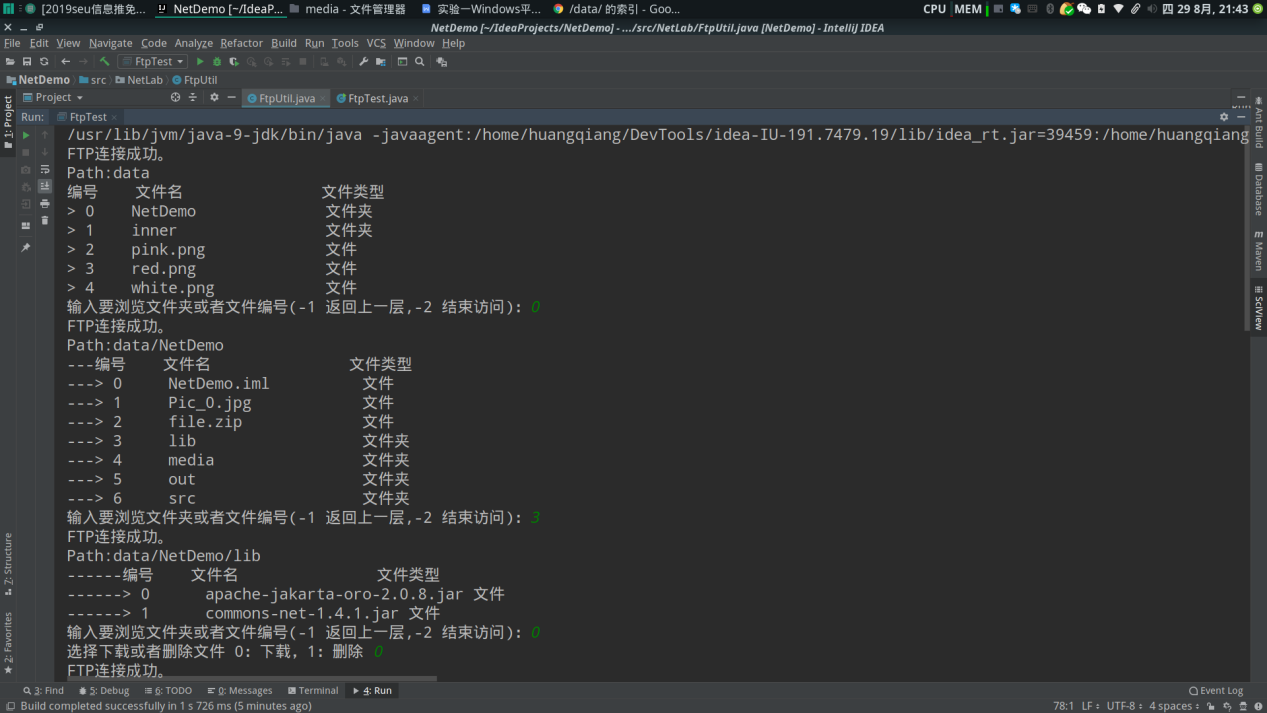


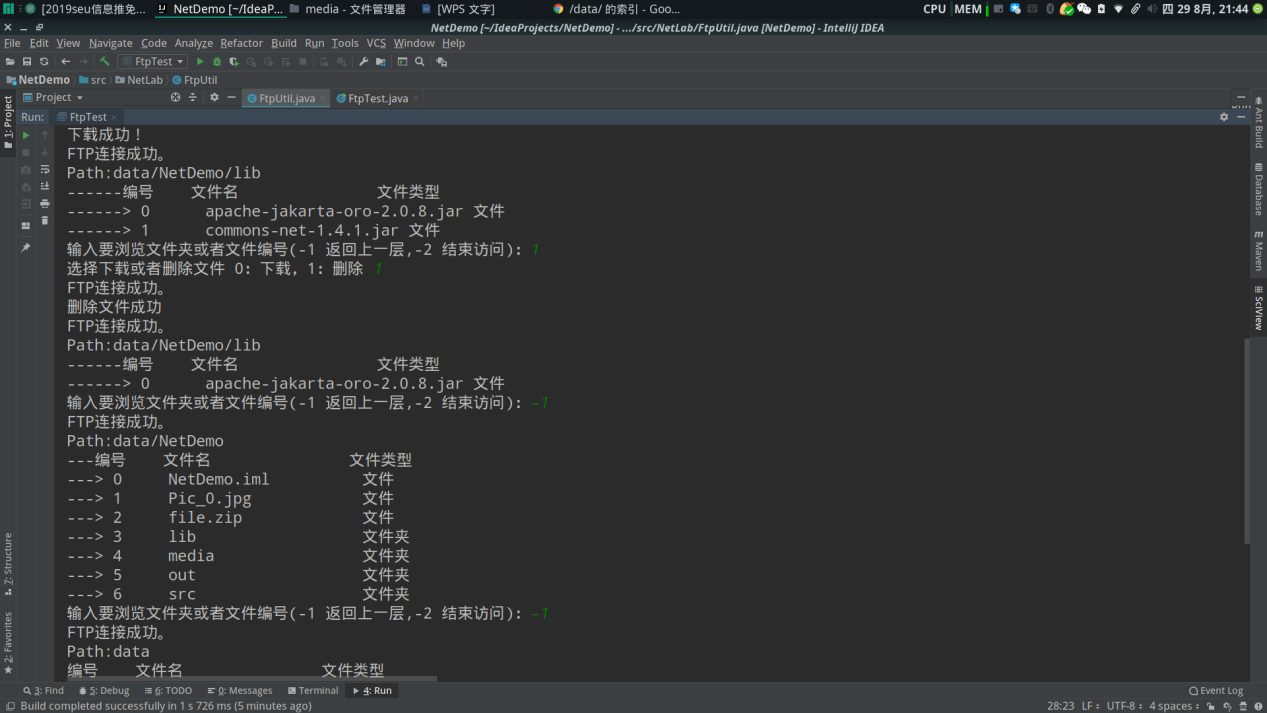
4:删除文件 pic\_1.jpg

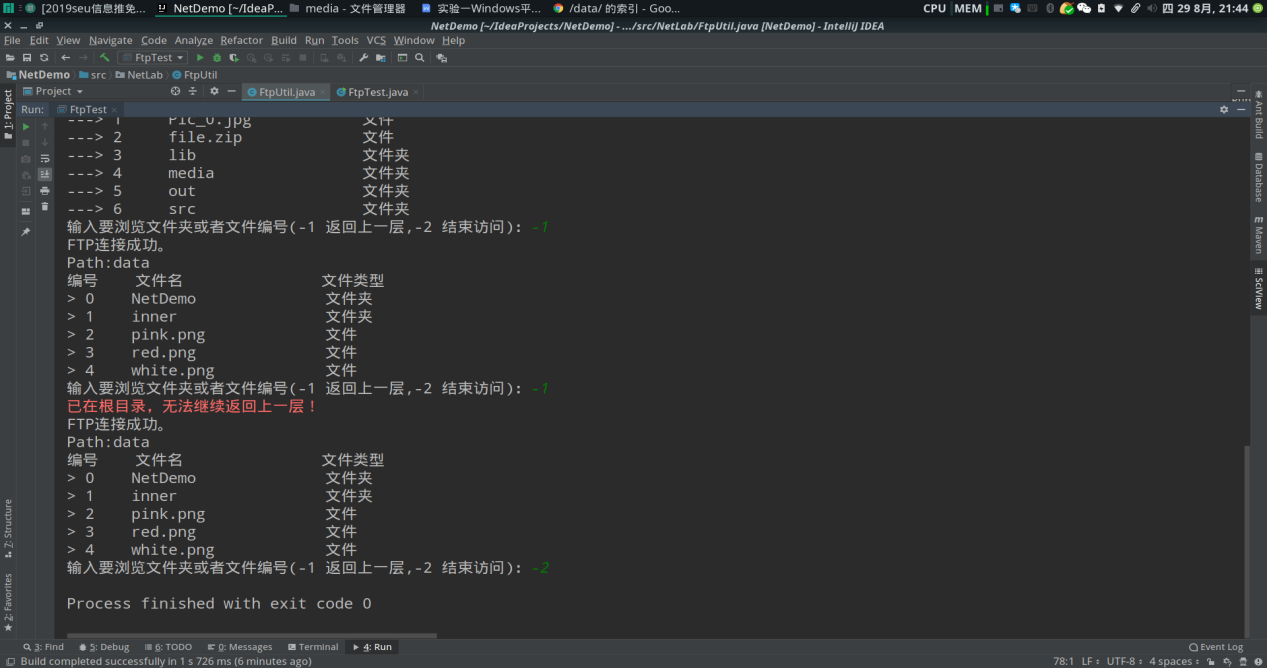




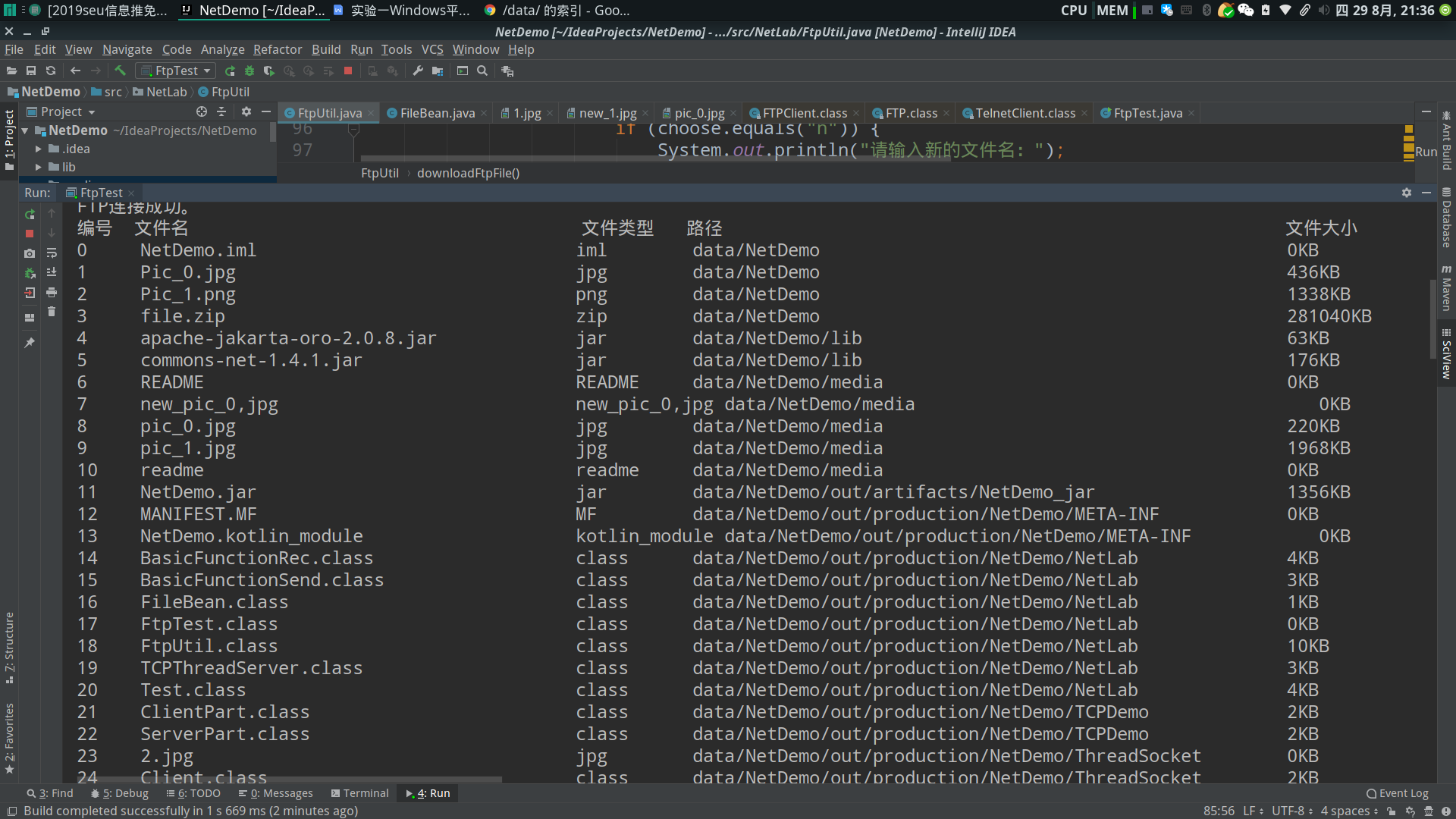
5：浏览目录：控制台内浏览整个FTP文件目录，实现文件的参看及下载、删除。

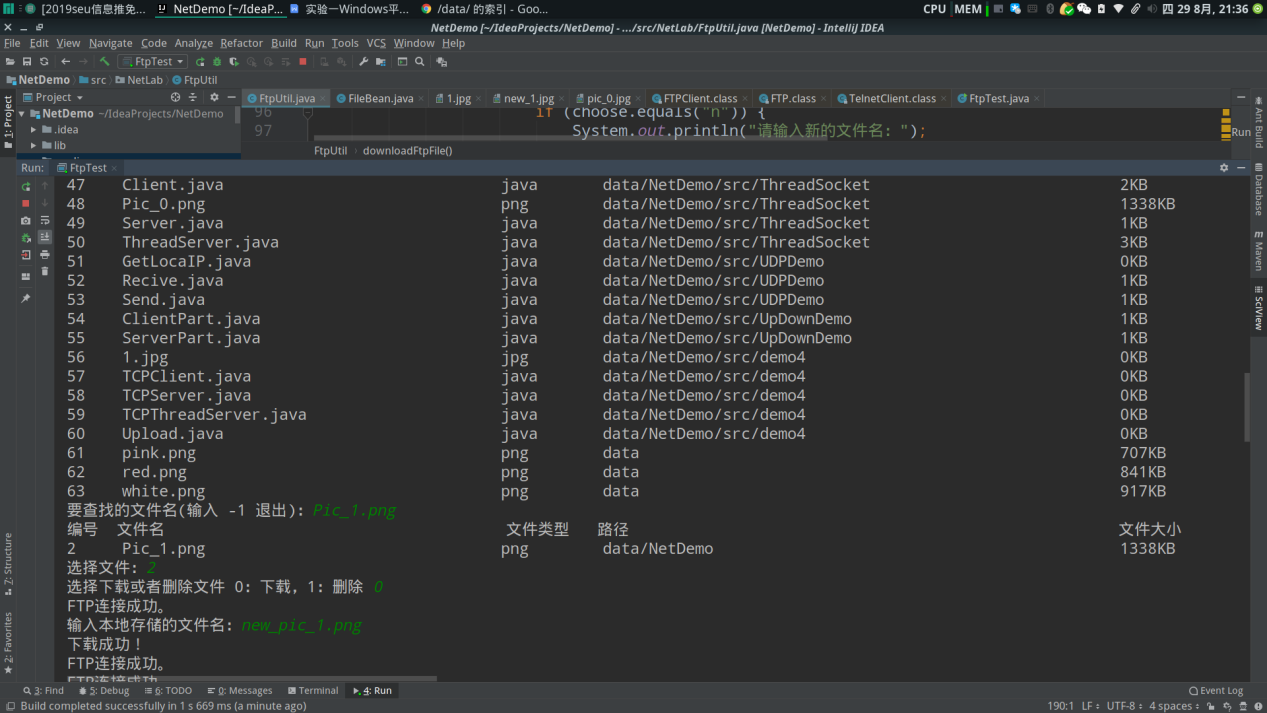


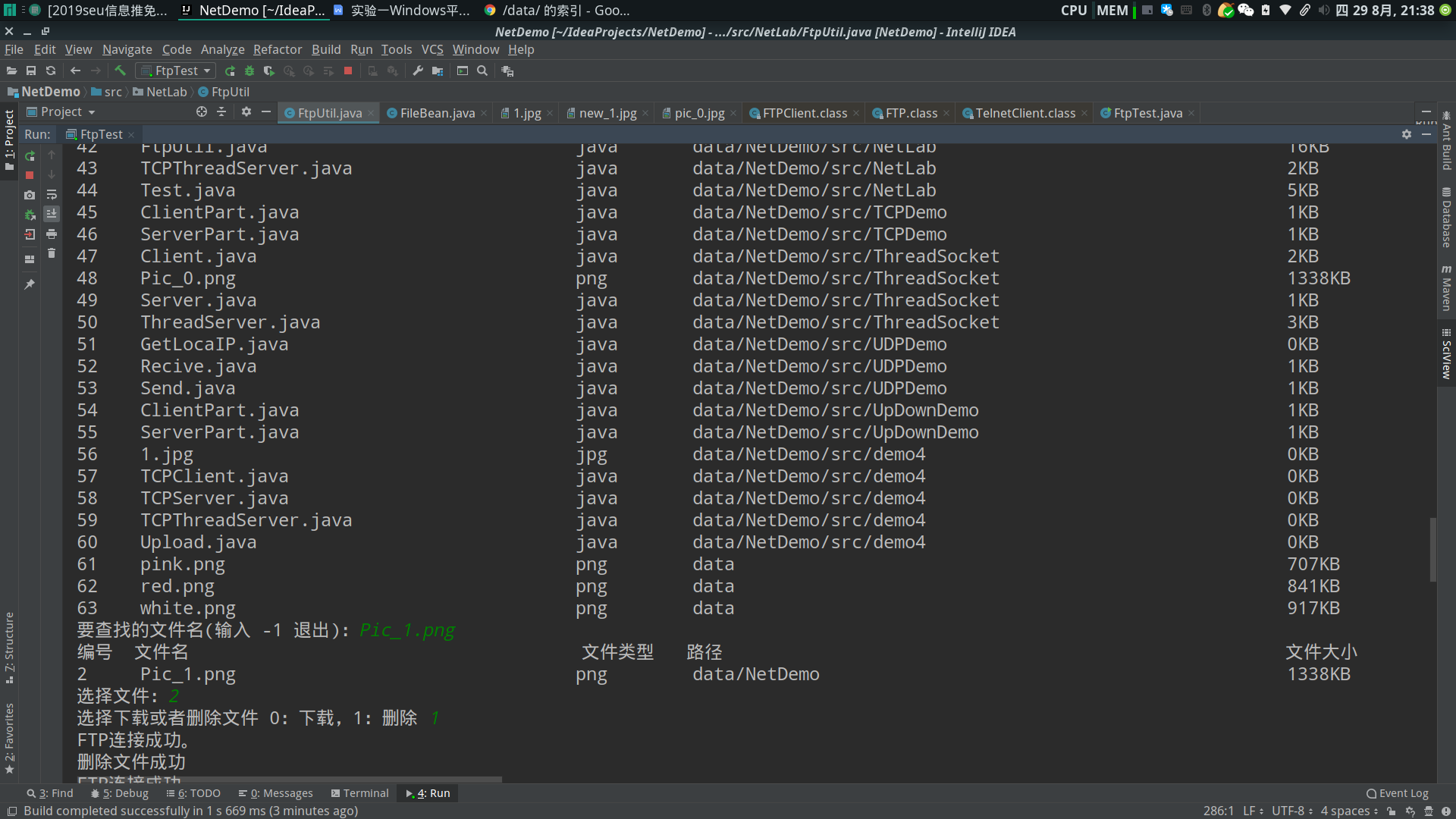




6:查找文件:在控制台内实现文件的查找，对查找到的文件进行删除、下载操作。







FTP 是 File Transfer Protocol（文件传输协议）的英文简称，而中文简称为“文传协议”用于 Internet 上的控制文件的双向传输。同时，它也是一个应用程序（Application）。基于不同的操作系统有不同的 FTP 应用程序，而所有这些应用程序都遵守同一种协议以传输文件。

"下载"文件就是从远程主机拷贝文件至自己的计算机上；"上传"文件就是将文件从自己的计算机中拷贝至远程主机上。用 Internet 语言来说，用户可通过客户机程序向（从）远程主机上传（下载）文件。