## 实验1 读取服务器端文件

**1．相关知识点**

java.net包中的 URL类是对统一资源定位符（Uniform Resource Locator）的抽象，使用URL创建对象的应用程序称作客户端程序，一个URL对象存放着一个具体的资源的引用，表明客户要访问这个URL中的资源，利用URL对象可以获取URL中的资源。URL对象调用InputStream openStream()方法可以返回一个输入流，该输入流指向URL对象所包含的资源。通过该输入流可以将服务器上的资源信息读入到客户端。

**2．实验目的**

学会使用URL对象。

**3．实验要求**

创建一个URL对象，然后让URL对象返回输入流，通过该输入流读取URL所包含的资源文件

**4．运行效果示例**

程序运行效果如图16.1。

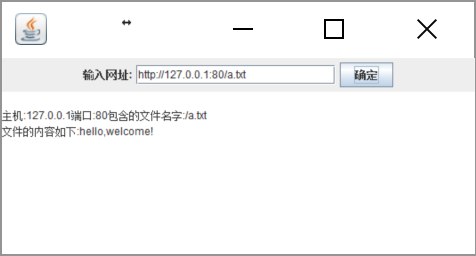


图16.1

**5．程序模板**

请按模板要求，将【代码】替换为Java程序代码。

**ReadFile.java**图16.1 读取文件

import java.awt.\*;

import java.awt.event.\*;

import java.net.\*;

import java.io.\*;

import javax.swing.\*;

public class ReadURLSource {

public static void main(String args[]) {

new NetWin();

}

}

class NetWin extends JFrame implements ActionListener,Runnable {

JButton button;

URL url;

JTextField inputURLText; //输入URL

JTextArea area;

byte b[]=new byte[118];

Thread thread;

NetWin() {

inputURLText=new JTextField(20);

area=new JTextArea(12,12);

button=new JButton("确定");

button.addActionListener(this);

thread=new Thread(this);

JPanel p=new JPanel();

p.add(new JLabel("输入网址:"));

p.add(inputURLText);

p.add(button);

add(area,BorderLayout.CENTER);

add(p,BorderLayout.NORTH);

setBounds(60,60,560,300);

setVisible(true);

validate();

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

public void actionPerformed(ActionEvent e) {

if(!(thread.isAlive()))

thread=new Thread(this);

try{ thread.start();

}

catch(Exception ee) {

inputURLText.setText("我正在读取"+url);

}

}

public void run() {

try { int n=-1;

area.setText(null);

String name=inputURLText.getText().trim();

【代码1】 //使用字符串name创建url对象

String hostName =【代码2】 //url调用getHost()

int urlPortNumber= url.getPort();

String fileName=url.getFile();

InputStream in = 【代码3】 //url调用方法返回一个输入流

area.append("\n主机:"+hostName+"端口:"+urlPortNumber+

"包含的文件名字:"+fileName);

area.append("\n文件的内容如下:");

while((n=in.read(b))!=-1) {

String s=new String(b,0,n);

area.append(s);

}

}

catch(MalformedURLException e1) {

inputURLText.setText(""+e1);

return;

}

catch(IOException e1) {

inputURLText.setText(""+e1);

return;

}

}

}

**6．实验指导**

URL资源的读取可能会引起堵塞，因此，程序需在一个线程中读取URL资源，以免堵塞主线程。

**7．实验后的练习**

public int getDefaultPort()、public String getRef()、public String getProtocol()等方法都是URL对象常用的方法，在模板中让url调用这些方法，并输出这些方法返回的值。

**8．填写实验报告**

实验编号：1601 学生姓名： 实验时间： 教师签字：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验效果评价** | A | B | C | D | E |
| 模板完成情况 |  |  |  |  |  |
| **实验后的练习效果评价** | A | B | C | D | E |
| 练习完成情况 |  |  |  |  |  |
| **总评** |  |  |  |  |  |

## 实验2 会结账的服务器

**1．相关知识点**

网络套接字是基于TCP协议的的有连接通信，套接字连接就是客户端的套接字对象和服务器端的套接字对象通过输入、输出流连接在一起。服务器建立ServerSocket对象，ServerSocket对象负责等待客户端请求建立套接字连接，而客户端建立Socket对象向服务器发出套接字连接请求。

可以使用Socket类不带参数的构造方法

public Socket()

创建一个套接字对象，该对象不请求任何连接。该对象再调用

public void connect(SocketAddress endpoint) throws IOException

请求和参数SocketAddress指定地址的套接字建立连接。为了使用connect方法，可以使用SocketAddress的子类：InetSocketAddress创建一个对象，InetSocketAddress的构造方法是：

public InetSocketAddress(InetAddress addr, int port)

**2．实验目的**

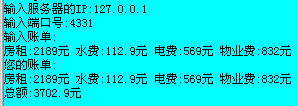
学会使用套接字读取服务器端的对象。

**3．实验要求**

客户端和服务器建立套接字连接后，客户将如下格式的账单发送给服务器：

"房租:2189元 水费:112.9元 电费:569元 物业费:832元"

图16.2（1） 客户端



服务器返回给客户的信息是：

您的账单：

"房租:2189元 水费:112.9元 电费:569元 物业费:832元

总计：3699.9元

**4．运行效果示例**

图16.2（2） 服务器端



程序运行效果如图16.2（1）、16.2（2）。

**5．程序模板**

请按模板要求，将【代码】替换为Java程序代码。

**客户端模板：ClientItem.java**

import java.io.\*;

import java.net.\*;

import java.util.\*;

public class ClientItem {

public static void main(String args[]) {

Scanner scanner = new Scanner(System.in);

Socket clientSocket=null;

DataInputStream inData=null;

DataOutputStream outData=null;

Thread thread ;

Read read=null;

try{ clientSocket=new Socket();

read = new Read();

thread = new Thread(read); //负责读取信息的线程

System.out.print("输入服务器的IP:");

String IP = scanner.nextLine();

System.out.print("输入端口号:");

int port = scanner.nextInt();

String enter=scanner.nextLine(); //消耗回车

if(clientSocket.isConnected()){}

else{

InetAddress address=InetAddress.getByName(IP);

InetSocketAddress socketAddress=new InetSocketAddress(address,port);

clientSocket.connect(socketAddress);

InputStream in=【代码1】//clientSocket调用getInputStream()返回输入流

OutputStream out=【代码2】//clientSocket调用getOutputStream()返回输出流

inData =new DataInputStream(in);

outData = new DataOutputStream(out);

read.setDataInputStream(inData);

read.setDataOutputStream(outData);

thread.start(); //启动负责读信息的线程

}

}

catch(Exception e) {

System.out.println("服务器已断开"+e);

}

}

}

class Read implements Runnable {

Scanner scanner = new Scanner(System.in);

DataInputStream in;

DataOutputStream out;

public void setDataInputStream(DataInputStream in) {

this.in = in;

}

public void setDataOutputStream(DataOutputStream out) {

this.out = out;

}

public void run() {

System.out.println("输入账单:");

String content = scanner.nextLine();

try{ out.writeUTF("账单"+content);

String str = in.readUTF();

System.out.println(str);

str = in.readUTF();

System.out.println(str);

str = in.readUTF();

System.out.println(str);

}

catch(Exception e) {}

}

}

**服务器端模板：ServerItem.java**

import java.io.\*;

import java.net.\*;

import java.util.\*;

public class ServerItem {

public static void main(String args[]) {

ServerSocket server=null;

ServerThread thread;

Socket you=null;

while(true) {

try{ server= 【代码3】//创建在端口4331上负责监听的 ServerSocket对象

}

catch(IOException e1) {

System.out.println("正在监听");

}

try{ System.out.println("正在等待客户");

you= 【代码4】 // server调用accept()返回和客户端相连接的Socket对象

System.out.println("客户的地址:"+you.getInetAddress());

}

catch (IOException e) {

System.out.println(""+e);

}

if(you!=null) {

new ServerThread(you).start();

}

}

}

}

class ServerThread extends Thread {

Socket socket;

DataInputStream in=null;

DataOutputStream out=null;

ServerThread(Socket t) {

socket=t;

try { out=new DataOutputStream(socket.getOutputStream());

in=new DataInputStream(socket.getInputStream());

}

catch (IOException e) {}

}

public void run() {

try{

String item = in.readUTF();

Scanner scanner = new Scanner(item);

scanner.useDelimiter("[^0123456789.]+");

if(item.startsWith("账单")) {

double sum=0;

while(scanner.hasNext()){

try{ double price = scanner.nextDouble();

sum = sum+price;

System.out.println(price);

}

catch(InputMismatchException exp){

String t = scanner.next();

}

}

out.writeUTF("您的账单:");

out.writeUTF(item);

out.writeUTF("总额:"+sum+"元");

}

}

catch(Exception exp){}

}

}

**6．实验指导**

套接字连接中涉及到输入流和输出流操作，客户或服务器读取数据可能会引起堵塞，我们应把读取数据放在一个单独的线程中去进行。另外，服务器端收到一个客户的套接字后，就应该启动一个专门为该客户服务的线程。Socket对象调用public void connect(SocketAddress endpoint) throws IOException方法可以和参数endpoin指定的SocketAddress地址建立套接字连接。

**7．实验后的练习**

改进服务器端程序，使得用户还可以发送如下格式的货品明细给服务器：

货品 宽90厘米 高69厘米 长156厘米

服务器返回给客户的信息是：

货品 宽90厘米 高69厘米 长156厘米

体积：968760立方厘米

**8．填写实验报告**

实验编号：1602 学生姓名： 实验时间： 教师签字：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验效果评价** | A | B | C | D | E |
| 模板完成情况 |  |  |  |  |  |
| **实验后的练习效果评价** | A | B | C | D | E |
| 练习完成情况 |  |  |  |  |  |
| **总评** |  |  |  |  |  |

## 实验3 读取服务器端的窗口

**1．相关知识点（与实验2相同）**

网络套接字是基于TCP协议的的有连接通信，套接字连接就是客户端的套接字对象和服务器端的套接字对象通过输入、输出流连接在一起。服务器建立ServerSocket对象，ServerSocket对象负责等待客户端请求建立套接字连接，而客户端建立Socket对象向服务器发出套接字连接请求。

可以使用Socket类不带参数的构造方法

public Socket()

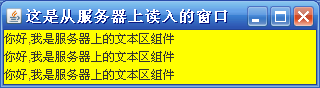
创建一个套接字对象，该对象不请求任何连接。该对象再调用

public void connect(SocketAddress endpoint) throws IOException

请求和参数SocketAddress指定地址的套接字建立连接。为了使用connect方法，可以使用SocketAddress的子类：InetSocketAddress创建一个对象，InetSocketAddress的构造方法是：

public InetSocketAddress(InetAddress addr, int port)

图16.3（1） 客户端



**2．实验目的**

学会使用套接字读取服务器端的对象。

**3．实验要求**

客户端利用套接字连接将服务器端的JFrame对象读取到客户端。首先将服务器端的程序编译通过，并运行起来，等待请求套接字连接。

**4．运行效果示例**

程序运行效果如图16.3（1）、16.3（2）。

图16.3（2） 服务器端



**5．程序模板**

请按模板要求，将【代码】替换为Java程序代码。

**客户端模板：Client.java**

import java.io.\*;

import java.net.\*;

import java.util.\*;

public class Client {

public static void main(String args[]) {

Scanner scanner = new Scanner(System.in);

Socket mysocket=null;

ObjectInputStream inObject=null;

ObjectOutputStream outObject=null;

Thread thread ;

ReadWindow readWindow=null;

try{ mysocket=new Socket();

readWindow = new ReadWindow();

thread = new Thread(readWindow); //负责读取信息的线程

System.out.print("输入服务器的IP:");

String IP = scanner.nextLine();

System.out.print("输入端口号:");

int port = scanner.nextInt();

if(mysocket.isConnected()){}

else{

InetAddress address=InetAddress.getByName(IP);

InetSocketAddress socketAddress=new InetSocketAddress(address,port);

mysocket.connect(socketAddress);

InputStream in=【代码1】 //mysocket调用getInputStream()返回输入流

OutputStream out=【代码2】//mysocket调用getOutputStream()返回输出流

inObject =new ObjectInputStream(in);

outObject = new ObjectOutputStream(out);

readWindow.setObjectInputStream(inObject);

thread.start(); //启动负责读取窗口的线程

}

}

catch(Exception e) {

System.out.println("服务器已断开"+e);

}

}

}

class ReadWindow implements Runnable {

ObjectInputStream in;

public void setObjectInputStream(ObjectInputStream in) {

this.in = in;

}

public void run() {

double result = 0;

while(true) {

try{ javax.swing.JFrame window = (javax.swing.JFrame)in.readObject();

window.setTitle("这是从服务器上读入的窗口");

window.setVisible(true);

window.requestFocusInWindow();//requestFocus();

window.setSize(600,800);

}

catch(Exception e) {

System.out.println("与服务器已断开"+e);

break;

}

}

}

}

**服务器端模板：Server.java**

import java.io.\*;

import java.net.\*;

import java.util.\*;

import java.awt.\*;

import javax.swing.\*;

public class Server {

public static void main(String args[]) {

ServerSocket server=null;

ServerThread thread;

Socket you=null;

while(true) {

try{ server= 【代码3】//创建在端口4331上负责监听的 ServerSocket对象

}

catch(IOException e1) {

System.out.println("正在监听");

}

try{ you= 【代码4】 // server调用accept()返回和客户端相连接的Socket对象

System.out.println("客户的地址:"+you.getInetAddress());

}

catch (IOException e) {

System.out.println("正在等待客户");

}

if(you!=null) {

new ServerThread(you).start();

}

}

}

}

class ServerThread extends Thread {

Socket socket;

ObjectInputStream in=null;

ObjectOutputStream out=null;

JFrame window;

JTextArea text;

ServerThread(Socket t) {

socket=t;

try { out=new ObjectOutputStream(socket.getOutputStream());

in=new ObjectInputStream(socket.getInputStream());

}

catch (IOException e) {}

window =new JFrame();

text = new JTextArea();

for(int i=1;i<=20;i++) {

text.append("你好,我是服务器上的文本区组件\n");

}

text.setBackground(Color.yellow);

window.add(text);

window.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

public void run() {

try{ out.writeObject(window);

}

catch (IOException e) {

System.out.println("客户离开");

}

}

}

**6．实验指导**

使用套接字读取对象时，应将套接字的流和对象流连接在一起。

**7．实验后的练习**

改进程序使得客户端能读入2个窗口。

**8．填写实验报告**

实验编号：1603 学生姓名： 实验时间： 教师签字：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验效果评价** | A | B | C | D | E |
| 模板完成情况 |  |  |  |  |  |
| **实验后的练习效果评价** | A | B | C | D | E |
| 练习完成情况 |  |  |  |  |  |
| **总评** |  |  |  |  |  |

## 实验4 与服务器玩猜数游戏

**1．相关知识点（与实验2相同）**

网络套接字是基于TCP协议的的有连接通信，套接字连接就是客户端的套接字对象和服务器端的套接字对象通过输入、输出流连接在一起。服务器建立ServerSocket对象，ServerSocket对象负责等待客户端请求建立套接字连接，而客户端建立Socket对象向服务器发出套接字连接请求。

可以使用Socket类不带参数的构造方法

public Socket()

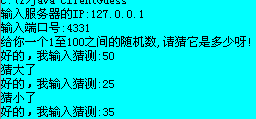
创建一个套接字对象，该对象不请求任何连接。该对象再调用

public void connect(SocketAddress endpoint) throws IOException

请求和参数SocketAddress指定地址的套接字建立连接。为了使用connect方法，可以使用SocketAddress的子类：InetSocketAddress创建一个对象，InetSocketAddress的构造方法是：

public InetSocketAddress(InetAddress addr, int port)

图16.4（1） 客户端



**2．实验目的**

学会使用套接字读取服务器端的对象。

**3．实验要求**

图16.4（2） 服务器端



客户端和服务器建立套接字连接后，服务器向客户发送一个1至100之间的随机数，用户将自己的猜测发送给服务器，服务器向用户发送有关信息：“猜大了”、“猜小了”或“猜对了”。

**4．运行效果示例**

程序运行效果如图16.4（1）、16.4（2）。

**5．程序模板**

请按模板要求，将【代码】替换为Java程序代码。

**客户端模板：ClientGuess.java**

import java.io.\*;

import java.net.\*;

import java.util.\*;

public class ClientGuess {

public static void main(String args[]) {

Scanner scanner = new Scanner(System.in);

Socket mysocket=null;

DataInputStream inData=null;

DataOutputStream outData=null;

Thread thread ;

ReadNumber readNumber=null;

try{ mysocket=new Socket();

readNumber = new ReadNumber();

thread = new Thread(readNumber); //负责读取信息的线程

System.out.print("输入服务器的IP:");

String IP = scanner.nextLine();

System.out.print("输入端口号:");

int port = scanner.nextInt();

if(mysocket.isConnected()){}

else{

InetAddress address=InetAddress.getByName(IP);

InetSocketAddress socketAddress=new InetSocketAddress(address,port);

mysocket.connect(socketAddress);

InputStream in=【代码1】 //mysocket调用getInputStream()返回输入流

OutputStream out=【代码2】//mysocket调用getOutputStream()返回输出流

inData =new DataInputStream(in);

outData = new DataOutputStream(out);

readNumber.setDataInputStream(inData);

readNumber.setDataOutputStream(outData);

thread.start(); //启动负责读取随机数的线程

}

}

catch(Exception e) {

System.out.println("服务器已断开"+e);

}

}

}

class ReadNumber implements Runnable {

Scanner scanner = new Scanner(System.in);

DataInputStream in;

DataOutputStream out;

public void setDataInputStream(DataInputStream in) {

this.in = in;

}

public void setDataOutputStream(DataOutputStream out) {

this.out = out;

}

public void run() {

try {

out.writeUTF("Y");

while(true) {

String str = in.readUTF();

System.out.println(str);

if(!str.startsWith("询问")) {

if(str.startsWith("猜对了"))

continue;

System.out.print("好的，我输入猜测:");

int myGuess = scanner.nextInt();

String enter = scanner.nextLine(); //消耗多余的回车

out.writeInt(myGuess);

}

else {

System.out.print("好的，我输入Y或N:");

String myAnswer = scanner.nextLine();

out.writeUTF(myAnswer);

}

}

}

catch(Exception e) {

System.out.println("与服务器已断开"+e);

return;

}

}

}

**服务器端模板：ServerNumber.java**

import java.io.\*;

import java.net.\*;

import java.util.\*;

public class ServerNumber {

public static void main(String args[]) {

ServerSocket server=null;

ServerThread thread;

Socket you=null;

while(true) {

try{ server= 【代码3】//创建在端口4331上负责监听的 ServerSocket对象

}

catch(IOException e1) {

System.out.println("正在监听");

}

try{ you= 【代码4】 // server调用accept()返回和客户端相连接的Socket对象

System.out.println("客户的地址:"+you.getInetAddress());

}

catch (IOException e) {

System.out.println("正在等待客户");

}

if(you!=null) {

new ServerThread(you).start();

}

}

}

}

class ServerThread extends Thread {

Socket socket;

DataInputStream in=null;

DataOutputStream out=null;

ServerThread(Socket t) {

socket=t;

try { out=new DataOutputStream(socket.getOutputStream());

in=new DataInputStream(socket.getInputStream());

}

catch (IOException e) {}

}

public void run() {

try{

while(true) {

String str = in.readUTF();

boolean boo =str.startsWith("Y")||str.startsWith("y");

if(boo) {

out.writeUTF("给你一个1至100之间的随机数,请猜它是多少呀!");

Random random=new Random();

int realNumber = random.nextInt(100)+1;

handleClientGuess(realNumber);

out.writeUTF("询问:想继续玩输入Y，否则输入N:");

}

else {

return;

}

}

}

catch(Exception exp){}

}

public void handleClientGuess(int realNumber){

while(true) {

try{ int clientGuess = in.readInt();

System.out.println(clientGuess);

if(clientGuess>realNumber)

out.writeUTF("猜大了");

else if(clientGuess<realNumber)

out.writeUTF("猜小了");

else if(clientGuess==realNumber) {

out.writeUTF("猜对了！");

break;

}

}

catch (IOException e) {

System.out.println("客户离开");

return;

}

}

}

}

**6．实验指导**

服务器经常需要根据用户提供不同的信息做出不同的选择，为此，服务器经常需要使用判断语句分析所读入的信息。

**7．实验后的练习**

改进服务器端程序，能向客户发送用户所猜测的次数。

**8．填写实验报告**

实验编号：1604 学生姓名： 实验时间： 教师签字：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验效果评价** | A | B | C | D | E |
| 模板完成情况 |  |  |  |  |  |
| **实验后的练习效果评价** | A | B | C | D | E |
| 练习完成情况 |  |  |  |  |  |
| **总评** |  |  |  |  |  |

## 实验5 传输图像

**1．相关知识点**

基于UDP的通信和基于TCP的通信不同，基于UDP的信息传递更快，但不提供可靠性保证。也就是说，数据在传输时，用户无法知道数据能否正确到达目的地主机，也不能确定数据到达目的地的顺序是否和发送的顺序相同。可以把UDP通信比作邮递信件，我们不能肯定所发的信件就一定能够到达目的地，也不能肯定到达的顺序是发出时的顺序，可能因为某种原因导致后发出的先到达，另外，也不能确定对方收到信就一定会回信。既然UDP是一种不可靠的协议，为什么还要使用它呢？如果要求数据必须绝对准确地到达目的地，显然不能选择UDP协议来通信。但有时候人们需要较快速地传输信息，并能容忍小的错误，就可以考虑使用UDP协议。

基于UDP通信的基本模式是：

图16.5（1） 客户端



（1）将数据封装在数据包中，然后将数据包发往目的地。

（2）接受数据包，然后查看数据包中的内容。

**2．实验目的**

掌握DatagramSocket类的使用。

**3．实验要求**

编写C/S程序，客户端使用DatagramSocket对象将数据包发送到服务器，请求获取服务器端的图像。服务器端将图像文件封装在数据包中，并使用DatagramSocket对象将该数据包发送到客户端。首先将服务器端的程序编译通过，并运行起来，等待客户的请求。

图16.5（2） 服务器端



**4．运行效果示例**

程序运行效果如图16.5（1）、16.5（2）。

**5．程序模板**

请按模板要求，将【代码】替换为Java程序代码。

**客户端模板：ClientImage.java**

import java.net.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.io.\*;

import javax.swing.\*;

class ImageCanvas extends Canvas {

Image image=null;

public ImageCanvas() {

setSize(200,200);

}

public void paint(Graphics g) {

if(image!=null)

g.drawImage(image,0,0,this);

}

public void setImage(Image image) {

this.image=image;

}

}

public class ClientGetImage extends JFrame implements Runnable,ActionListener {

JButton b=new JButton("获取图像");

ImageCanvas canvas;

ClientGetImage() {

super("I am a client");

setSize(320,200);

setVisible(true);

b.addActionListener(this);

add(b,BorderLayout.NORTH);

canvas=new ImageCanvas();

add(canvas,BorderLayout.CENTER);

Thread thread=new Thread(this);

validate();

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

thread.start();

}

public void actionPerformed(ActionEvent event) {

byte b[]="请发图像".trim().getBytes();

try{ InetAddress address=InetAddress.getByName("127.0.0.1");

DatagramPacket data=【代码1】//创建data，该数据包的目标地址和端口分别是

//address和1234，其中的数据为数组b的全部字节

DatagramSocket mailSend=【代码2】//创建负责发送数据的mailSend对象

【代码3】 //mailSend发送数据data

}

catch(Exception e){}

}

public void run() {

DatagramPacket pack=null;

DatagramSocket mailReceive=null;

byte b[]=new byte[8192];

ByteArrayOutputStream out=new ByteArrayOutputStream();

try{ pack=new DatagramPacket(b,b.length);

mailReceive =【代码4】//创建在端口5678负责收取数据包的mailReceive对象

}

catch(Exception e){}

try{ while(true)

{ mailReceive.receive(pack);

String message=new String(pack.getData(),0,pack.getLength());

if(message.startsWith("end")) {

break;

}

out.write(pack.getData(),0,pack.getLength());

}

byte imagebyte[]=out.toByteArray();

out.close();

Toolkit tool=getToolkit();

Image image=tool.createImage(imagebyte);

canvas.setImage(image);

canvas.repaint();

validate();

}

catch(IOException e){}

}

public static void main(String args[]) {

new ClientGetImage();

}

}

**服务器端模板：Server.Imagejava**

import java.net.\*;

import java.io.\*;

public class ServerImage {

public static void main(String args[]) {

DatagramPacket pack=null;

DatagramSocket mailReceive=null;

ServerThread thread;

byte b[]=new byte[8192];

InetAddress address=null;

pack=new DatagramPacket(b,b.length);

while(true) {

try{ mailReceive= new DatagramSocket(1234);

}

catch(IOException e1) {

System.out.println("正在等待");

}

try{ mailReceive.receive(pack);

address=pack.getAddress();

System.out.println("客户的地址:"+address);

}

catch (IOException e) {}

if(address!=null) {

new ServerThread(address).start();

}

}

}

}

class ServerThread extends Thread {

InetAddress address;

DataOutputStream out=null;

DataInputStream in=null;

String s=null;

ServerThread(InetAddress address) {

this.address=address;

}

public void run() {

FileInputStream in;

byte b[]=new byte[8192];

try{ in=new FileInputStream ("a.jpg");

int n=-1;

while((n=in.read(b))!=-1) {

DatagramPacket data=new DatagramPacket(b,n,address,5678);

DatagramSocket mailSend=new DatagramSocket();

mailSend.send(data);

}

in.close();

byte end[]="end".getBytes();

DatagramPacket data=new DatagramPacket(end,end.length,address,5678);

DatagramSocket mailSend=new DatagramSocket();

mailSend.send(data);

}

catch(Exception e){}

}

}

**6．实验指导与检查**

基于UDP的通信和基于TCP的通信不同，基于UDP的信息传递更快，但不提供可靠性保证。也就是说，数据在传输时，用户无法知道数据能否正确到达目的地主机，也不能确定数据到达目的地的顺序是否和发送的顺序相同。基于UDP通信的基本模式是：创建数据包，然后将数据包发往目的地；接受数据包，然后查看数据包中的内容。

**7．实验后的练习**

将上述模板程序改成用户从服务器获取一个文本文件的内容，并显示在客户端。

**8．填写实验报告**

实验编号：1605 学生姓名： 实验时间： 教师签字：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验效果评价** | A | B | C | D | E |
| 模板完成情况 |  |  |  |  |  |
| **实验后的练习效果评价** | A | B | C | D | E |
| 练习完成情况 |  |  |  |  |  |
| **总评** |  |  |  |  |  |