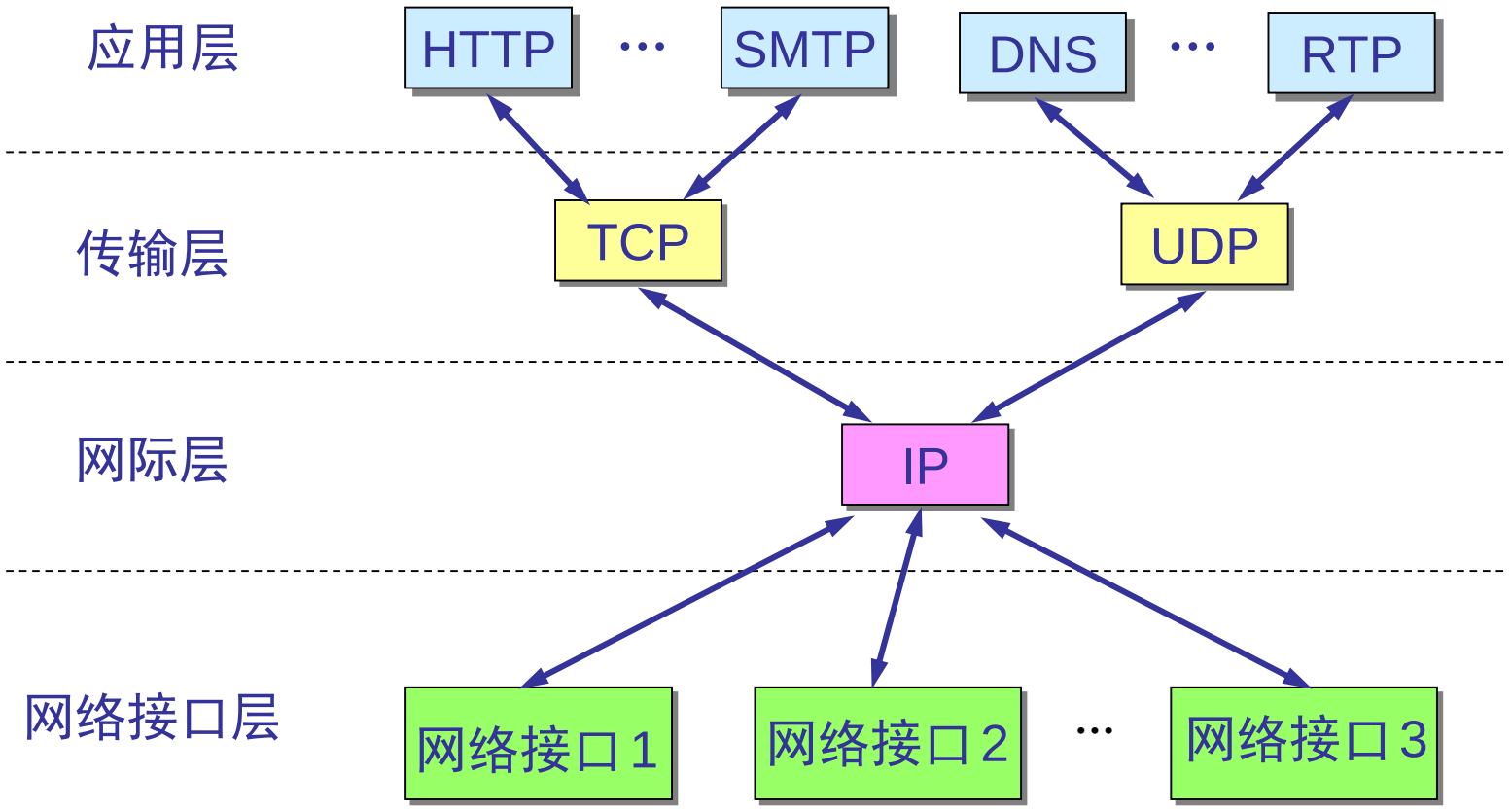
内容提要

- 网络编程基础
- Java对网络通信的支持机制
- Java访问Internet资源的类
- Socket通信机制
- 基于TCP的Socket编程
- 基于UDP的Socket编程
- 多播Socket编程

网络编程基础

- Internet网络协议

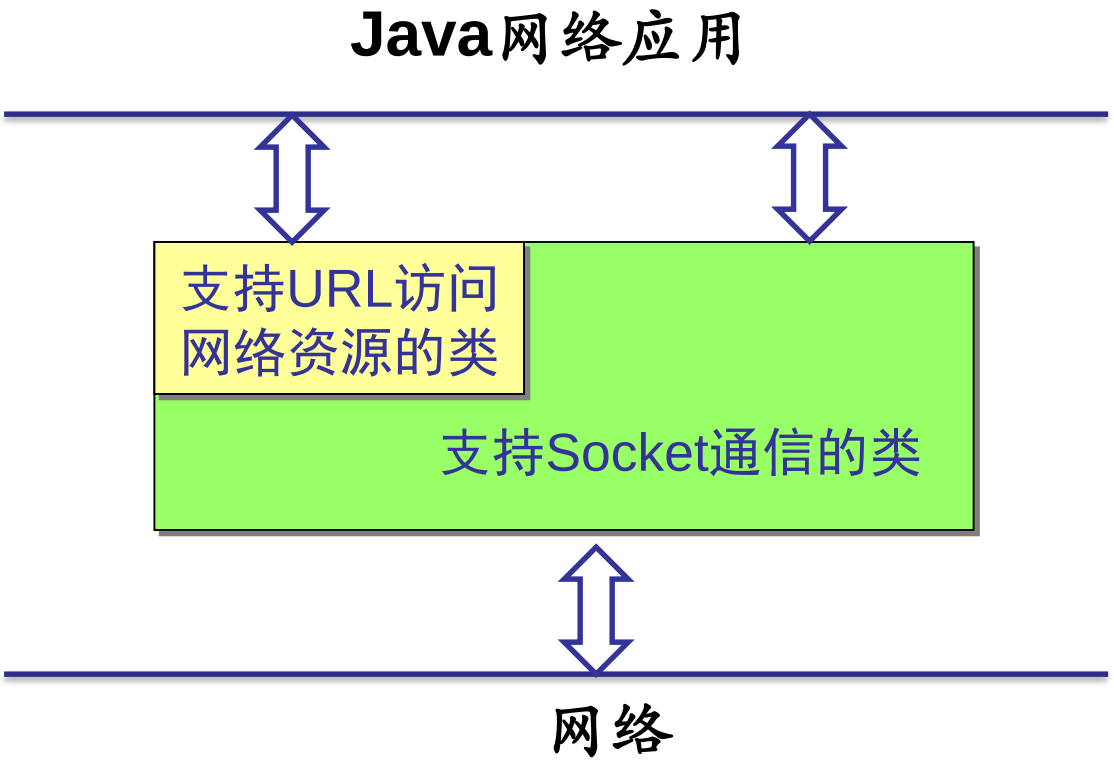


网络编程基础

- IP（Internet Protocol，Internet协议）
  - 是最底层的协议
  - 定义了数据按照数据报（Datagram，一种自带寻址信息的、独立地从数据源走到终点的数据包）传输的格式和规则
- TCP（Transport Control Protocol，传输控制协议）
  - 建立在IP之上，定义了网络上程序到程序的数据传输格式和规则，提供了IP数据包的传输确认、丢失数据包的重新请求、将收到的数据包按照它们的发送次序重新装配的机制
  - 是面向连接的协议，在开始数据传输之前，必须先建立明确的连接
- UDP（User Datagram Protocol，用户数据报协议）
  - 与TCP相似，比TCP具有更好的传输效率。
  - 不可靠的，不保证数据的传输，也不提供重新排列次序或重新请求功能，是一种无连接协议

Java对网络通信的支持机制

- Java是一种网络编程能力很强的语言，它提供了很多内置的网络功能，使得基于Internet和Web的应用开发变得更加容易



Java对网络通信的支持机制

- 支持URL(Uniform Resource Locator，统一资源定位符)的类
  - 用户不需要考虑URL中标识的各种协议的处理过程，就可以直接获得URL资源信息
  - 适合于访问Internet尤其是WWW上的资源
- 支持Socket通信的类
  - 套接字Socket表示应用程序和网络之间的接口，例如TCP Socket，UDP Socket
  - Socket通信主要针对C/S模式的应用和实现某些特殊协议的应用
  - 通信过程是基于TCP/IP协议中的传输层接口Socket来实现的，Java有一组对应Socket机制的类
  - 用户需要自己考虑通信双方约定的协议，虽然繁琐，当具有更大的灵活性和更广泛的使用领域
  - 支持URL的类也依赖于下层支持Socket通信的类来实现，不过这些类中已有FTP，HTTP等主要协议的处理

Java对网络通信的支持机制

- Java支持网络通信的类在*java.net*包中
  - URL类、URLConnection类、Socket类和ServerSocket类都使用TCP协议实现网络通信
  - DatagramSocket类、MulticastSocket类和DatagramPacket类都使用UDP协议实现网络通信

Java访问Internet资源的类：InetAddress类

- 用于描述Internet主机的IP地址或域名
- 子类Inet4Address和Inet6Address分别表示IPv4地址和IPv6地址
- 没有提供构造方法，通过其静态方法创建该类对象

```
InetAddress A1=InetAddress.getByName("www.upc.edu.cn");
InetAddress A2=InetAddress.getByName("211.87.178.210");
Byte[] ip={211, 87, 178, 210}; // 字节数组组成的IP地址
InetAddress A3=InetAddress.getByAddress(ip);
InetAddress A4=InetAddress.getLocalHost();
```

- 常用方法
  - String **getHostName()** 获取此 IP 地址的主机名
  - String **getHostAddress()** 返回字符串表示的IP 地址
  - byte[] **getAddress()** 返回字节数组表示的IP 地址

【例12-1】InetAddress类的创建及使用

```
PC205/172.18.193.195
主机名：PC205
IP地址：172.18.193.195
www.upc.edu.cn/211.87.178.210
localhost/127.0.0.1
```

```
import java.net.*;
public class InetAddressTest {
    public static void main(String[] args) {
        try { InetAddress A1=InetAddress.getLocalHost();
              System.out.println(A1); // 输出主机名和IP地址
              System.out.println("主机名： "+A1.getHostName());
              System.out.println("IP地址： "+A1.getHostAddress());
              InetAddress A2=InetAddress.getByName("www.upc.edu.cn");
              System.out.println(A2);
              InetAddress A3=InetAddress.getByName(null); //代表本机的默
    认名称和IP地址
             System.out.println(A3);
        } catch (UnknownHostException e) {  e.printStackTrace(); }
    }
}
```

Java访问Internet资源的类：URL类

- URL的格式：协议名://主机名:端口+文件名+引用
  - 例如 http://www.upc.edu.cn
    http://127.0.01:8080/index.htm#Top
- 常用构造方法
  - URL(String url)
  - URL(String protocol,String host,String file)
  - URL(String protocol,String host,String port,String file)
- 获取URL信息的常用方法
  - String getProtocol() 获取协议名
  - String getHost() 获取主机名
  - Int getPort() 获取端口号
  - Int getDefaultPort() 获取协议的默认端口号
  - String getFile() 获取文件名
  - String getRef() 获取引用
  - String getQuery() 获取URL中的查询字符串部分

Java访问Internet资源的类：URL类

【例12-2】创建URL对象及相关信息的获取

协议：http
主机：127.0.0.1
端口：8080
该协议默认端口：80
文件名：/test/index.htm

```
import java.net.*;
public class URLTest {
   public static void main(String[] args) {
      try { URL url=new URL("http://127.0.0.1:8080/test/index.htm");
            System.out.println("协议："+url.getProtocol());
            System.out.println("主机："+url.getHost());
            System.out.println("端口："+url.getPort());
            System.out.println("该协议默认端口："+url.getDefaultPort());
            System.out.println("文件名："+url.getFile());
          } catch (MalformedURLException e) { e.printStackTrace();}
   }
}
```

Java访问Internet资源的类：URL类

- **openConnection方法**
  - URLConnection **openConnection**()
    throws IOException
  - 用于创建到**URL**指定资源的连接
- **openStream方法**
  - final InputStream **openStream**()
    throws IOException
  - 打开到 **URL** 的连接并返回一个用于从该连接读取数据的 **InputStream**对象
  - 利用该输入流可以方便地从网站下载文件（**htm、pdf、zip、mp3**等）

【例12-3】使用URL对象读取其所指向资源的数据，输出至控制台

```
import java.io.*;
import java.net.*;
import java.nio.charset.Charset;
public class DownloadFileByURL {
   public static void main(String[] args) throws Exception {
          URL url;
          url = new URL(“http://www.upc.edu.cn”); // 本页面编码为UTF-8
          InputStream in =url.openStream(); // 打开读取数据的输入流
          InputStreamReader isr=new
                    InputStreamReader(in, Charset.forName("UTF-8"));
          BufferedReader br=new BufferedReader(isr);
          String line;
          while( (line=br.readLine()) != null ) //逐行读取数据
          {  System.out.println(line); }
          br.close();
   }
}
```

【例12-4】下载文件：使用URL对象读取其所指向资源的数据并保存至文件(按二进制方式写入即按原样写入，不存在编码问题)

```
import java.io.*;
import java.net.*;
public class DownloadFileByURL4 {
   public static void main(String[] args) throws Exception {
         URL url = new URL("http://www.upc.edu.cn");
         String fileName="c:/index.htm";
         InputStream in =url.openStream(); // 打开读取数据的输入流
         FileOutputStream fos=new FileOutputStream(fileName);
         byte[] buf=new byte[1024]; //1k的字节数组，作为读取字节的缓存
         int len=0;
         while( (len=in.read(buf)) != -1 ){ //读取数据块(这里为1K)
            fos.write(buf,0,len); // 写入数据块
         }
         in.close();        fos.close();
         System.out.println("文件下载完成后！ ");
   }
}
```

试着下载一个MP3文件，例如：
http://qqma.tingge123.com:83/20081118/%E5%87%A1%E4%BA%BA%E6%AD%8C.mp3

Java访问Internet资源的类：URLConnection类

- 是一个抽象类。与URL类相似，用来读取一个URL所表示的资源。此外，它还可以进行写操作。
- 利用URLConnection进行读写URL资源的过程

① 创建URLConnection对象

```
URLConnection conn=url. openConnection();
```

② 处理设置参数

- void setDoInput(boolean doinput) 如果打算使用 URL 连接进行输入即读操作，则将参数设置为 true
- void setDoOutput(boolean dooutput) 如果打算使用 URL 连接进行输出即写操作，则将参数 设置为 true
- 说明：默认情况下，服务器建立连接后只会产生读取服务器信息的输入流。如果希望是得到一个向服务器进行写操作的输出流（将网页表单数据提交给服务器），则应将调用setDoOutput方法，并将参数设置为true

③ 建立通信连接：conn.connect(); // 建立实际的连接

Java访问Internet资源的类：URLConnection类

④ 建立到远程对象的连接后，可以访问该资源的头字段或通过IO流读写该资源的数据

- String getHeaderField(String name) 获取指定字段的值
- int getContentLength() 返回内容的长度（字节）
- String getContentType() 返回内容的类型
- String getContentEncoding() 返回内容的编码
- long getDate() 返回内容的发送日期
- InputStream **getInputStream()** 用于获取响应的内容
- OutputStream **getOutputStream()** 用于向**URLConnection**发送请求参数

– 关于HTTP参数请求方式的说明

- **GET方式**：请求参数放在URL字符串之后发送给服务器，用户可见
- **POST方式**：要通过URLConnectin对象对应的输出流来发送请求参数，用户不可见

【例12-5】使用URLConnection获取响应的头字段

```
import java.net.*;
import java.util.*;
public class HeaderInfo {
    public static void main(String[] args) throws Exception {
        URL url = new URL("http://www.upc.edu.cn");
        URLConnection conn = url.openConnection(); // 创建连接对象
        conn.connect(); // 建立与远程对象的实际连接
        __.println("内容编码：" + conn.getContentEncoding());
        __.println("内容长度：" + conn.getContentLength());
        __.println("内容类型：" + conn.getContentType());
        __.println("发送时间：" + new Date(conn.getDate()));

        Map<String, List<String>> header = conn.getHeaderFields();
        Set<String> keys = header.keySet();
        for (String key : keys) {
            String value = conn.getHeaderField(key);
            __.println(key + "------" + value);
    }
}
```

试获取一个MP3文件的大小，例如：
http://qqma.tingge123.com:83/20081118/%E5%87%A1%E4%BA%BA%E6%AD%8C.mp3

- 运行结果

```
内容编码：null
内容长度：-1
内容类型：text/html
发送时间：Thu Oct 13 16:50:46 CST 2016
Transfer-Encoding ---> chunked
Keep-Alive ---> timeout=5, max=100
Accept-Ranges ---> bytes
null ---> HTTP/1.1 200 OK
Server ---> YxlinkWAF
Connection ---> Keep-Alive
Date ---> Thu, 13 Oct 2016 08:50:46 GMT
Content-Type ---> text/html
```

换一个URL地址试一试，例如：
https://www.sina.com.cn

- 说明
  - 各www服务器发送的对http请求的响应头字段各不相同
  - 只有服务器发送了某个响应字段，该字段才可用

【例12-6】使用URLConnection读取URL资源数据

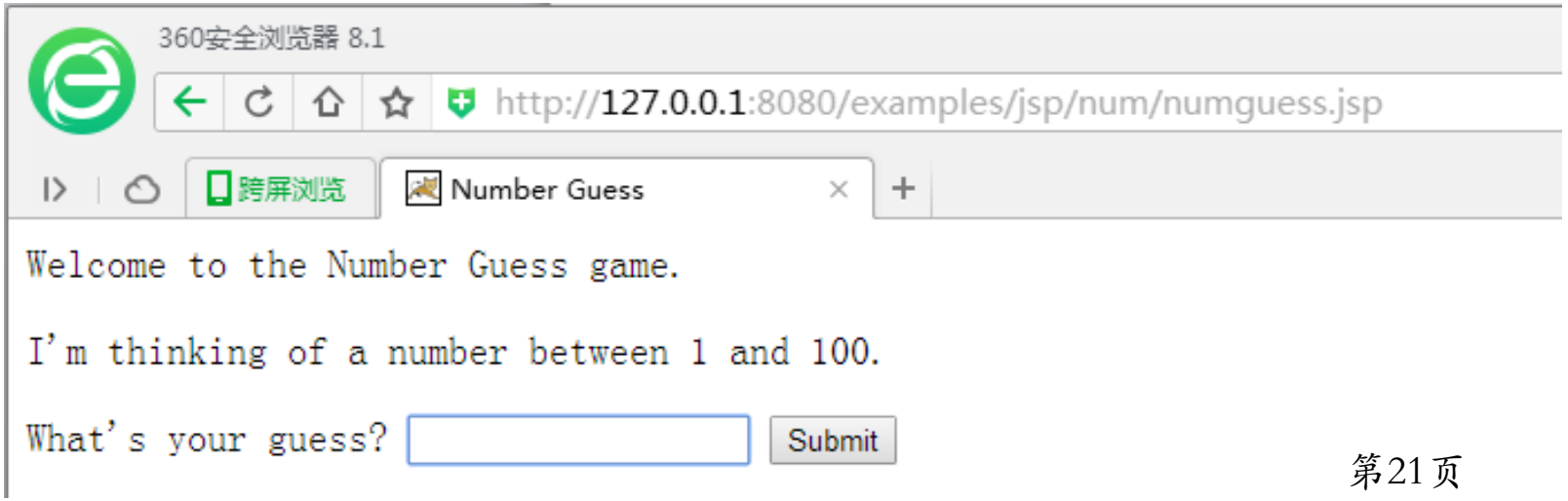
```
import java.io.*;
import java.net.*;
import java.nio.charset.Charset;
public class DownloadFileByURL {
   public static void main(String[] args) throws Exception {
          URL url;
          url = new URL(“http://www.upc.edu.cn”); // 本页面编码为UTF-8
          URLConnection conn = url.openConnection(); // 创建连接对象
          conn.connect(); // 建立与远程对象的实际连接
          InputStream in =conn.getInputStream(); // 获取读取数据的输入流
          InputStreamReader isr=new
                    InputStreamReader(in, Charset.forName("UTF-8"));
          BufferedReader br=new BufferedReader(isr);
          String line;
          while( (line=br.readLine()) != null ) {  System.out.println(line); }
          br.close();
   }
}
```

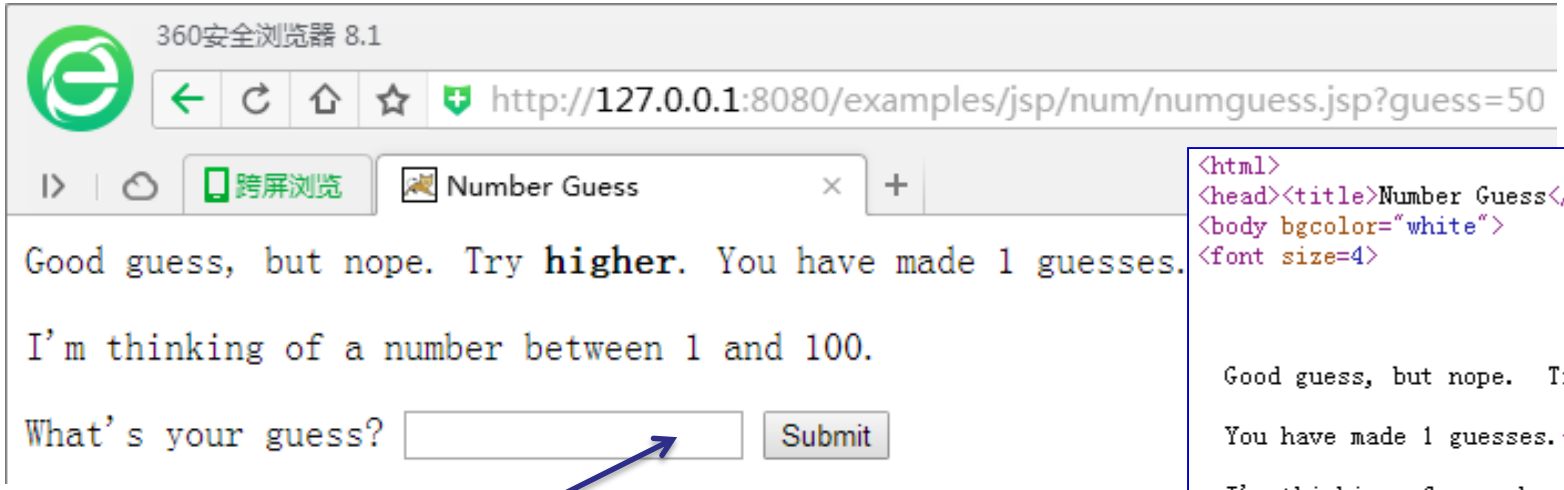
【例12-7】向Web站点发送GET请求：参数直接跟在URL后。此外，从cookie获取会话sessionID，用于保持与服务器的会话

```
import java.io.*;                          import java.net.*;
import java.nio.charset.Charset;   import java.util.Scanner;
public class GETforURLConnection {
   private static String sessionID=""; // 客户端与服务器的http会话ID
   // 从cookie获取会话sessionID，用于保持与服务器的会话
   public static String getSessionID(String actionURL) throws IOException{
         String sessionID="";
         URL url = new URL(actionURL);
         URLConnection connection = url.openConnection();
         String cookieValue = connection.getHeaderField("set-cookie");
         if(cookieValue != null)
           // 其中的session格式为：JSESSIONID=XXXXXXXXX
           // 例如：Set-Cookie --->
      JSESSIONID=547F1F10359CDF0D0B8F987304FE4EDA;path=/examples/;HttpOnly
        sessionID = cookieValue.substring(0, cookieValue.indexOf(";"));
         return sessionID;
   }
```

```
// 以GET方式发送http请求，并读取url资源数据
public static String sendGet(String urlStr, String param) throws Exception {
        String result = "";
        URL url = new URL(urlStr + "?" + param);
        URLConnection conn = url.openConnection(); // 创建连接对象
        if(!sessionID.equals("")){ //如果sessionID存在，即存在会话
           conn.setRequestProperty("cookie", sessionID); // 保持该会话，否则就是一个新的会话
        }
        conn.connect(); // 建立与远程对象的实际连
        InputStream in = conn.getInputStream(); // 获取读取数据的输入流
        InputStreamReader isr = new
                        InputStreamReader(in, Charset.forName("UTF-8"));
        BufferedReader br = new BufferedReader(isr);
        String line;
        while ((line = br.readLine()) != null) { result += line + "\n"; }
        br.close();

        return result;
}
```

```
public public static void main(String[] args) throws Exception {
        String urlStr, result;
        urlStr ="http://127.0.0.1:8080/examples/jsp/num/numguess.jsp";

        sessionID=getSessionID(urlStr); // 先获取与http请求的会话ID

        Scanner scan=new Scanner(System.in);
        System.out.println("What's your guess?");
        while(scan.hasNextInt()) {
            result=sendGet(urlStr,"guess="+scan.nextInt());
            System.out.println(result);
            System.out.println("What's your guess?");
        }
    }
}
```

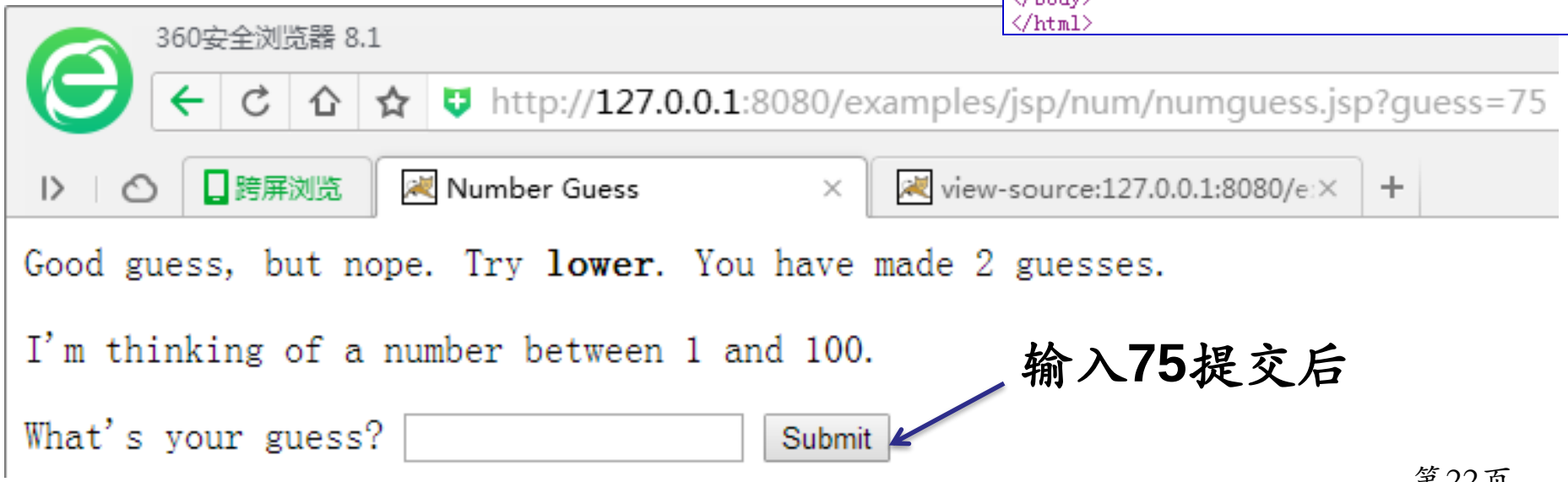




输入50提交后

```
<html>
<head><title>Number Guess</title></head>
<body bgcolor="white">
<font size=4>


  Good guess, but nope.  Try <b>higher</b>.

  You have made 1 guesses.<p>

  I'm thinking of a number between 1 and 100.<p>

  <form method=get>
  What's your guess? <input type=text name=guess>
  <input type=submit value="Submit">
  </form>



</font>
</body>
</html>
```

页面源码，内容与控制台的输出相同



输入75提交后

【例12-8】向Web站点发送POST请求：先设置doIn和doOut两个字段的值，然后使用URLConnection的输出流向服务器发送参数请求，最后读取服务器返回的资源数据。

```
public static String sendPost(String urlStr, String param) throws Exception {
    String result = "";
    URL url = new URL(urlStr);
    URLConnection conn = url.openConnection(); // 创建连接对象
    if(!sessionID.equals("")){ //如果sessionID存在，即存在会话
        conn.setRequestProperty("cookie", sessionID); // 保持该会话
    }
    // 发送POST请求必须设置如下两行
    conn.setDoInput(true);
    conn.setDoOutput(true);

    conn.connect(); // 建立与远程对象的实际连接

    //获取URLConnection对象对应的输出流
    OutputStream os=conn.getOutputStream();
    OutputStreamWriter out=new OutputStreamWriter(os);
    out.write(param); // 写入请求参数，即发送请求参数
    out.flush(); // flush输出流的缓冲
```

```
    InputStream in = conn.getInputStream(); // 获取读取数据的输入流
    InputStreamReader isr = new InputStreamReader(in, Charset.forName("UTF-8"));
    BufferedReader br = new BufferedReader(isr);
    String line;
    while ((line = br.readLine()) != null) { result += line + "\n"; }

    out.close();   br.close();
    return result;
}
 public public static void main(String[] args) throws Exception {
        String urlStr, result;
        urlStr ="http://127.0.0.1:8080/examples/jsp/num/numguess.jsp";
        sessionID=getSessionID(urlStr); // 先获取与http请求的会话ID
        Scanner scan=new Scanner(System.in);
        System.out.println("What's your guess?");
        while(scan.hasNextInt()) {
            result=sendPost(urlStr,"guess="+scan.nextInt());
            System.out.println(result);
            System.out.println("What's your guess?");
        }
}
```

Socket通信机制

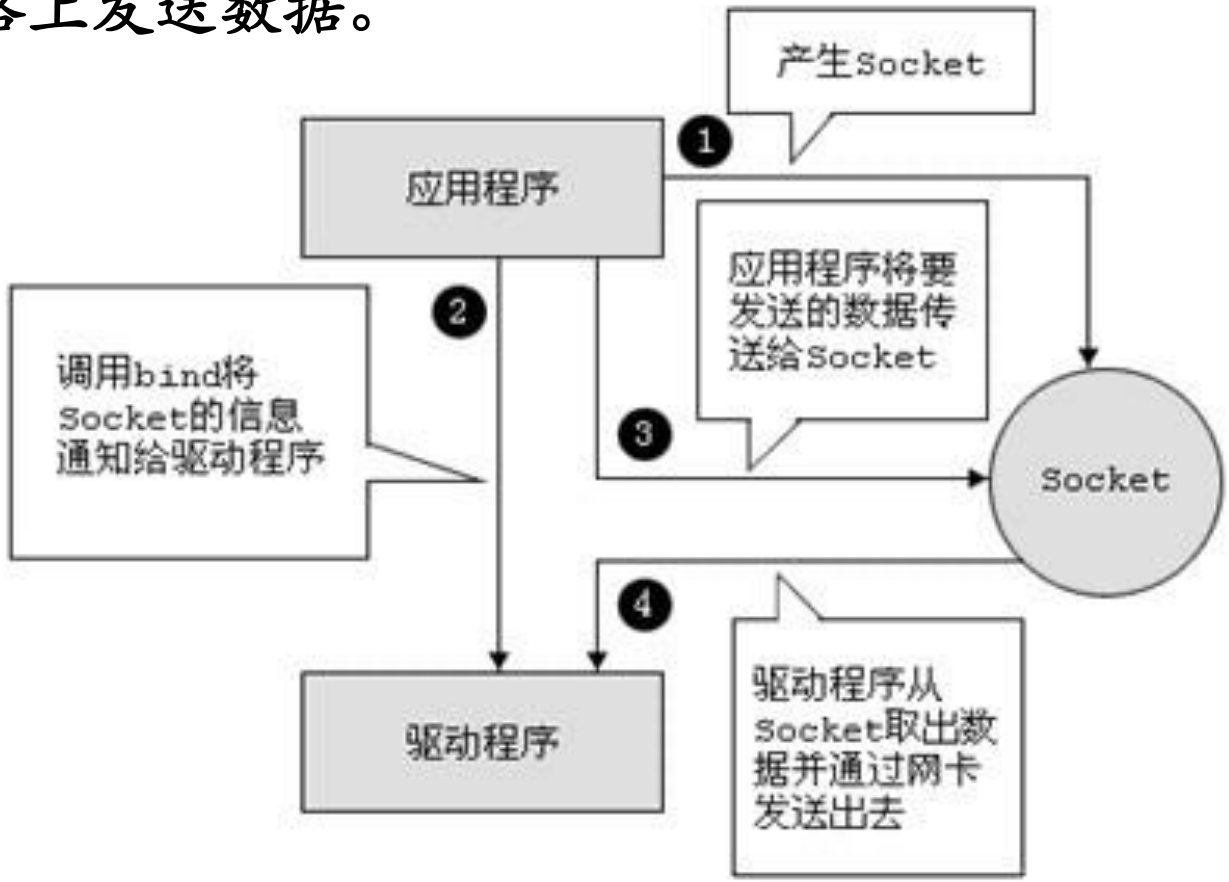
- Socket是两个进程进行双向数据传输的网络通信端口，由一个IP地址和一个端口号来标识。每个服务进程都在一个众所周知的端口上提供服务，而想使用该服务的客户端进程则需要连接到该端口。
- 是一种底层的通信机制，通过Socket的数据是原始字节，通信双方必须根据约定的协议对数据进行处理和解释
- 提供了两种通信方式
  - 面向连接方式（TCP）
  - 无连接方式（UDP）

Socket通信机制

- Socket是网络驱动层提供给应用程序的编程接口和一种机制。
- 可以把 Socket 比喻成是一个港口码头。应用程序只要把货物放到港口码头上，就算完成了货物的运送。对于接收方应用程序也要创建一个港口码头，只需要等待货物到达码头后将货物取走。
- Socket 是在应用程序中创建的，它是通过一种绑定机制与驱动程序建立关系，告诉自己所对应的 IP 和 Port。在网络上传输的每一个数据帧，必须包含发送者的 IP 地址和端口号。

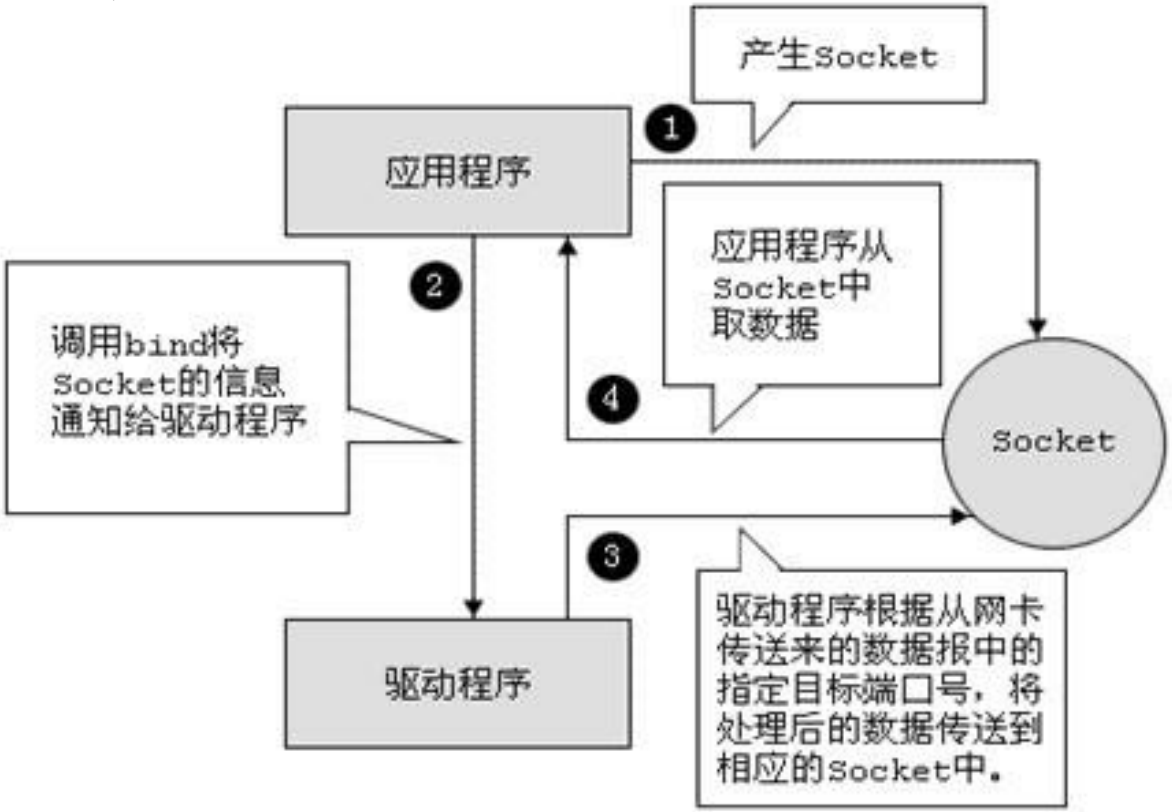
Socket通信机制

- Socket发送数据的过程：创建完 Socket 以后，应用程序写入到 Socket 的数据，由 Socket 交给驱动程序向网络上发送数据。



Socket通信机制

- **Socket接收数据的过程**：网卡从网络上收到与某个 Socket 绑定的 IP 和 Port 相关的数据后，由驱动程序再交给 Socket ，应用程序就可以从这个 Socket 中读取接收到的数据。

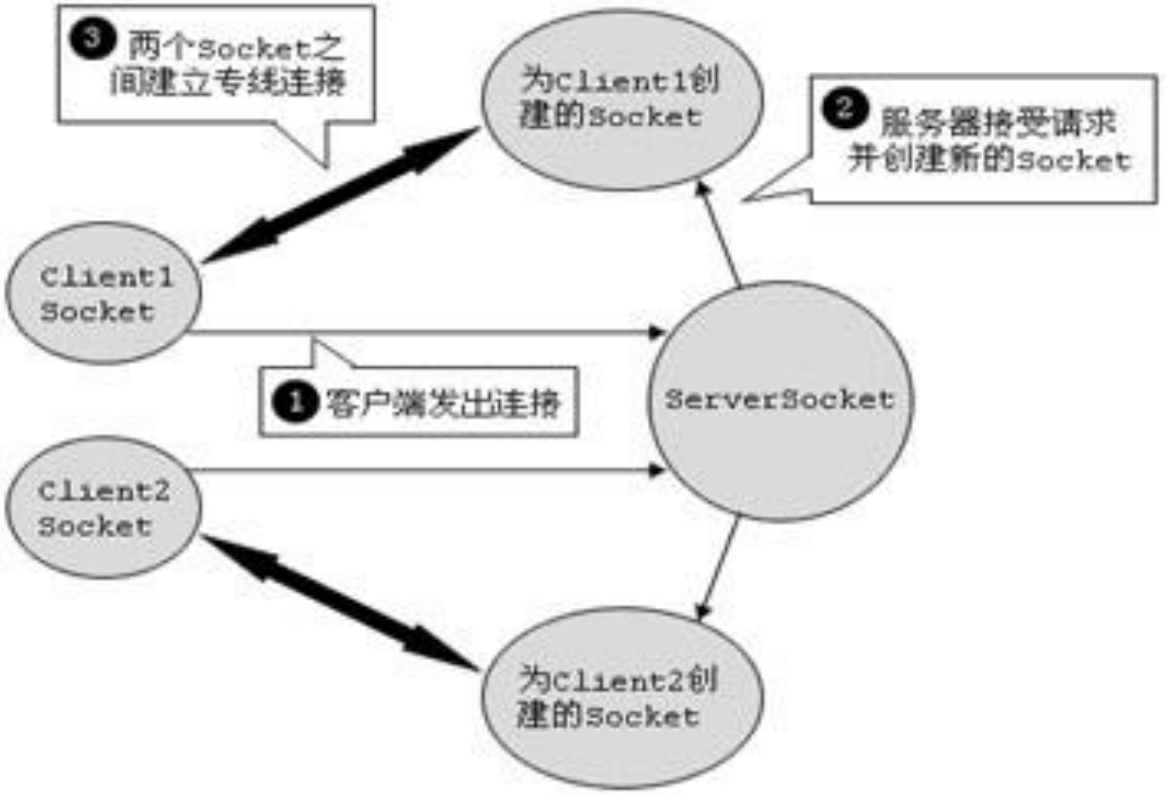


Socket通信机制

- Java支持Socket通信的类（*java.net*包）
  - 基于TCP的
    - ServerSocket类—表示连接的Server端，负责监听客户端的连接请求，接受一个连接请求后将产生一个与客户端Socket通信的Socket
    - Socket类—表示连接的Client端Socket
    - 两端的Socket成对出现
  - 基于UDP的
    - DatagramSocket类—表示发送和接收数据报的端口
- 通信双方都需要知道对方的IP地址和端口号
- 服务端必须先运行，以等待连接（基于TCP）或等待接收数据报（基于UDP）

基于TCP的Socket通信

- Server端Listen(监听)某个端口是否有连接请求，Client端向Server端发出Connect(连接)请求，Server端向Client端发回Accept（接受）消息。一个连接就建立起来了。此后，双方就可以相互收发数据。 TCP Socket的通信过程：



基于TCP的Socket通信：Server端

- 用Java建立简单的服务器程序需要5个步骤
  - **Step 1**：创建ServerSocket对象

    ```
    ServerSocket server = new ServerSocket (port);
    ```

    或者：`new ServerSocket ( port, backlog);`
    - int Port: 指定一个可用的端口号(一般应大于1024)，用来定位服务器上的服务器应用程序
    - int backlog: 指定socket连接请求队列的最大长度（未指定则为50）。如果队列满时收到连接指示，则拒绝该连接。
  - **Step 2**：通过ServerSocket的accept方法监听并接受客户的连接，服务器无限期的监听客户连接

    ```
    Socket socket = server.accept();
    ```

    - 有一个客户连接时，将产生并返回一个socket

基于TCP的Socket通信：Server端

- 用Java建立简单的服务器程序需要5个步骤
  - **Step 3**：通过Socket的方法getInputStream和getOutputStream获取InputStream和OutputStream对象；通常将其他流类型和它们联系起来，如：

```
InputStreamReader isr = new
                    InputStreamReader( socket.getInputStream( ) );
OutputStreamWriter osw = new
                    OutputStreamWriter( socket.getOutputStream( ) );
```

  - **Step 4**：客户和服务器通过I/O流对象进行通信
  - **Step 5**：通信传输完毕，服务器通过调用流和套接字的close方法关闭连接

基于TCP的Socket通信：Client端

- 建立客户端程序与服务端类似，分为4个步骤
  - **Step 1**：创建一个Socket，实现与服务器的连接

    ```
    Socket  socket = new Socket ( InetAddress addr, int port);
    ```

    - addr：服务器IP地址；port：服务器I端口号
    - 连接成功，将返回一个Socket，否则产生异常
  - **Step 2**：通过getInputStream和getOutputStream分别获取Socket的InputStream和OutputStream的引用。同样，也可以将其他流类型与InputStream和OutputStream联系起来
  - **Step 3**：客户和服务器通过I/O流对象进行通信
  - **Step 4**：通信传输完毕，调用流和套接字的close方法关闭连接

基于TCP的Socket通信应用

- 单个Client端与Server端通信应用

【例12-9】客户端与服务端建立通信连接后，双方互相收发数据。

- 服务端：接收客户端发送的文本消息，并向客户端发送当前时间表示对该消息的确认；每收到一个消息，就发送一个确认。
- 客户端：给服务端发送文本消息，并接收服务端发送的确认

```
import java.io.*;          import java.net.*;                    import java.util.*;
public class Server {
   public static void main(String[] args) throws IOException {
      ServerSocket server = null;
      Socket socket = null;
       try { server = new ServerSocket(1088, 2);
             System.out.println("服务器已启动，等待客户连接请求...");
             while(true){ // 循环监听客户端请求。当前客户端退出通信后，可
   以接受下一个客户端的连接请求
                  //监听客户连接并接受(此时线程阻塞，直到监听到有客户连接)
                   socket = server.accept();
                   __.println("接受了来自客户端" + socket + "的连接请求");
                   // 获取socket的输入流，用于读取客户端发来的消息
                   InputStream is = socket.getInputStream();
                   InputStreamReader isr = new InputStreamReader(is);
                   BufferedReader br = new BufferedReader(isr);
                   // 获取socket的输出流，用于向客户端发送消息
                   OutputStream os = socket.getOutputStream();
                   OutputStreamWriter osr = new OutputStreamWriter(os);
                   BufferedWriter bw = new BufferedWriter(osr);
// 约定：请求接收后，服务端向客户端发送一个消息“socket通信就绪”
                   bw.write("socket通信就绪");
                   bw.newLine(); // 写入行分隔符
                   bw.flush(); // 立即发送
```

Server
端程序

```
            try{
                while (true) { //循环接收客户端的输入信息
                    String line = br.readLine(); // 读取来自客户端的数据
                     __.println("来自" + socket + "的消息:" + line);
                     if(line.toUpperCase().equals("QUIT")){
                         socket.close(); // 关闭socket
                         __.println("连接关闭。等待下一个客户端连接...");
                        break;
                     }
                     bw.write(new Date().toString());//发送收到消息的时间
                     bw.newLine(); // 写入行分隔符
                     bw.flush(); // 立即发送
                } // end of while (true)循环接收客户端的输入信息
             }catch(IOException e1){
                 __.println(e1.getMessage());
                 socket.close(); // 关闭socket
                 __.println("连接关闭。等待下一个客户端连接...");
             }
        } // end of while (true)循环监听客户端请求。
    } catch (IOException e) { __.println(e.getMessage());  }
    finally {   __.println("服务器关闭");
                server.close(); // 关闭serverSocket
    }
  }
}
```

Client端程序

```
import java.io.*;          import java.net.*;                    import java.util.*;
public class Server {
   public static void main(String[] args) throws IOException {
         Socket socket = null;
         Scanner scan = new Scanner(System.in); //用于从键盘读入消息
         try {     socket = new Socket("127.0.0.1", 1088); // 与服务器建立
   socket连接(线程阻塞，直到加入请求队列);无法连接则抛出IO异常,不返回
   socket对象
                  System.out.println(socket.isConnected());
                  __.println("连接服务器成功(仅加入了服务端的socket请求队列,
              还要等待serverSocket接受该连接并产生与客户socket通信的socket) " );
                  __.println ("创建的socket对象：" + socket);
                  // 获取socket的输入流，用于读取客户端发来的消息
                  InputStream is = socket.getInputStream();
                  InputStreamReader isr = new InputStreamReader(is);
                  BufferedReader br = new BufferedReader(isr);
                  // 获取socket的输出流，用于向客户端发送消息
                  OutputStream os = socket.getOutputStream();
                  OutputStreamWriter osr = new OutputStreamWriter(os);
                  BufferedWriter bw = new BufferedWriter(osr);
```

```
// 约定：请求接收后，服务端向客户端发送一个消息“socket通信就绪”
            __.println(br.readLine()); // 读取服务端的消息
            while (scan.hasNextLine()) {
                String msgSend = scan.nextLine(); // 从键盘读入消息
                bw.write(msgSend); // 向服务端发送消息
                bw.newLine();
                bw.flush(); // 立即发送

                String line = br.readLine(); // 读取服务端的数据
                __.println("来自服务端[IP=" + socket.getInetAddress().
                                        getHostAddress() + "]的消息:" + line);
                if (msgSend.toUpperCase().equals("QUIT"))
                   break; // 退出while
            }
        } catch (IOException e) {  __.println(e.getMessage()); }
        finally {
                    if (socket!=null) socket.close(); // 关闭socket
                    scan.close(); // 关闭scanner
                    __.println("连接关闭");
        }
    }
}
```

基于TCP的Socket通信应用

- 多个Client端与Server端并发通信应用

【例12-10】在前一个例中，服务端不支持与多个客户端的并发通信。要实现与多个客户端的并发通信，服务端应建立多个Socket和多个线程，在每个线程中引用一个Socket，负责与一个客户端的Socket通信。

- 服务端：设置一个最大客户数，循环监听客户连接。如果客户数未达到最大，一旦有客户连接就接受它，此时得到一个能与该客户通信的Socket；接着创建一个线程，其中引用该Socket，实现与客户通信；接收客户端发送的文本消息，并向客户端发送当前时间表示对该消息的确认；每收到一个消息，就发送一个确认。
- 客户端：给服务端发送文本消息，并接收服务端发送的确认。（程序代码同前例）

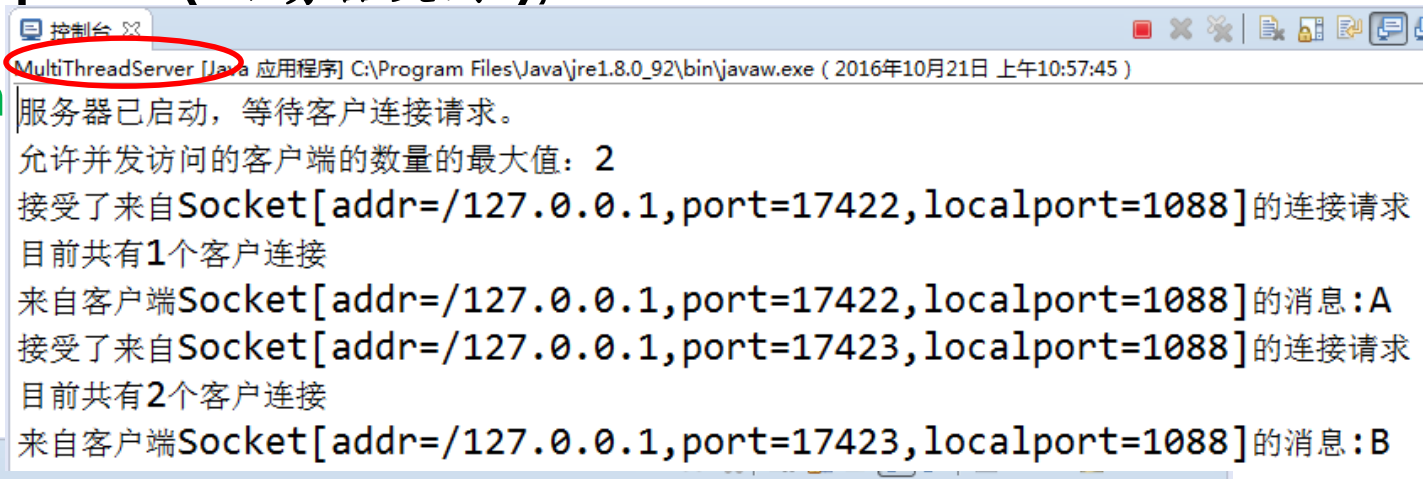
多线程的Server端程序

```
import java.io.*;
import java.net.*;
import java.util.*;
public class MultiThreadServer implements Runnable {
   private static int ClientsNumber = 0; // 连接到服务端的Client端的数量
   private Socket socket; // 与客户端socket通信的socket

   public MultiThreadServer(Socket socket) {
         this.socket = socket;
         ClientsNumber++; // 线程创建后，客户端的数量加1
   }
   // Server端socket与客户端socket通信线程的线程体
   public void run() {
         try {    // 获取socket的输入流，用于读取客户端发来的消息
                  InputStream is = socket.getInputStream();
                  InputStreamReader isr = new InputStreamReader(is);
                  BufferedReader br = new BufferedReader(isr);
                  // 获取socket的输出流，用于向客户端发送消息
                  OutputStream os = socket.getOutputStream();
                  OutputStreamWriter osr = new OutputStreamWriter(os);
                  BufferedWriter bw = new BufferedWriter(osr);
```

```
            bw.write("socket通信就绪");
            bw.newLine(); // 写入行分隔符
            bw.flush();

            while (true) {
                String line = br.readLine(); // 读取来自客户端的数据
                __.println("来自[IP=" + socket.getInetAddress().
                                    getHostAddress() + "]的消息:" + line);
                if(line.toUpperCase().equals("QUIT")) break;
                // 约定:  向客户端发送一个收到消息的时间
                bw.write(new Date().toString());
                bw.newLine(); // 写入行分隔符
                bw.flush();
            }
        } catch (IOException e) { __.println(e.getMessage());  }
        finally {  try {    __.println("与客户端" + socket + "的连接关闭");
                            socket.close(); // 关闭socket
                            socket=null;
                    } catch (IOException e) { __.println(e.getMessage());}
                   ClientsNumber--; // 线程结束，客户端的数量减1
        }
}
```

```
public static void main(String[] args) {
        final int MAX = 2; // 允许并发访问的最大客户端数量
        ServerSocket server = null;
        Socket[ ] sockets = new Socket[MAX]; //用于引用多少个Socket对象的数组
        try { server = new ServerSocket(1088,1); // 请求队列最大长度为1
              __.println("服务器已启动，等待客户连接请求。");
            while(true){ // 无限循环，任何时候只要客户数量小于允许的最大值，
就允许与一个新客户建立连接
                if (MultiThreadServer.ClientsNumber < MAX) {
                    try{ int k;
                        // 找一个为null或者已经关闭的socket的引用
                       for(k=0;k<MaxClientsNumber;k++)
                           if(sockets[k]==null)      break;
                           else if(sockets[k].isClosed()) break;
                        sockets[k] = server.accept();
                        // 创建socket与客户通信的线程
                        Thread t = new Thread(new MultiThreadServer(sockets[k]));
                        t.start(); // 启动线程
                       __.println("接受了来自" + sockets[k] + "的连接请求");
                       __.println("目前共有" +
                             MultiThreadServer.ClientsNumber + "个客户连接");
                    } catch (IOException e) { __.println(e.getMessage()); }
                }
            } // end of while(true)
        }
```

```
        } catch (IOException e) {  __.println(e.getMessage()); }
       finally { try {  server.close(); // 关闭serverSocket
                  } catch (IOException e) { __.println(e.getMessage()); }
                  __.println("服务器关闭");
        }
    } // end of  main
}
```



基于TCP的Socket通信应用

- Client端发送文件给Server端

【例12-11】客户端将一个文件传输给服务端：

- 服务端：首先监听并接受客户端连接，然后接收服务端发送的文件数据，并保存到文件，此后通信结束。
- 客户端：首先向服务器发送连接请求，建立通信连接后，读取文件数据并发送至服务器，此后通信结束。

基本思路：

- 客户端：用FileInputStream读取文件字节，然后将字节写入Socket的OutputStream中，以实现文件发送。
- 服务端：从Socket的InputStream中读取字节，然后使用FileOutputStream将字节写入文件，以实现文件接收。

Server端程序

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

public class Server3 {
   public static void main(String[] args) throws IOException {
          ServerSocket server = null;
          Socket socket = null;
          byte[] bytes=new byte[1024]; // 发送消息缓冲
          int length=0;
          try {
                    server = new ServerSocket(1088);
                    __.println("服务器已启动，等待客户连接请求。");
                    socket = server.accept();
                    __.println("接受了来自客户端" + socket + "的连接请求");

                     // 获取socket的输入流，用于读取客户端发来的消息
                    InputStream is = socket.getInputStream();
                    BufferedInputStream bis = new BufferedInputStream(is);
```

```
                // 接收文件
                String path="c:/recv_凡人歌.mp3";
                FileOutputStream fos=new FileOutputStream(path);
                length = bis.read(bytes); // 读取来自客户端的数据
                while (length>0) {
                        fos.write(bytes,0,length); // 写入文件流
                        length = bis.read(bytes); // 读取来自客户端的数据
                }
                fos.close();
                System.out.println("文件已保存至"+path);
        } catch (IOException e) {
                System.out.println(e.getMessage());
        } finally {
                System.out.println("连接关闭");
                socket.close(); // 关闭socket
                server.close(); // 关闭serverSocket
        }
    }
}
```

```
import java.io.*;      import java.net.*;
public class Client3 {
   public static void main(String[] args) throws IOException {
          Socket socket = null;
          byte[] bytes=new byte[1024]; // 发送消息缓冲
          int length=0;
          try {     socket = new Socket("127.0.0.1", 1088);
                    __.println("连接服务器成功：  " + socket);
                    OutputStream os = socket.getOutputStream();
                    BufferedOutputStream bos = new BufferedOutputStream(os);
                    // 向服务端发送一个文件
                    String path="c:/凡人歌.mp3";
                    FileInputStream fis=new FileInputStream(path);
                   length=fis.read(bytes); // 读取文件
                    while( length>0 ){
                              bos.write(bytes,0,length); // 写入socket的输出流
                              length=fis.read(bytes); // 读取文件
                    }
                    fis.close();
                    bos.flush(); // 立即发送
                    __.println(path+"已发送");
          } catch (IOException e) { __.println(e.getMessage()); }
         finally {  if (socket!=null) socket.close(); // 关闭socket
                    __.println("连接关闭");
          }
   }
}
```

Client
端程序

基于UDP的Socket通信

- UDP协议是一种不可靠的网络协议，它在通讯实例的两端个建立一个Socket，但这两个Socket之间并没有虚拟链路，这两个Socket只是发送和接受数据报的对象
- java.net包中提供了两个类用来支持数据报通信
  - DatagramSocket类用于在程序之间建立传送数据报的通信Socket。主要方法：
    - DatagramSocket(int port) 创建一个绑定到本机默认IP地址与指定的端口号的DatagramSocket对象
    - void receive(DatagramPacket p) throws IOException

      从此套接字接收数据报。当此方法返回时，DatagramPacket 的缓冲区填充了接收的数据。数据报包也包含发送方的 IP 地址和发送方机器上的端口号
    - void send(DatagramPacket p) throws IOException

      从此套接字发送数据报。DatagramPacket 包含的信息指示：将要发送的数据、其长度、远程主机的 IP 地址和端口号
    - void close() 关闭此数据报套接字

基于UDP的Socket通信

- DatagramPacket类表示数据报，是传输数据的载体
  - DatagramPacket(byte[] buf, int length)
    构造 DatagramPacket，用来接收长度为 length 的数据包
  - DatagramPacket(byte[] buf, int length, InetAddress address, int port) 构造数据报，用来将长度为 length 的包发送到指定主机上的指定端口号。
  - byte[] getData() 返回数据缓冲区。接收到的或将要发送的数据从缓冲区中的偏移量 offset 处开始，持续 length 长度
  - int getOffset() 返回将要发送或接收到的数据的偏移量
  - int getLength() 返回将要发送或接收到的数据的长度
  - InetAddress getInetAddress() 获取远程主机的IP
  - int getPort() 获取远程主机的通信端口号

基于UDP的Socket通信

- 用数据报方式编写通信程序的步骤
  - **Step 1**：建立一个DatagramSocket对象

    DatagramSocket socket = new DatagramSocket (int port)
  - **Step 2**：创建用于接收或发送的数据报DatagramPacket

    DatagramPacket sendPacket=new

    DatagramPacket (byte[] buf, int length)

    或者：DatagramPacket recvPacket=new DatagramPacket (

    byte[] buf, int length, InetAddress address, int port)
  - **Step 3**：调用所创建的DatagramSocket对象的receive或send方法来接收或发送数据报

    socket.receive(recvPacket)

    或者： socket.send(sendPacket)

    通常建立一个线程，专门用于接收数据报
  - **Step 4**：通信结束后，关闭DatagramSocket对象

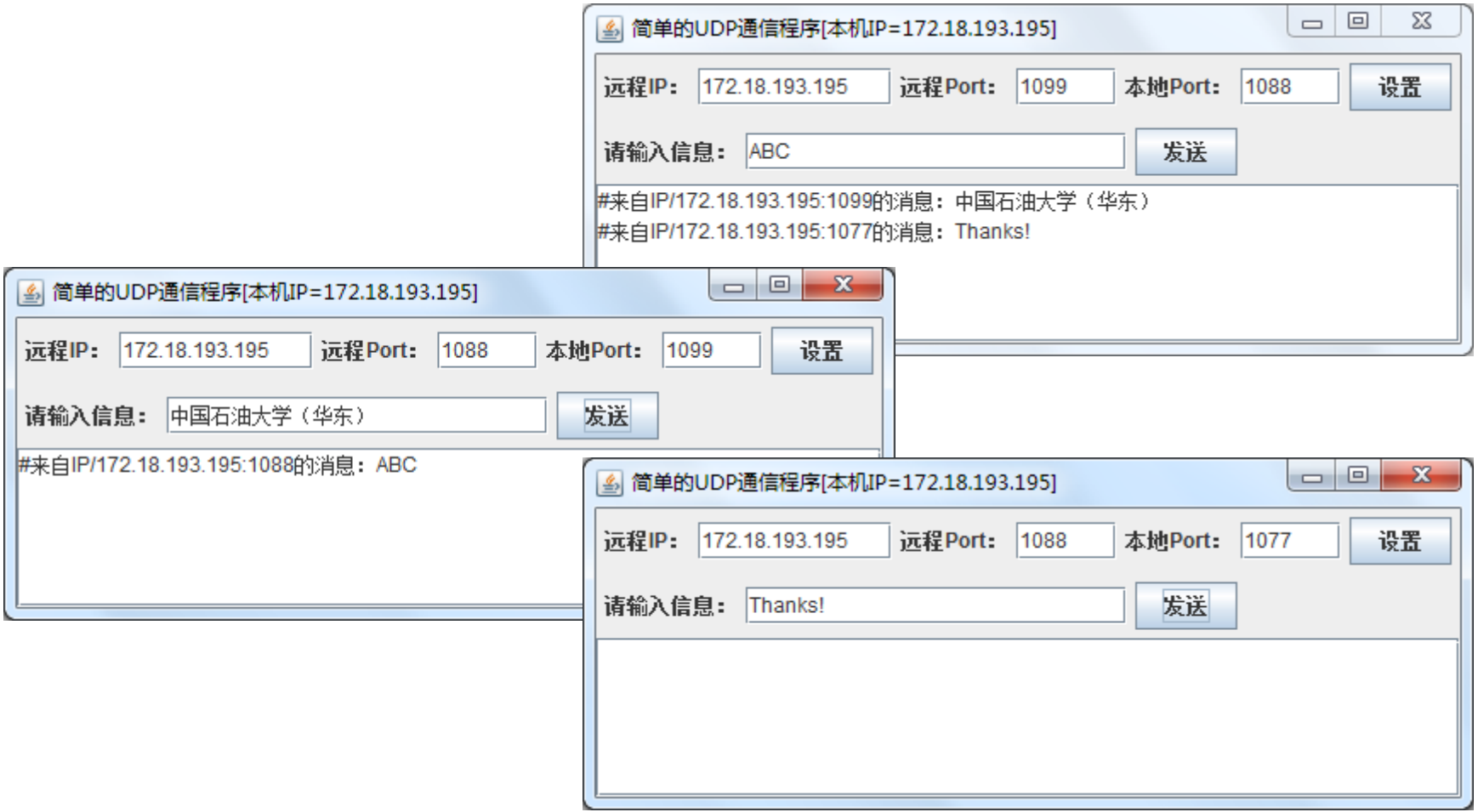
    socket.close()

基于UDP的Socket通信应用

- “服务端”程序用于接收数据，多个相同的“客户端”程序同时运行，用于发送数据。基于UDP，并不是正真的C/S模式!
- 同一个基于UDP的程序在不同计算机之间相互通信，或者在它同一计算机上运行的多个进程相互通信（绑定到不同的端口）。

基于UDP的Socket通信应用

【例12-13】同一个UDP程序之间相互发送文本消息，具有GUI，收发消息之前先设置好远程主机的IP地址和端口号，以及本地主机的端口号。



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

public class UDP extends JFrame  implements ActionListener{
   // 成员变量
   private int localPort=1088;
   private int remotePort=1099;
   private String remoteIP="127.0.0.1";
   private DatagramSocket socket=null;
   // 界面中的组件
   private JLabel lbl1,lbl2,lbl3,lbl4;
   private JTextField,txtRemoteIP,txtRemotePort,txtLocalPort,txtSendMsg;
   private JButton btnSet,btnSend;
   private JTextArea taRecvMsg;
   private JPanel p,p2,p3;
   private JScrollPane sp;
```

```
public UDP(){ // 窗体构造方法
        super("简单的UDP通信程序");
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        try {     String localIP=InetAddress.getLocalHost().getHostAddress();
                  this.setTitle(getTitle()+"[本机IP="+localIP+"]");
                  remoteIP=localIP; // 默认本机作为远程主机
        } catch (UnknownHostException e) {   e.printStackTrace();     }

        p1=new JPanel(new FlowLayout(FlowLayout.LEFT));
        lbl1=new JLabel("远程IP：");
        lbl2=new JLabel("远程Port：");
        lbl3=new JLabel("本地Port：");
        txtRemoteIP=new JTextField(remoteIP,10);
        txtRemotePort=new JTextField(Integer.toString(remotePort),5);
        txtLocalPort=new JTextField(Integer.toString(localPort),5);
        btnSet=new JButton("设置");
        btnSet.setActionCommand("设置"); // 设置动作命令
        btnSet.addActionListener(this); // 添加动作事件监听器
        p1.add(lbl1);p1.add(txtRemoteIP);
        p1.add(lbl2);p1.add(txtRemotePort);
        p1.add(lbl3);p1.add(txtLocalPort);
        p1.add(btnSet);
```

```
        p2=new JPanel(new FlowLayout(FlowLayout.LEFT));
        lbl4=new JLabel("请输入信息： ");
        txtSendMsg=new JTextField("",20);
        btnSend=new JButton("发送");
        btnSend.setActionCommand("发送"); // 设置动作命令
        btnSend.addActionListener(this); // 添加动作事件监听器
        btnSend.setEnabled(false);
        p2.add(lbl4);p2.add(txtSendMsg);
        p2.add(btnSend);

        p3=new JPanel(new GridLayout(2,1));
        p3.add(p1);p3.add(p2);

        taRecvMsg=new JTextArea(5, 10);
        sp=new JScrollPane(taRecvMsg);

        this.add(p3, "North");
        this.add(sp, "Center");
        this.pack();
}
```

```
public void setting(){ // 设置远程IP、端口号及本地端口号
        remoteIP=txtRemoteIP.getText().trim();
        remotePort=Integer.parseInt(txtRemotePort.getText().trim());
        localPort=Integer.parseInt(txtLocalPort.getText().trim());
        try {   if(socket!=null) socket.close(); //如果之前创建了数据报
  Socket，则先关闭它
                socket=new DatagramSocket(localPort); // 数据报Socket
                JOptionPane.showMessageDialog(this, "设置成功",
                        "提示", JOptionPane.INFORMATION_MESSAGE);
                btnSend.setEnabled(true);
        } catch (SocketException e1) { e1.printStackTrace();  }
  }
public void sendPacket (String msg){ // 发送数据报
        try {     byte[] buff=msg.getBytes();
                // 创建用于发送数据的数据报
                DatagramPacket pack=new DatagramPacket(buff,
            buff.length, InetAddress.getByName(remoteIP), remotePort);
                socket.send(pack); // 发送数据报
        } catch (IOException e1) { e1.printStackTrace(); }
  }
```

```
    public void actionPerformed(ActionEvent e) {
        String cmd=e.getActionCommand();
        if(cmd=="设置"){
            this.setting();
            //创建接收数据报的线程
            Thread receiveThead = new
              Thread(new ReceivePacketThread(socket, taRecvMsg));
            receiveThead.start();
        }else if(cmd=="发送"){
            String msg=txtSendMsg.getText();
            this.sendPacket(msg);
        }
    }
    public static void main(String[] args) {
        UDP frm=new UDP();
        frm.setVisible(true);
    }
}
```

专门用于接收数据报的线程类

```
import java.io.*;
import java.net.*;
import javax.swing.*;
public class ReceivePacketThread implements Runnable {
    private DatagramSocket socket;
     private DatagramPacket pack; // 用于接收数据的数据报
    private JTextArea taRecvMsg;
    private byte[] rBuff=new byte[1024]; // 接收消息的缓冲区

    public ReceivePacketThread(DatagramSocket socket, JTextArea
    taRecvMsg) {
            this.socket = socket;
            this.taRecvMsg = taRecvMsg;
            pack=new DatagramPacket(rBuff, rBuff.length); // 创建数据报对象
    }

   public void run() {
            try {
                    while(true)       receivePacket();  // 循环接收消息
            } catch (IOException e) {  e.printStackTrace(); }
    }
```

```
public void receivePacket() throws IOException{ // 接收数据报

        socket.receive(pack); // 接收数据报

       // 获取发送数据报的主机的IP地址、端口号
        InetAddress IPAddr=pack.getAddress();
        int port =pack.getPort();

        // 从数据报中获取消息
        String msg, line;
        msg=new
                String(pack.getData(), pack.getOffset(), pack.getLength());
        line="#来自IP"+IPAddr+":"+port+"的消息： "+msg+"\n";
        taRecvMsg.append(line);
   }

} // end of class ReceivePacketThread
```

基于UDP的Socket多播通信

- MulticastSocket类是DatagramSocket的一个子类，既可以将数据报（即DatagramPacket对象）发送到多播地址，也可以接收其他主机的多播的数据报。
- 要使用多播，则需要让一个数据报标有一个多播地址。当数据报发出后，加入到该多播组的所有主机都能收到该数据报。
- IP中的D类IP地址（224.0.0.1~239.255.255.255）就是组播地址，用于多点多播。D类IP地址不能分配给某个特定的主机，其中的每一个地址代表了一组主机。
- 加入到同一多播组的主机，既可以在某个端口上多播数据报，也可以在该端口上接收数据报。多播消息的主机也可以不加入多播组，但此时它就无法接收其他主机多播的消息 了。
- 多播数据报中的IP地址是多播地址，端口是多播目的端口，也就是多播组内的主机只有在这个端口上才能接收到该多播数据报。

基于UDP的Socket多播通信

- **MulticastSocket类的主要方法**
  - **MulticastSocket**() throws IOException
    使用本机默认IP、随机端口创建多播套接字
  - **MulticastSocket**(int port) throws IOException
    使用本机默认IP、指定端口创建多播套接字
  - void **joinGroup**(InetAddress mcastaddr) throws IOException 加入多播组
  - void **leaveGroup**(InetAddress mcastaddr) throws IOException 离开多播组
  - 接收和发送数据报的方法**send**、**receive**与其父类**DatagramSocket**完全相同离开多播组

基于UDP的Socket多播通信

- **MulticastSocket类的主要方法**
  - void **setTimeToLive**(int ttl) throws IOException

    设置在此 MulticastSocket 上发出的多播数据报的默认生存时间，以便控制多播的范围。ttl 必须在 0 <= ttl <= 255 范围内，否则将抛出IllegalArgumentException
    - 当ttl=0时，指定数据报应停留在本机
    - 当ttl=1（默认值）时，指定数据报发送到本地局域网
    - 当ttl=32时，指定数据报只能发送到本站点的网络上
    - 当ttl=64时，指定数据报应保留在本地区
    - 当ttl=128时，指定数据报应保留在本大洲
    - 当ttl=255时，指定数据报可发送到所有地方

基于UDP的Socket多播通信

- 用MulticastSocket类编写多播通信程序的步骤
  - **Step 1**：建立一个多播套接字对象
    MulticastSocket socket = new MulticastSocket(int port)
  - **Step 2**：加入多播组
    InetAddress group=InetAddress.getByName("230.0.0.1");//多播地址
    socket.joinGroup(group);
  - **Step 3**：创建用于多播接收或发送的数据报
    DatagramPacket sendPacket=new
    DatagramPacket (byte[] buf, int length)
    或者：DatagramPacket recvPacket=new DatagramPacket (
    byte[] buf, int length, InetAddress multicastAddr, int multicastPort)
  - **Step 4**：调用所创建的MulticastSocket对象的receive或send方法来接收或发送多播数据报
    socket.receive(recvPacket) 或者： socket.send(sendPacket)
  - **Step 5**：通信结束后，关闭多播套接字对象，并离开多播组
    socket.close();
    socket.leaveGroup(group);

通常建立一个线程，
专门用于接收数据报

基于UDP的Socket多播通信应用

【例12-14】同一个多播程序之间相互发送文本消息，具有GUI，收发消息之前先设置好多播IP地址和端口号，通信的所有站点都加入到多播组内。

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

public class Multicast extends JFrame  implements ActionListener{
   // 成员变量
   private String multicastIP="230.0.0.1"; //多播IP地址
   private InetAddress group; // 多播地址
   private int multicastPort=3000; // 多播端口
   private MulticastSocket socket=null;
   // 界面中的组件
   private JLabel lbl1,lbl2,lbl4;
   private JTextField txtMulticastIP,txtMulticastPort,txtSendMsg;
   private JButton btnSet,btnSend;
   private JTextArea taRecvMsg;
   private JPanel p,p2,p3;
   private JScrollPane sp;
```

```
public Multicast(){ // 窗体构造方法
        super("简单的多播通信程序");
        try {  String localIP=InetAddress.getLocalHost().getHostAddress();
               this.setTitle(getTitle()+"[本机IP="+localIP+"]");
        } catch (UnknownHostException e) {   e.printStackTrace(); }

        p1=new JPanel(new FlowLayout(FlowLayout.LEFT));
        lbl1=new JLabel("多播IP：");
        lbl2=new JLabel("多播Port：");
        txtMulticastIP=new JTextField(multicastIP,10);
        txtMulticastPort=new JTextField(Integer.toString(multicastPort),5);
        btnSet=new JButton("设置");
        btnSet.setActionCommand("设置"); // 设置动作命令
        btnSet.addActionListener(this); // 添加动作事件监听器
        p1.add(lbl1);p1.add(txtRemoteIP);
        p1.add(lbl2);p1.add(txtRemotePort);
        p1.add(lbl3);p1.add(txtLocalPort);
        p1.add(btnSet);
```

```
        p2=new JPanel(new FlowLayout(FlowLayout.LEFT));
        lbl4=new JLabel("请输入信息： ");
        txtSendMsg=new JTextField("",20);
        btnSend=new JButton("发送");
        btnSend.setActionCommand("发送"); // 设置动作命令
        btnSend.addActionListener(this); // 添加动作事件监听器
        btnSend.setEnabled(false);
        p2.add(lbl4);p2.add(txtSendMsg);
        p2.add(btnSend);

        p3=new JPanel(new GridLayout(2,1));
        p3.add(p1);p3.add(p2);

        taRecvMsg=new JTextArea(5, 10);
        sp=new JScrollPane(taRecvMsg);

        this.add(p3, "North");
        this.add(sp, "Center");
        this.pack();
}
```

```
public void setting(){ // 设置多播IP、端口号
        multicastIP=txtMulticastIP.getText().trim();
        multicastPort=Integer.parseInt(txtMulticastPort.getText().trim());
        try {     if(socket!=null) { //如果之前创建了多播Socket
                        socket.leaveGroup(group); //离开多播组
                        socket.close(); // 关闭socket
                }
                socket=new MulticastSocket(multicastPort);//多播Socket
                socket.setTimeToLive(1); // TTL=1--多播范围为本地LA
                group=InetAddress.getByName(multicastIP); // 多播地址
                socket.joinGroup(group); // 加入多播组group后,socket发
  送的数据报可以被该组成员接收到
                JOptionPane.showMessageDialog(this, "设置成功",
                        "提示", JOptionPane.INFORMATION_MESSAGE);
                btnSend.setEnabled(true);
        } catch (SocketException e1) {
                e1.printStackTrace();
        }
}
```

```
    public void sendPacket (String msg){ // 发送数据报
        try {     byte[] buff=msg.getBytes();
                  // 创建用于发送数据的数据报
                  DatagramPacket pack=new DatagramPacket(buff,
                                          buff.length, group, multicastPort);
                  socket.send(pack); // 发送数据报
        } catch (IOException e1) { e1.printStackTrace(); }
    }
    public void actionPerformed(ActionEvent e) {
        String cmd=e.getActionCommand();
        if(cmd=="设置"){
                  this.setting();
                  Thread receiveThead = new   //创建接收数据报的线程
                    Thread(new ReceivePacketThread(socket, taRecvMsg));
                  receiveThead.start();
        }else if(cmd=="发送"){ this.sendPacket( txtSendMsg.getText() ); }
    }
    public static void main(String[] args) {
         Multicast frm=new Multicast();
         frm.setVisible(true);
    }
}
```

该线程与前一个例子中的完全相同！

小 结

1. 用URL类访问WWW站点上的资源，例如下载文件
2. 用ServerSocket与Socket类创建基于TCP的socket通信程序
3. 用DatagramSocket类和DatagramPacket类创建基于UDP的通信程序
4. 用MulticastSocket类和DatagramPacket类创建基于UDP的多播通信程序

习 题

a) 设计一个文件下载器，具有GUI界面，能输入文件所在的URL，下载时显示保存对话框供用户选择保存位置并输入文件名。提示：利用URL类读取URL资源，利用文件流将其保存到文件，参考【例12-4】。

b) 设计一个服务端程序和一个客户端程序，能进行简单的数据通信，具有GUI界面。服务器能接受多个客户端连接，实现并发通信。提示：

① 使用ServerSocket类和Socket类实现基于TCP的Socket通信

② 服务器采用多线程：每接受一个客户连接就创建一个Socket，然后建立一个线程，其中该Socket负责接收客户端信息，并自动给每条信息回复一个确认

③ 客户端：连接到服务器后，能送消息；建立一个线程用于接收服务端发送的消息；

④ （可选）服务端维护一个当前保持连接的客户端信息列表，能随时给某个指定客户发送消息