

JAVA程序设计

潘微科

感谢：教材《Java大学实用教程》的作者和其他老师提供PowerPoint讲义等资料！
说明：本课程所使用的所有讲义，都是在以上资料上修改的。

Outline

- 11.1 URL类
- 11.2 读取URL中的资源
- 11.3 显示URL资源中的HTML文件
- 11.4 处理超链接
- 11.5 InetAddress类
- 11.6 套接字Socket
- 11.7 使用多线程处理套接字连接
- 11.8 UDP数据报
- 11.9 广播数据报
- 11.10 Java远程调用

11.1 URL类

- 一个URL对象通常包含最基本的三部分信息：协议、地址、资源
 - 常用的http、ftp、file**协议**都是JVM支持的协议
 - **地址**必须是能连接的有效的IP地址或域名（host name）
 - **资源**可以是主机上的任何一个**文件**
- IP: Internet Protocol
- DNS: Domain Name Servers，用来把host name转换成IP地址

11.1 URL类

- 1.URL的构造方法

public **URL(String spec)** throws MalformedURLException

- 该构造方法使用字符串初始化一个URL对象，例如

```
try
{
    url=new URL("http://yahoo.com.cn");
}
catch(MalformedURLException e)
{
    System.out.println("Bad URL:"+url);
}
```

- 该URL对象使用的**协议**是http协议，即用户按这种协议和指定的服务器通信；该URL对象包含的**地址**是yahoo.com.cn；所包含的**资源**是**默认**的资源（主页）。

11.1 URL类

- public **URL(String protocol, String host, String file)** throws `MalformedURLException`
- 该构造方法创建的URL对象的**协议**、**地址**和**资源**分别由参数protocol、host和file指定。

Outline

- 11.1 URL类
- 11.2 读取URL中的资源
- 11.3 显示URL资源中的HTML文件
- 11.4 处理超链接
- 11.5 InetAddress类
- 11.6 套接字Socket
- 11.7 使用多线程处理套接字连接
- 11.8 UDP数据报
- 11.9 广播数据报
- 11.10 Java远程调用

11.2 读取URL中的资源

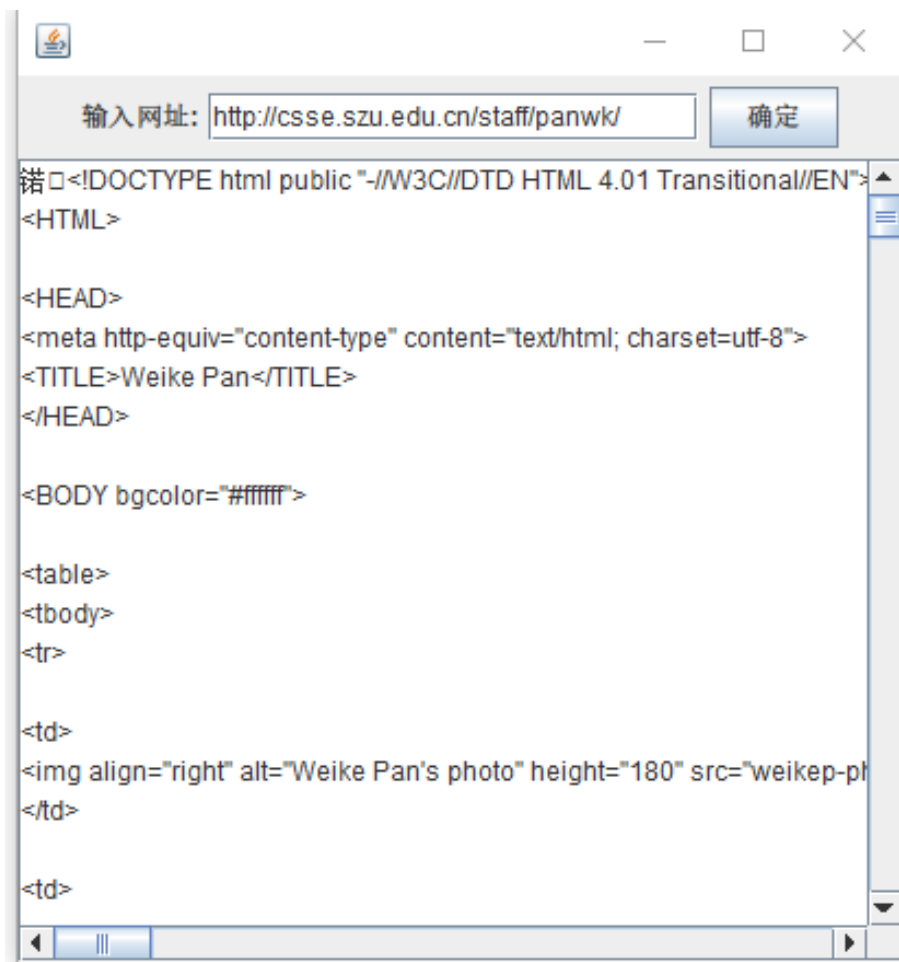
- URL对象调用

`InputStream openStream()`

- 该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。
- 下面的例子Example11_1.java在一个文本框中输入网址，然后点击确定按钮读取服务器上的资源。由于网络速度或其它因素，URL资源的读取可能会引起堵塞，因此，程序需要在`一个线程中`读取URL资源，以免堵塞主线程。

11.2 读取URL中的资源

- 【例子1】
- Example11_1.java




```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
public class Example11_1
{
    public static void main(String args[])
    {
        new NetWin();
    }
}
```

```
class NetWin extends JFrame implements ActionListener, Runnable
{
    JButton button;
    URL url;
    JTextField text;
    JTextArea area;

    byte b[] = new byte[118];
    Thread thread;

    NetWin()
    {
        text = new JTextField(20);
        area = new JTextArea(12,12);
        add(new JScrollPane(area), BorderLayout.CENTER);
        button = new JButton("确定");
        button.addActionListener(this);

        JPanel p = new JPanel();
        p.add(new JLabel("输入网址:"));
        p.add(text);
        p.add(button);


        add(p, BorderLayout.NORTH);

        setBounds(60, 60, 360, 300);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        thread = new Thread(this); // 创建线程但没有启动
    }
}
```

```
public void actionPerformed(ActionEvent e)
{
    if( !(thread.isAlive()) )
    {
        thread = new Thread(this);
    }

    try
    {
        thread.start();
    }
    catch(Exception ee){}
}
```



```
public void run()
{
    try
    {
        → url = new URL(text.getText().trim()); // !

        int n = -1;

        area.setText(null);

        → InputStream in = url.openStream(); // !

        while( (n=in.read(b)) != -1 )
        {
            String s = new String(b,0,n);
            area.append(s);
        }
    }
    catch(MalformedURLException e1)
    {
        text.setText( "" + e1 );
        return;
    }
    catch(IOException e1)
    {
        text.setText( "" + e1 );
        return;
    }
}
}
```

Outline

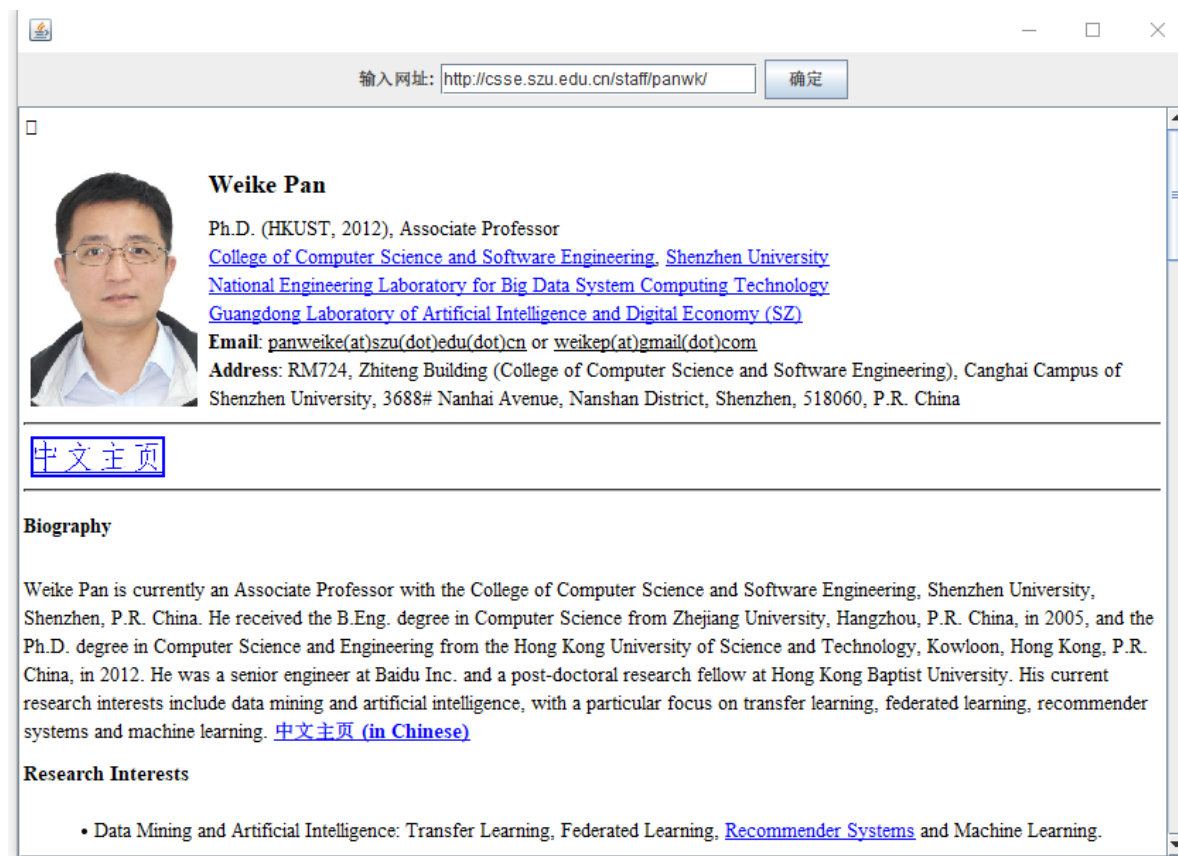
- 11.1 URL类
- 11.2 读取URL中的资源
- 11.3 显示URL资源中的HTML文件
- 11.4 处理超链接
- 11.5 InetAddress类
- 11.6 套接字Socket
- 11.7 使用多线程处理套接字连接
- 11.8 UDP数据报
- 11.9 广播数据报
- 11.10 Java远程调用

11.3 显示URL资源中的HTML文件

- javax.swing包中的**JEditorPane**类可以**解释执行**html文件，也就是说，如果你把html文件读入到JEditorPane，该html文件就会被解释执行，显示在JEditorPane中，这样就可以看到网页的运行效果了。
- JEditorPane类的构造方法：
public JEditorPane()
public JEditorPane(URL initialPage) throws IOException
public JEditorPane(String url) throws IOException
- 后两个构造方法使用参数initialPage或url指定**最初的**URL中的**资源**。
- JEditorPane对象调用public void **setPage**(URL page) throws IOException可以显示**新的**URL中的**资源**。

11.3 显示URL资源中的HTML文件

- 【例子2】
- Example11_2.java



```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;

public class Example11_2
{
    public static void main(String args[])
    {
        new WinOne();
    }
}
```



```
class WinOne extends JFrame implements ActionListener, Runnable
{
    JButton button;
    URL url;
    JTextField text;
    JEditorPane editPane;
    Thread thread;

    public WinOne()
    {
        text = new JTextField(20);
        editPane = new JEditorPane();
        editPane.setEditable(false);
        button = new JButton("确定");
        button.addActionListener(this);

        JPanel p = new JPanel();
        p.add(new JLabel("输入网址:"));
        p.add(text);
        p.add(button);

        Container con = getContentPane();
        con.add(new JScrollPane(editPane), BorderLayout.CENTER);
        con.add(p, BorderLayout.NORTH);

        setBounds(60, 60, 360, 300);
        setVisible(true); validate();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        thread = new Thread(this); // 创建线程但没有启动
    }
}
```

```
public void actionPerformed(ActionEvent e)
{
    if(!(thread.isAlive()))
    {
        thread=new Thread(this);
    }
    try
    {
        thread.start();
    }
    catch(Exception ee)
    {
        text.setText("我正在读取"+url);
    }
}
```

```
public void run()
{
    try
    {
        url = new URL(text.getText().trim());

        int n=-1;

        editPane.setText(null);

        → editPane.setPage(url); // !
    }
    catch(MalformedURLException e1)
    {
        text.setText("" + e1);
        return;
    }
    catch(IOException e1)
    {
        text.setText("" + e1);
        return;
    }
}
```

Outline

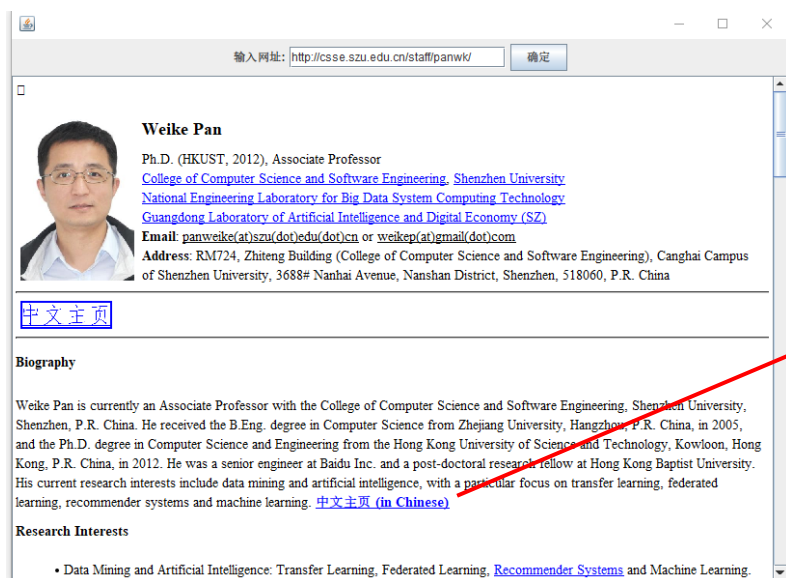
- 11.1 URL类
- 11.2 读取URL中的资源
- 11.3 显示URL资源中的HTML文件
- **11.4 处理超链接**
- 11.5 InetAddress类
- 11.6 套接字Socket
- 11.7 使用多线程处理套接字连接
- 11.8 UDP数据报
- 11.9 广播数据报
- 11.10 Java远程调用

11.4 处理超链接

- 程序可以通过处理HyperlinkEvent事件，来显示新的URL中的资源。
- JEditorPane对象调用addHyperlinkListener(HyperlinkListener listener)获得监视器。
- 监视器需要实现HyperlinkListener接口，该接口中的方法是
void **hyperlinkUpdate**(HyperlinkEvent e)
- 在下面的例子Example11_3.java中，当单击超链接时，JEditorPane对象将显示超链接所指向的网页。

11.4 处理超链接

- 【例子3】
- Example11_3.java



```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import javax.swing.event.*;

public class Example11_3
{
    public static void main(String args[])
    {
        new WinTwo ();
    }
}
```

```
class WinTwo extends JFrame implements ActionListener,Runnable
{
    JButton button;
    URL url;
    JTextField text;
    JEditorPane editPane;
    Thread thread;
    public WinTwo ()
    {
        text = new JTextField(20);
        editPane = new JEditorPane();
        editPane.setEditable(false);
        button = new JButton("确定");
        button.addActionListener(this);


        JPanel p = new JPanel();
        p.add(new JLabel("输入网址:"));
        p.add(text);
        p.add(button);

        Container con = getContentPane();
        con.add(new JScrollPane(editPane),BorderLayout.CENTER);
        con.add(p,BorderLayout.NORTH);

        setBounds(60,60,360,300);
        setVisible(true); validate();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        thread = new Thread(this);
```



```
editPane.addHyperlinkListener(new HyperlinkListener()
{
    public void hyperlinkUpdate(HyperlinkEvent e)
    {
        if(e.getEventType()==HyperlinkEvent.EventType.ACTIVATED)
        {
            try
            {
editPane.setPage(e.getURL());
            }
            catch(IOException e1)
            {
                editPane.setText(""+e1);
            }
        }
    }
});
}
```

```
public void actionPerformed(ActionEvent e)
{
    if(!(thread.isAlive()))
    {
        thread = new Thread(this);
    }

    try
    {
        thread.start();
    }
    catch(Exception ee)
    {
        text.setText("我正在读取"+url);
    }
}
```

```
public void run()
{
    try
    {
        url = new URL(text.getText().trim());

        int n = -1;

        editPane.setText(null);

        editPane.setPage(url);
    }
    catch(MalformedURLException e1)
    {
        text.setText(""+e1);
        return;
    }
    catch(IOException e1)
    {
        text.setText(""+e1);
        return;
    }
}
```

Outline

- 11.1 URL类
- 11.2 读取URL中的资源
- 11.3 显示URL资源中的HTML文件
- 11.4 处理超链接
- 11.5 InetAddress类
- 11.6 套接字Socket
- 11.7 使用多线程处理套接字连接
- 11.8 UDP数据报
- 11.9 广播数据报
- 11.10 Java远程调用

11.5 InetAddress类

- InetAddress类的对象含有一个Internet主机地址的**域名**和**IP地址**，例如：
www.sina.com.cn/157.255.224.244
- 域名容易记忆，当你在连接网络时输入一个主机地址的域名后，域名服务器（Domain Name Servers, DNS）负责将域名转化为IP地址，这样我们才能和主机建立连接。

11.5 InetAddress类

- 1.获取Internet上主机的地址
- InetAddress类的静态方法:
 `getByName(String s);`
- 获得一个InetAddress对象，该对象含有主机地址的域名和IP地址，该对象用如下格式表示它包含的信息：

`www.sina.com.cn/157.255.224.244`

11.5 InetAddress类

- 【例子4】
- Exampel11_4.java

```
www.sina.com.cn/157.255.224.244  
www.sina.com.cn  
157.255.224.244
```

```
import java.net.*;  
public class Example11_4  
{  
    public static void main(String args[])  
    {  
        try  
        {  
            InetAddress address_1 = InetAddress.getByName("www.sina.com.cn");  
            System.out.println( address_1.toString() );  
            System.out.println( address_1.getHostName() );  
            System.out.println( address_1.getHostAddress() );  
        }  
        catch(UnknownHostException e)  
        {  
            System.out.println("无法找到www.sina.com.cn");  
        }  
    }  
}
```

InetAddress类没有构造方法，
可以通过**静态方法**获得对象



Outline

- 11.1 URL类
- 11.2 读取URL中的资源
- 11.3 显示URL资源中的HTML文件
- 11.4 处理超链接
- 11.5 InetAddress类
- **11.6 套接字Socket**
 - 11.7 使用多线程处理套接字连接
 - 11.8 UDP数据报
 - 11.9 广播数据报
 - 11.10 Java远程调用

11.6 套接字Socket

- 端口号与IP地址的组合得到一个**网络套接字**。
- **端口号**被规定为一个16位的整数0~65535。其中，0~1023被预先定义的服务通信占用（如telnet占用端口23，http占用端口80等）。除非我们需要访问这些特定的服务，否则，就应该使用1024~65535这些端口中的某一个进行通信，以免发生**端口冲突**。

11.6 套接字Socket

- 1.套接字连接
- 所谓套接字连接就是**客户端**的**套接字对象**和**服务器端**的**套接字对象**通过**输入流**和**输出流**连接在一起，现在我们分三个步骤来说明套接字连接的基本模式。
- (1) 服务器建立ServerSocket对象
- ServerSocket对象负责等待客户端的请求，进而建立套接字连接。
- ServerSocket的构造方法是：ServerSocket(int port)

```
try
{
    ServerSocket waitSocketConnection = new ServerSocket(1880);
}
catch(IOException e){}
```

11.6 套接字Socket

- 当服务器端的ServerSocket对象waitSocketConnection建立后，就可以使用方法accept()**接收**客户端的套接字连接请求，代码如下所示：

```
Socket socketAtServer = waitSocketConnection.accept();
```

- 所谓“接收”客户的套接字请求，就是accept()方法会返回一个**Socket对象**（即socketAtServer），称作服务器端的**套接字对象**。

11.6 套接字Socket

- (2) 客户端创建Socket对象
- 客户端程序可以使用Socket类创建对象，Socket的构造方法是：
Socket(String host, int port)

```
Socket socketAtClient = new Socket(localhost, 1880);
```
- 也可以使用Socket类不带参数的构造方法public Socket()
- 该对象再调用public void connect(InetSocketAddress endpoint) throws IOException **请求**和参数InetSocketAddress指定地址的套接字建立连接。
- 客户端建立**Socket对象**（即socketAtClient）的过程就是向服务器发出套接字连接请求，如果服务器端相应的端口上有套接字对象正在使用accept()方法等待客户端，那么双方的**套接字对象**（即socketAtClient和socketAtServer）就都诞生了。

11.6 套接字Socket

- (3) 流连接
- 客户端和服务端端的**Socket对象**诞生以后，还必须进行输入流、输出流的连接。
- 服务器端的这个Socket对象（即socketAtServer）使用方法getOutputStream()获得的**输出流**，将指向客户端Socket对象（即socketAtClient）使用方法getInputStream()获得的那个**输入流**。
- 服务器端的这个Socket对象（即socketAtServer）使用方法getInputStream()获得的**输入流**，将指向客户端Socket对象（即socketAtClient）使用方法getOutputStream()获得的那个**输出流**。
- 因此，当服务器向这个输出流写入信息时，客户端通过相应的输入流就能读取，反之亦然。

11.6 套接字Socket

- 【例子5】
- Example11_5
 - Client.java
 - Server.java
- 先运行服务器端程序Server.java
- 再运行客户端程序Client.java

```
C:\Windows\system32\cmd.exe - java Server

E:\WorkspaceEclipse\Example11_5\bin>java Server
Server received: 1
Server received: 2
Server received: 4
Server received: 8
Server received: 16
Server received: 32
Server received: 64
Server received: 128
Server received: 256
Server received: 512
Server received: 1024
Server received: 2048
Server received: 4096
Server received: 8192
Server received: 16384
Server received: 32768
Server received: 65536
Server received: 131072
Server received: 262144
Server received: 524288
Server received: 1048576
Server received: 2097152
```

```
C:\Windows\system32\cmd.exe - java Client

E:\WorkspaceEclipse\Example11_5\bin>java Client
Client received: 2
Client received: 4
Client received: 8
Client received: 16
Client received: 32
Client received: 64
Client received: 128
Client received: 256
Client received: 512
Client received: 1024
Client received: 2048
Client received: 4096
Client received: 8192
Client received: 16384
Client received: 32768
Client received: 65536
Client received: 131072
Client received: 262144
Client received: 524288
Client received: 1048576
Client received: 2097152
Client received: 4194304
```

```
import java.io.*;
import java.net.*;
public class Server
{
    public static void main(String args[])
    {
        ServerSocket server = null;
        Socket socketAtServer = null;
        DataOutputStream out = null;
        DataInputStream in = null;

        try
        {
            server = new ServerSocket(4333);
        }
        catch(IOException e1)
        {
            System.out.println("ERRO:"+e1);
        }
    }
}
```



```
try
```

```
{
```

```
→ socketAtServer = server.accept();
```

```
in = new DataInputStream(socketAtServer.getInputStream());
```

```
out = new DataOutputStream(socketAtServer.getOutputStream());
```

```
while(true)
```

```
{
```

```
    int m = 0;
```

```
    m = in.readInt();
```

```
    out.writeInt(m*2);
```

```
    System.out.println("Server received: " + m);
```

```
    Thread.sleep(500);
```

```
}
```

```
}
```

```
catch(IOException e)
```

```
{
```

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

```
}
```

```
catch(InterruptedException e){}
```

```
}
```

```
}
```



```
import java.io.*;
import java.net.*;
public class Client
{
    public static void main(String args[])
    {
        Socket socketAtClient;
        DataInputStream in = null;
        DataOutputStream out = null;
        try
        {
            socketAtClient = new Socket("localhost", 4333);
            in = new DataInputStream(socketAtClient.getInputStream());
            out = new DataOutputStream(socketAtClient.getOutputStream());
            out.writeInt(1);
            while(true)
            {
                int m2 = 0;
                m2 = in.readInt();
                out.writeInt(m2);
                System.out.println("Client received: " + m2);
                Thread.sleep(500);
            }
        }
        catch(IOException e)
        {
            System.out.println("Unable to connect to the server");
        }
        catch(InterruptedException e){}
    }
}
```

Outline

- 11.1 URL类
- 11.2 读取URL中的资源
- 11.3 显示URL资源中的HTML文件
- 11.4 处理超链接
- 11.5 InetAddress类
- 11.6 套接字Socket
- **11.7 使用多线程处理套接字连接**
- 11.8 UDP数据报
- 11.9 广播数据报
- 11.10 Java远程调用

11.7 使用多线程处理套接字连接

- 在下面的例子6中，客户输入一个一元二次方程的系数并发送给服务器，服务器把计算出的方程的实根返回给客户端。因此，你可以将计算量大的工作放在服务器端，客户端负责计算量小的工作，实现客户端-服务器端的交互计算，进而完成某项任务。

11.7 使用多线程处理套接字连接

- 【例子6】
- Example11_6
 - MutiServer.java
 - ClientFrame.java

客户的地址: /127.0.0.1
正在监听

The screenshot shows a Java Swing window titled "ClientFrame.java" with standard Windows window controls (minimize, maximize, close). The window contains a "连接服务器" (Connect to Server) button on the left. To the right of this button are three input fields for a quadratic equation: "输入2次项系数" (Input 2nd degree coefficient) with value "1", "输入1次项系数" (Input 1st degree coefficient) with value "2", and "输入常数项" (Input constant term) with value "-3". Below these inputs is a "求方程的根" (Solve for the roots of the equation) button. To the right of this button is a text area displaying the results: "两个根:" (Two roots:), "1.0", and "-3.0".

```
import java.io.*;
import java.net.*;
import java.util.*;

public class MutiServer
{
    public static void main(String args[])
    {
        ServerSocket server = null;
        // ServerThread thread;
        Socket socketAtServer = null;
    }
}
```

```
while(true) // !!
{
    try
    {
        server = new ServerSocket(4332);
    }
    catch(IOException e1)
    {
        System.out.println("正在监听");    //ServerSocket对象不能重复创建
    }
    try
    {
        socketAtServer = server.accept();
        System.out.println("客户的地址:" + socketAtServer.getInetAddress());
    }
    catch (IOException e)
    {
        System.out.println("正在等待客户");
    }
    if(socketAtServer!=null)
    {
        → new ServerThread(socketAtServer).start(); //为每个客户启动一个专门的线程
    }
    else
    {
        continue;
    }
}
}
```

```
→ class ServerThread extends Thread
{
    Socket socket;
    DataOutputStream out = null;
    DataInputStream in = null;
    String s = null;

    ServerThread(Socket t)
    {
        socket = t;
        try
        {
            in = new DataInputStream(socket.getInputStream());
            out = new DataOutputStream(socket.getOutputStream());
        }
        catch (IOException e){}
    }
}
```

```

public void run()
{
    while(true)
    {
        double a=0,b=0,c=0,root1=0,root2=0;
        try
        {
            a = in.readDouble(); //堵塞状态，除非读取到信息
            b = in.readDouble();
            c = in.readDouble();

            double disk=b*b-4*a*c;
            root1 = (-b+Math.sqrt(disk))/(2*a);
            root2 = (-b-Math.sqrt(disk))/(2*a);

            out.writeDouble(root1);
            out.writeDouble(root2);
        }
        catch (IOException e)
        {
            System.out.println("客户离开");
            break;
        }
    }
}

```



```
import java.net.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ClientFrame extends JFrame implements Runnable, ActionListener
{
    JButton connection,computer;
    JTextField inputA,inputB,inputC;
    JTextArea showResult;
    Socket socket = null;
    DataInputStream in = null;
    DataOutputStream out = null;
Thread thread;
}
```

```

public ClientFrame()
{
    socket = new Socket(); //待连接的套接字
    connection = new JButton("连接服务器");
    connection.addActionListener(this);

    computer = new JButton("求方程的根");
    computer.setEnabled(false); //没有和服务器连接之前，该按钮不可用
    computer.addActionListener(this);

    inputA = new JTextField("0",12);
    inputB = new JTextField("0",12);
    inputC = new JTextField("0",12);

    Box boxV1 = Box.createVerticalBox();
    boxV1.add(new JLabel("输入2次项系数"));
    boxV1.add(new JLabel("输入1次项系数"));
    boxV1.add(new JLabel("输入常数项"));
    Box boxV2 = Box.createVerticalBox();
    boxV2.add(inputA);
    boxV2.add(inputB);
    boxV2.add(inputC);
    Box baseBox = Box.createHorizontalBox();
    baseBox.add(boxV1);
    baseBox.add(boxV2);

    showResult = new JTextArea(8,18);

```

```
showResult = new JTextArea(8,18);

// ---
Container con = getContentPane();
con.setLayout(new FlowLayout());
con.add(connection);
con.add(baseBox);
con.add(computer);
con.add(new JScrollPane(showResult));

// ---
setBounds(100,100,360,310);
setVisible(true);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// ---
thread = new Thread(this);
}
```

```
public void run()
{
    while(true)
    {
        try
        {
            double root1 = in.readDouble(); //堵塞状态，除非读取到信息
            double root2 = in.readDouble();
            showResult.append("\n两个根:\n"+root1+"\n"+root2);
            showResult.setCaretPosition((showResult.getText()).length());
        }
        catch(IOException e)
        {
            showResult.setText("与服务器已断开");
            computer.setEnabled(false);
            break;
        }
    }
}
```

```

public void actionPerformed(ActionEvent e)
{
    if(e.getSource()==connection)
    {
        try
        {
            if(socket.isConnected())
            {
            }
            else
            {
                InetAddress address = InetAddress.getByName("127.0.0.1");
                InetSocketAddress socketAddress = new InetSocketAddress(address,4332);
                socket.connect(socketAddress);

                in=new DataInputStream(socket.getInputStream());
                out=new DataOutputStream(socket.getOutputStream());

                computer.setEnabled(true);
                thread.start();
            }
        }
        catch (IOException ee)
        {
            System.out.println(ee);
            socket=new Socket();
        }
    }
}

```

```

if(e.getSource()==computer)
{
    try
    {
        double a = Double.parseDouble(inputA.getText()),
        b = Double.parseDouble(inputB.getText()),
        c = Double.parseDouble(inputC.getText());
        double disk=b*b-4*a*c;
        if(disk>=0)
        {
            out.writeDouble(a);
            out.writeDouble(b);
            out.writeDouble(c);

            double root1 = in.readDouble(); //堵塞状态，除非读取到信息
            double root2 = in.readDouble();
            showResult.append("\n两个根:\n"+root1+"\n"+root2);
            showResult.setCaretPosition((showResult.getText()).length());
        }
        else
        {
            inputA.setText("此2次方程无实根");
        }
    }
    catch(Exception ee)
    {
        inputA.setText("请输入数字字符");
    }
}
}

```

```
public static void main(String args[])
{
    ClientFrame win = new ClientFrame();
}
```

Outline

- 11.1 URL类
- 11.2 读取URL中的资源
- 11.3 显示URL资源中的HTML文件
- 11.4 处理超链接
- 11.5 InetAddress类
- 11.6 套接字Socket
- 11.7 使用多线程处理套接字连接
- **11.8 UDP数据报**
- 11.9 广播数据报
- 11.10 Java远程调用

11.8 UDP数据报

- 基于UDP通信的基本模式是
 - (1) 将数据打包，称为数据包（好比将信件装入信封一样），然后将数据包发往目的地。
 - (2) 接收别人发来的数据包（好比接收信封一样），然后查看数据包中的内容。
-
- 1.发送数据
 - (1) 创建DatagramPacket对象
 - 首先，用DatagramPacket类将数据打包，即用DatagramPacket类创建一个对象，称为数据包。

11.8 UDP数据报

- 用DatagramPacket的以下两个构造方法创建待发送的数据包：
DatagramPacket(byte data[], int length, InetAddress address, int port)
DatagramPacket(byte data[], int offset, int length, InetAddress address, int port)
- 使用构造方法创建的数据包对象具有下列两个性质：
 - 含有data数组指定的数据
 - 该数据包将发送到地址是address、端口号是port的主机上
- 我们称address是它的目标地址、port是这个数据包的目标端口号。其中，第2个构造方法创建的数据包对象含有数组data从offset开始指定长度的数据。

11.8 UDP数据报

- 【例子】

```
byte data[]="近来好吗".getBytes();  
InetAddress address = InetAddress.getName("www.sina.com.cn");  
DatagramPacket data_pack = new DatagramPacket(data, data.length, address, 5678);
```

11.8 UDP数据报

- (2)发送数据
- 然后用DatagramSocket类的不带参数的构造方法DatagramSocket()创建一个对象，该对象负责发送数据包。

```
DatagramSocket mail_out = new DatagramSocket();  
mail_out.send(data_pack);
```

11.8 UDP数据报

- 2.接收数据
- DatagramSocket类的另一个构造方法DatagramSocket(int port)创建一个对象，其中的参数必须和待接收的数据包的端口号相同。例如，如果发送方发送的数据包的端口号是5678：

```
DatagramSocket mail_in = new DatagramSocket(5678);
```

11.8 UDP数据报

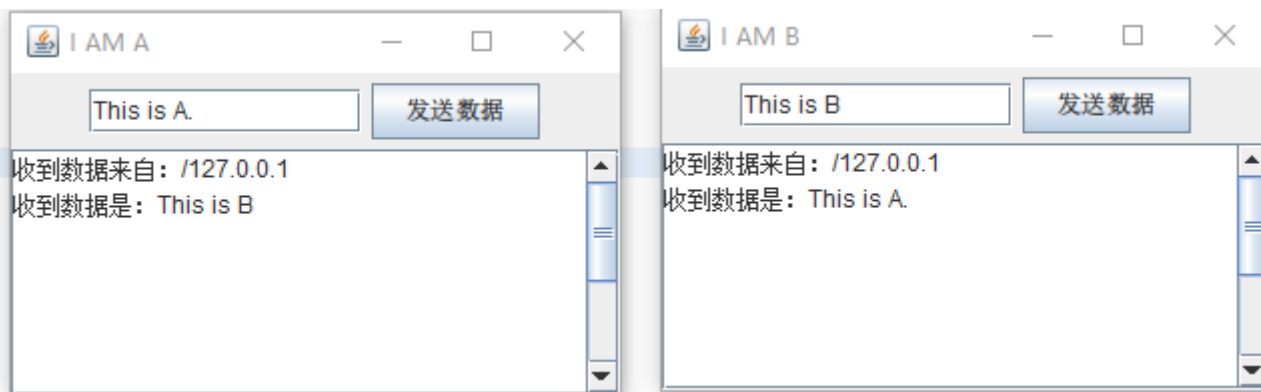
- 该对象mail_in使用方法receive(DatagramPacket pack)接收数据包。该方法有一个数据包参数pack，方法receive()把收到的数据包传递给该参数。因此，我们必须预备一个数据包以便收取数据包。这时需使用DatagramPack类的另外一个构造方法：DatagramPack(byte data[], int length)创建一个数据包，用于接收数据包，例如：
- 该数据包pack将接收长度为length的数据放入data。

```
byte[] data = new byte[100];  
int length = 90;  
DatagramPacket pack = new DatagramPacket(data, length);  
mail_in.receive(pack);
```

- 在下面的例子7中，两个主机（可用本地机模拟）互相发送和接收数据包。

11.8 UDP数据报

- 【例子7】
- Example11_7
 - A.java
 - B.java



```
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class A extends JFrame implements Runnable, ActionListener
{
    JTextField outMessage = new JTextField(12);
    JTextArea inMessage = new JTextArea(12,20);
    JButton b = new JButton("发送数据");

    A()
    {
        super("I AM A");
        b.addActionListener(this);

        setSize(320,200); setVisible(true);

        JPanel p = new JPanel();
        p.add(outMessage);
        p.add(b);

        Container con = getContentPane();
        con.add(new JScrollPane(inMessage), BorderLayout.CENTER);
        con.add(p, BorderLayout.NORTH);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        validate();

        Thread thread = new Thread(this);
        thread.start(); //线程负责接收数据
    }
}
```



```
public void actionPerformed(ActionEvent event)
{
    // 单击按钮发送数据
    byte b[] = outMessage.getText().trim().getBytes();
    try
    {
        InetAddress address = InetAddress.getByName("127.0.0.1");

        DatagramPacket data = new DatagramPacket(b,b.length,address,1234);

        DatagramSocket mail = new DatagramSocket();

        → mail.send(data);
    }
    catch(Exception e){}
}
```

```
public void run()
{ // 接收数据
    DatagramPacket pack = null;
    DatagramSocket mail = null;
    byte b[] = new byte[8192];
    try
    {
        pack = new DatagramPacket(b,b.length);
        mail = new DatagramSocket(5678);
    }
    catch(Exception e){}

    while(true)
    {
        try
        {
            → mail.receive(pack);
            String message = new String(pack.getData(),0,pack.getLength());
            inMessage.append("收到数据来自: " + pack.getAddress());
            inMessage.append("\n收到数据是: " + message+"\n");
            inMessage.setCaretPosition(inMessage.getText().length());
        }
        catch(Exception e){}
    }
}
```

```
public static void main(String args[])  
{  
    new A();  
}
```

```
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class B extends JFrame implements Runnable, ActionListener
{
    JTextField outMessage = new JTextField(12);
    JTextArea inMessage = new JTextArea(12,20);
    JButton b = new JButton("发送数据");
    B()
    {
        super("I AM B");
        b.addActionListener(this);

        setBounds(350,100,320,200); setVisible(true);

        JPanel p = new JPanel();
        p.add(outMessage);
        p.add(b);

        Container con=getContentPane();
        con.add(new JScrollPane(inMessage),BorderLayout.CENTER);
        con.add(p, BorderLayout.NORTH);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        validate();

        Thread thread = new Thread(this);
        thread.start(); // 线程负责接收数据
    }
}
```

```
public void actionPerformed(ActionEvent event)
{
    // 单击按钮发送数据
    byte b[]=outMessage.getText().trim().getBytes();
    try
    {
        InetAddress address = InetAddress.getByName("127.0.0.1");
        DatagramPacket data = new DatagramPacket(b,b.length,address,5678);
        DatagramSocket mail = new DatagramSocket();
        → mail.send(data);
    }
    catch(Exception e){}
}
```

```
public void run()
{
    // 接收数据
    DatagramPacket pack = null;
    DatagramSocket mail = null;
    byte b[]=new byte[8192];
    try
    {
        pack = new DatagramPacket(b,b.length);
        mail = new DatagramSocket(1234);
    }
    catch(Exception e){}

    while(true)
    {
        try
        {
            mail.receive(pack);
            String message=new String(pack.getData(),0,pack.getLength());
            inMessage.append("收到数据来自: "+pack.getAddress());
            inMessage.append("\n收到数据是: "+message+"\n");
            inMessage.setCaretPosition(inMessage.getText().length());
        }
        catch(Exception e){}
    }
}
```

```
public static void main(String args[])  
{  
    new B();  
}
```

Outline

- 11.1 URL类
- 11.2 读取URL中的资源
- 11.3 显示URL资源中的HTML文件
- 11.4 处理超链接
- 11.5 InetAddress类
- 11.6 套接字Socket
- 11.7 使用多线程处理套接字连接
- 11.8 UDP数据报
- **11.9 广播数据报**
- 11.10 Java远程调用

11.9 广播数据报

- 广播数据包类似于电台广播，进行广播的电台需在指定的波段和频率上广播信息，接收者只有将收音机调到指定的波段和频率上才能收听到广播的内容。
- 广播数据包涉及到地址和端口。我们知道，Internet的地址是**a.b.c.d**的形式。该地址的一部分代表用户自己的**主机**，而另一部分代表用户所在的**网络**。
 - 当a:0-127，那么**b.c.d**就用来表示主机，这类地址称做**A类地址**。
 - 当a:128-191，则a.b表示网络地址，而**c.d**表示主机地址，这类地址称做**B类地址**。
 - 当a:192-223，则网络地址是a.b.c，**d**表示主机地址，这类地址称做**C类地址**。
 - 当a:224-239，这类地址称做**D类地址**，D类地址并不代表某个特定主机的位置。

11.9 广播数据报

- 一个具有A类、B类或C类地址的主机要广播数据或接收广播，都**必须**加入到同一个**D类地址**。
- 一个D类地址也称做一个**组播地址**，加入到同一个组播地址的主机可以在某个端口上广播信息，也可以在某个端口上接收信息。

11.9 广播数据报

- 准备广播或接收的主机需经过下列步骤：
 1. 设置组播地址
- 使用InetAddress类创建**组播地址**，例如：
 - `InetAddress group = InetAddress.getByName("239.255.8.0");`
- 2. 创建多点广播套接字
- 使用MulticastSocket类创建一个多点广播套接字对象。MulticastSocket的构造方法：`public MulticastSocket(int port) throws IOException`创建的多点广播套接字可以在参数指定的端口上**广播**。

11.9 广播数据报

- 3.设置广播的**范围**
- 准备广播的主机必须让多点广播套接字（MulticastSocket）对象调用 `public void setTimeToLive(int ttl) throws IOException` 设置多播的**范围**（即多播数据包的默认生存时间）。
- 4.加入**组播组**
- 准备广播或接收的主机必须让多点广播套接字（MulticastSocket）对象调用 `public void joinGroup(InetAddress mcastaddr) throws IOException` 方法**加入**组播**组**。
- 多点广播套接字（MulticastSocket）对象调用 `public void leaveGroup(InetAddress mcastaddr) throws IOException` 方法可以**离开**已经加入的组播**组**。

11.9 广播数据报

- 5.广播数据和接收数据
- 进行广播的主机可以让多点广播套接字（MulticastSocket）对象调用 `public void send(DatagramPacket p) throws IOException` 将参数p指定的数据包**广播**到组播**组**中的其它主机。
- **接收**广播的主机可以让多点广播套接字（MulticastSocket）对象调用 `public void receive(DatagramPacket p) throws IOException` 方法来接收数据。

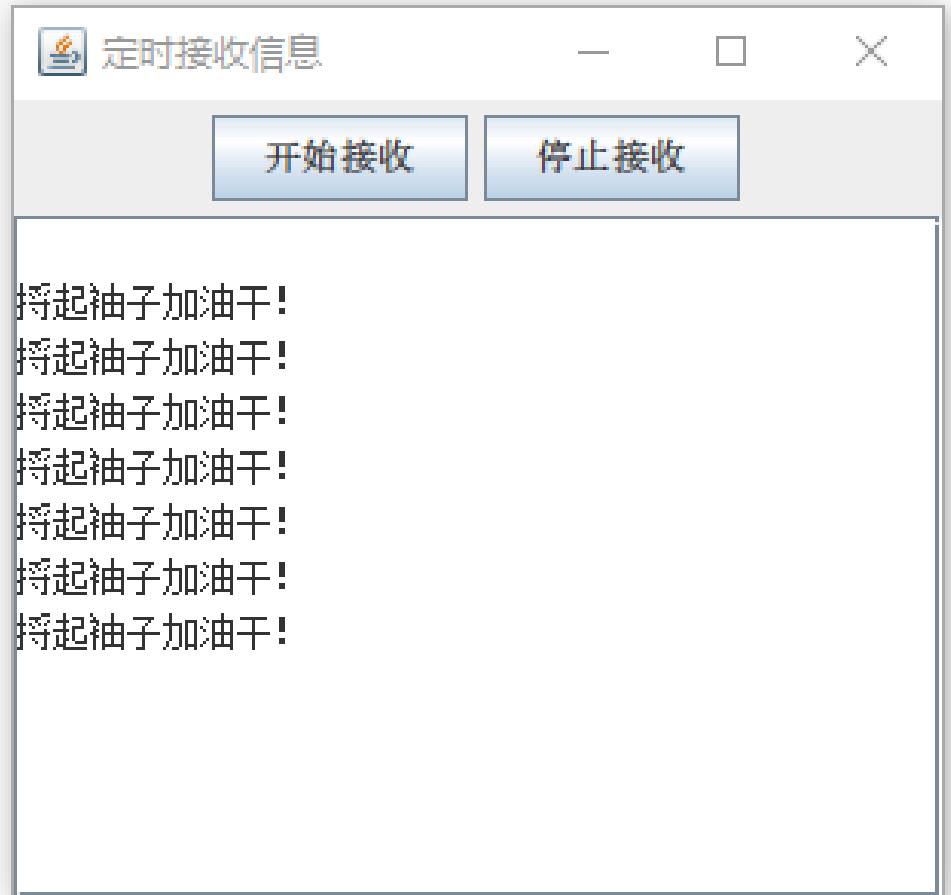
11.9 广播数据报

- 在下面的例子8中，一个主机不断地重复广播同一信息，加入到同一组的主机都可以随时接收广播的信息。接收者将正在接收的信息放入一个文本区，把已接收到的全部信息放入另一个文本区。

11.9 广播数据报

- 【例子8】
- Example11_8
 - BroadCast.java
 - Receive.java

撻起袖子加油干!
撻起袖子加油干!
撻起袖子加油干!
撻起袖子加油干!
撻起袖子加油干!
撻起袖子加油干!
撻起袖子加油干!
撻起袖子加油干!
撻起袖子加油干!
撻起袖子加油干!



```
import java.net.*;

public class BroadCast extends Thread
{
    String s = "撸起袖子加油干！";
    int port = 5858; // 组播的端口
    InetAddress group = null; // 组播组
    MulticastSocket socket = null; // 多点广播套接字

    BroadCast()
    {
        try
        {
            group = InetAddress.getByName("239.255.8.0"); // 设置组播组为239.255.8.0
            socket = new MulticastSocket(port); // 多点广播套接字将在port端口广播
            socket.setTimeToLive(0); // 多点广播套接字发送数据报范围为本地主机
            → socket.joinGroup(group); // 加入组播组,加入group后,socket发送的数据报可以被加入到
group中的成员接收到
        }
        catch(Exception e){}
    }
}
```



```
public void run()
{
    while(true)
    {
        try
        {
            DatagramPacket packet = null; // 待广播的数据报
            byte data[] = s.getBytes();
            packet = new DatagramPacket(data, data.length, group, port);
            System.out.println(new String(data));
            → socket.send(packet); // 广播数据报
            sleep(2000);
        }
        catch(Exception e){}
    }
}

public static void main(String args[])
{
    new BroadCast().start();
}
}
```

```
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Receive extends JFrame implements Runnable, ActionListener
{
    int port; // 组播的端口
    InetAddress group = null; // 组播组的地址
    MulticastSocket socket = null; // 多点广播套接字
    JButton startReceive, stopReceive;
    JTextArea showArea;
    Thread thread; // 负责接收信息的线程
    boolean stop = false;
```

```
public Receive()
{
    super("定时接收信息");
    thread = new Thread(this);
    startReceive = new JButton("开始接收");
    startReceive.addActionListener(this);
    stopReceive = new JButton("停止接收");
    stopReceive.addActionListener(this);

    showArea = new JTextArea(10,10);

    JPanel north = new JPanel();
    north.add(startReceive);
    north.add(stopReceive);

    Container con = getContentPane();
    con.add(north, BorderLayout.NORTH);
    con.add(new JScrollPane(showArea), BorderLayout.CENTER);

    port=5858;
    try
    {
        group = InetAddress.getByName("239.255.8.0");
        socket = new MulticastSocket(port);
        → socket.joinGroup(group);
    }
    catch(Exception e){}
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(320,300); validate(); setVisible(true);
}
```

```
public void actionPerformed(ActionEvent e)
{
    if(e.getSource()==startReceive)
    {
        if(!(thread.isAlive()))
        {
            thread=new Thread(this);
            stop=false;
        }
        try
        {
            thread.start();
        }
        catch(Exception ee){}
    }

    if(e.getSource()==stopReceive)
    {
        stop=true;
    }
}
```

```
public void run()
{
    while(true)
    {
        byte data[] = new byte[8192];
        DatagramPacket packet = null;
        packet = new DatagramPacket(data,data.length,group,port);
        try
        {
            → socket.receive(packet);
            String message = new String(packet.getData(),0,packet.getLength());
            showArea.append("\n"+message);
            showArea.setCaretPosition(showArea.getText().length());
        }
        catch(Exception e){}
        if(stop==true)
        {
            break;
        }
    }
}

public static void main(String args[])
{
    new Receive();
}
}
```

Outline

- 11.1 URL类
- 11.2 读取URL中的资源
- 11.3 显示URL资源中的HTML文件
- 11.4 处理超链接
- 11.5 InetAddress类
- 11.6 套接字Socket
- 11.7 使用多线程处理套接字连接
- 11.8 UDP数据报
- 11.9 广播数据报
- 11.10 Java远程调用

11.10 Java远程调用

- Java远程调用（Remote Method Invocation, RMI）是一种分布式技术，使用RMI可以让一个虚拟机上的应用程序请求调用位于网络上另一处虚拟机上的对象。
- 习惯上称发出调用请求的虚拟机为（本地）客户机，称接受并执行请求的虚拟机为（远程）服务器。

11.10 Java远程调用

- 1. 远程对象及其代理
- 1) 远程对象
- 驻留在（远程）服务器上的对象是客户要请求的对象，称作远程对象，即客户程序请求远程对象调用方法，然后远程对象调用方法并返回必要的结果。
- 2) 代理与存根（Stub）
- RMI不希望客户应用程序直接与远程对象打交道，代替地让用户程序和**远程对象的代理**打交道。代理的特点是它与远程对象**实现了相同的接口**，当用户请求代理调用这样的方法时，如果代理确认远程对象能调用相同的方法，就把实际的方法调用委派给远程对象。
- RMI会帮助生成一个存根（Stub）：一种特殊的字节码，并**让这个存根产生的对象作为远程对象的代理**。代理需要驻留在客户端。因此，在RMI中，用户实际上是在和远程对象的代理直接打交道，用户想请求远程对象调用某个方法，只需向**远程对象的代理**发出同样的请求即可，如图11.8所示。

11.10 Java远程调用

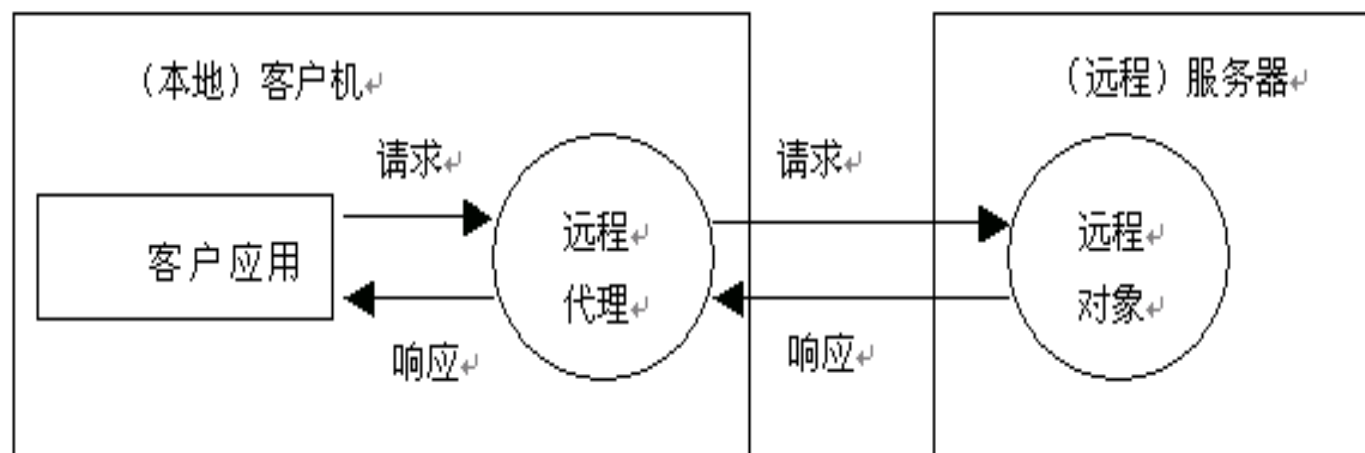


图 11.8 远程代理与远程对象

11.10 Java远程调用

- 3) Remote接口
- RMI为了标识一个对象是远程对象，即可以被客户请求的对象，要求远程对象必须实现java.rmi包中的**Remote接口**，也就是说只有实现该接口的类的实例才被RMI认为是一个远程对象。
- Remote接口中没有方法，该接口仅仅起到一个**标识**作用，因此，**必须扩展（extends）Remote接口**，以便规定远程对象的哪些方法是客户可以请求的方法。
- 用户程序不必编写和远程代理有关的代码，只需知道远程代理和远程对象实现了相同的接口。

11.10 Java远程调用

- 2.RMI的设计细节
- 为了叙述方便，我们假设本地客户机存放有关类的目录是D:\Client；远程服务器的IP是127.0.0.1，存放有关类的目录是D:\Server。
- 1) 扩展Remote接口
- 定义一个接口是java.rmi包中Remote的子接口，即扩展Remote接口。
- 我们定义的Remote的子接口是RemoteSubject。RemoteSubject子接口中定义了计算面积的方法，即要求远程对象为用户计算某种几何图形的面积。RemoteSubject的代码见下一页。

11.10 Java远程调用

```
import java.rmi.*;  
public interface RemoteSubject extends Remote  
{  
    public void setHeight(double height) throws RemoteException;  
    public void setWidth(double width) throws RemoteException;  
    public double getArea() throws RemoteException;  
}
```

- **Step 1** 该接口需要保存在远程服务器的D:\Server目录中，并编译它生成相应的.class字节码文件。
- **Step 2** 由于客户端的远程代理也需要该接口，因此需要将生成的字节码文件RemoteSubject.class复制到客户机的D:\Client目录中。

11.10 Java远程调用

- 2) 远程对象
- 创建远程对象的类必须要实现Remote接口，RMI使用Remote接口来标识远程对象。Remote接口中没有方法，因此创建远程对象的类需要实现Remote接口的一个子接口。在编写创建远程对象的类时，可以让该类是RMI提供的java.rmi.server包中的UnicastRemoteObject类的子类即可。

11.10 Java远程调用

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
public class RemoteConcreteSubject extends UnicastRemoteObject implements RemoteSubject
{
    double width,height;
    public RemoteConcreteSubject() throws RemoteException {}
    public void setWidth(double width) throws RemoteException{
        this.width=width;
    }
    public void setHeight(double height) throws RemoteException{
        this.height=height;
    }
    public double getArea() throws RemoteException {
        return width*height;
    }
}
```

Step 3

- 该类的子类，并实现了上述RemoteSubject接口，所创建的远程对象可以**计算矩形的面积**。

11.10 Java远程调用

- 3) 存根（Stub）与代理
- RMI负责产生存根（Stub Object），如果创建远程对象的字节码是RemoteConcreteSubject.class，那么存根（Stub）的字节码是RemoteConcreteSubject_Stub.class，即后缀为"_Stub"。

Step 4

- RMI使用rmic命令生成存根：首先进入D:\Server目录，然后执行如下rmic命令：rmic RemoteConcreteSubject

Step 5

- 客户端需要使用存根（Stub）来创建一个对象，即远程代理，因此需要将RemoteConcreteSubject_Stub.class复制到前面约定的客户机的D:\Client目录中。

Step 6

11.10 Java远程调用

- 4) 启动注册（rmiregistry）
- 在远程服务器创建远程对象之前，RMI要求远程服务器必须首先启动注册rmiregistry，只有启动了rmiregistry，远程服务器才可以创建远程对象，并将该对象注册到rmiregistry所管理的注册表中。
- 在远程服务器开启一个终端，比如在MS-DOS命令行窗口进入D:\Server

Step 7

目录，然后执行rmiregistry命令。

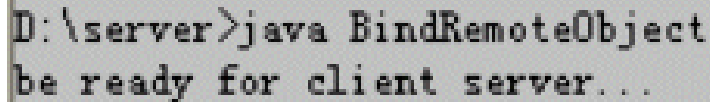
11.10 Java远程调用

- 5) 启动远程对象服务
- 远程服务器启动注册rmiregistry后，远程服务器就可以启动远程对象服务了，即编写程序来创建和注册远程对象，并运行该程序。远程服务器使用java.rmi包中的Naming类调用其类方法：**rebind(String name, Remote obj)** 绑定一个远程对象到rmiregistry所管理的注册表中，该方法的name参数是URL格式，obj参数是远程对象，**将来客户端的代理会通过name找到远程对象obj**。
- 以下是我们编写的远程服务器上的应用程序，运行该程序就启动了远程对象服务，该应用程序可以让用户访问它注册的远程对象。效果如图11.11。

11.10 Java远程调用

```
import java.rmi.*; //BindRemoteObject.java
public class BindRemoteObject {
    public static void main(String args[]) {
        try{
            RemoteConcreteSubject remoteObject = new RemoteConcreteSubject();
            Naming.rebind("rmi://127.0.0.1/rect", remoteObject);
            System.out.println("be ready for client server...");
        }
        catch(Exception exp){
            System.out.println(exp);
        }
    }
}
```

Step 8



```
D:\server>java BindRemoteObject
be ready for client server...
```

图 11.11 启动远程对象服务

11.10 Java远程调用

- 6) 运行客户端程序
- 远程服务器启动远程对象服务后，客户端就可以运行有关程序，访问使用远程对象。
- 客户端使用java.rmi包中的Naming类调用其类方法lookup(String name)返回一个远程对象的代理，即使用存根(Stub)产生一个和远程对象具有同样接口的对象。lookup(String name)方法中的name参数的取值必须是远程对象注册的name，比如："rmi://127.0.0.1/rect"。
- 客户程序可以像使用远程对象一样来使用lookup(String name)方法返回的远程代理。
- ClientApplication使用远程代理计算了矩形的面积。程序运行效果如图11.12所示。

11.10 Java远程调用

```
import java.rmi.*;
public class ClientApplication{
    public static void main(String args[]){
        try{
            Remote remoteObject = Naming.lookup("rmi://127.0.0.1/rect");
            RemoteSubject remoteSubject = (RemoteSubject)remoteObject;
            remoteSubject.setWidth(129);
            remoteSubject.setHeight(528);
            double area=remoteSubject.getArea();
            System.out.println("面积:"+area);
        }
        catch(Exception exp){
            System.out.println(exp.toString());
        }
    }
}
```

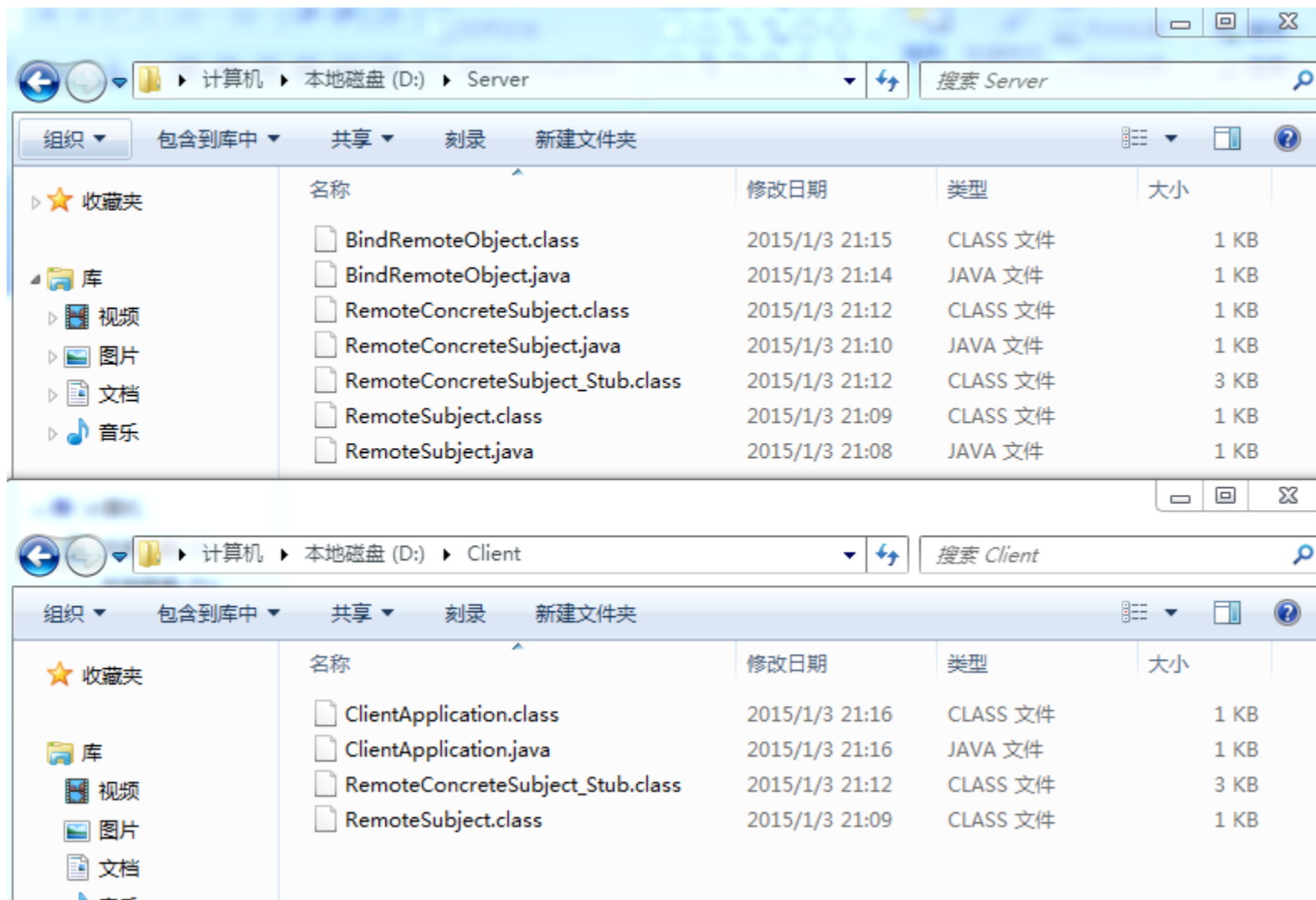
Step 9

```
D:\client>java ClientApplication
面积:68112.0
```

图 11.12 运行客户端程序

D:\Server
D:\Client

11.10 Java远程调用



Outline

- 11.1 URL类
- 11.2 读取URL中的资源
- 11.3 显示URL资源中的HTML文件
- 11.4 处理超链接
- 11.5 InetAddress类
- 11.6 套接字Socket
- 11.7 使用多线程处理套接字连接
- 11.8 UDP数据报
- 11.9 广播数据报
- 11.10 Java远程调用