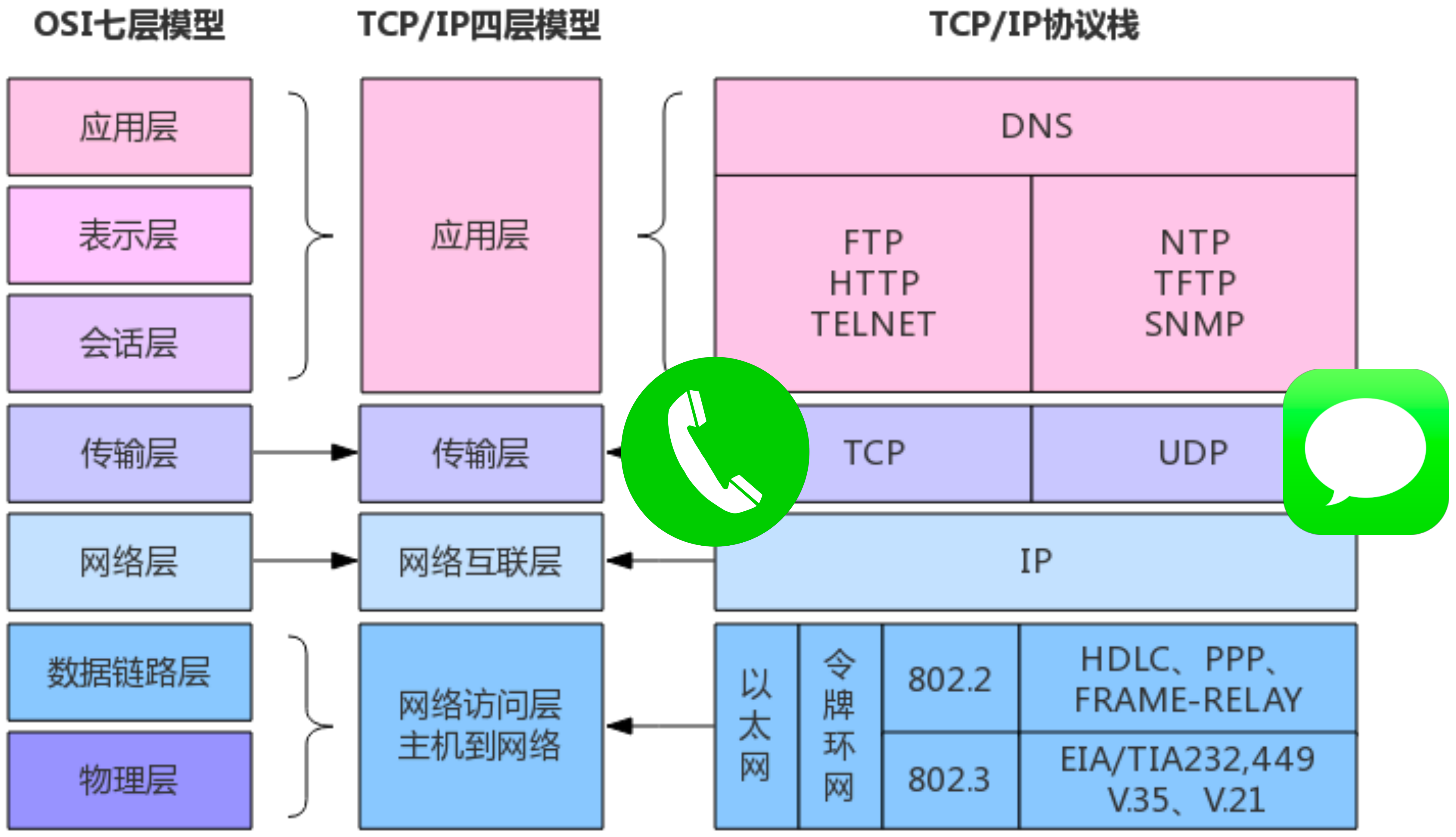
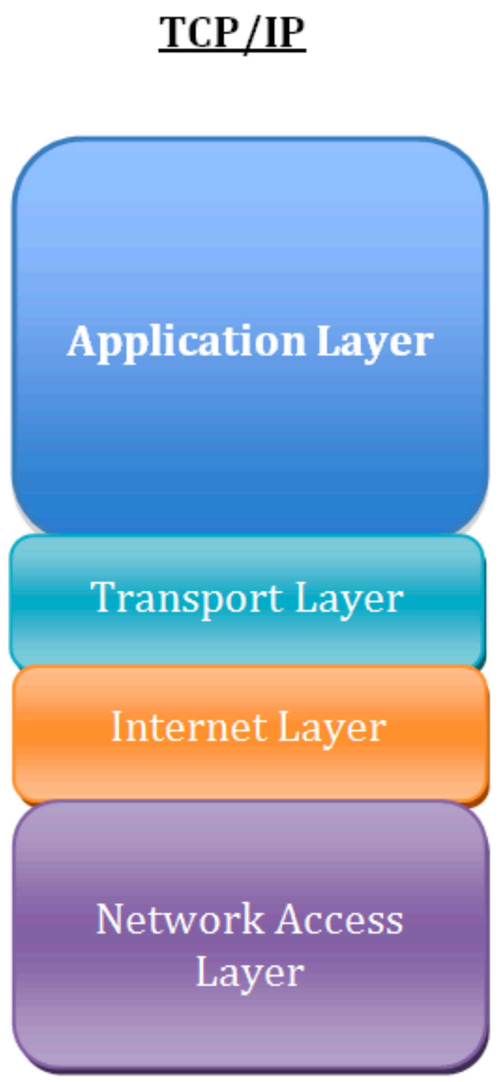
网络编程

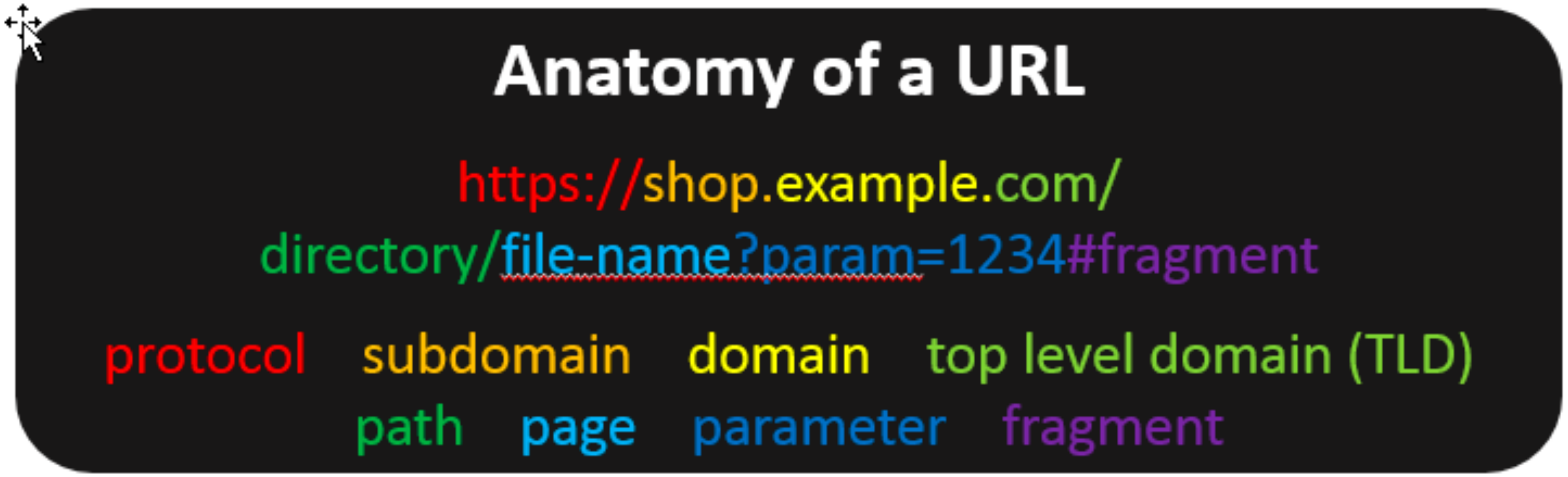
杨亮

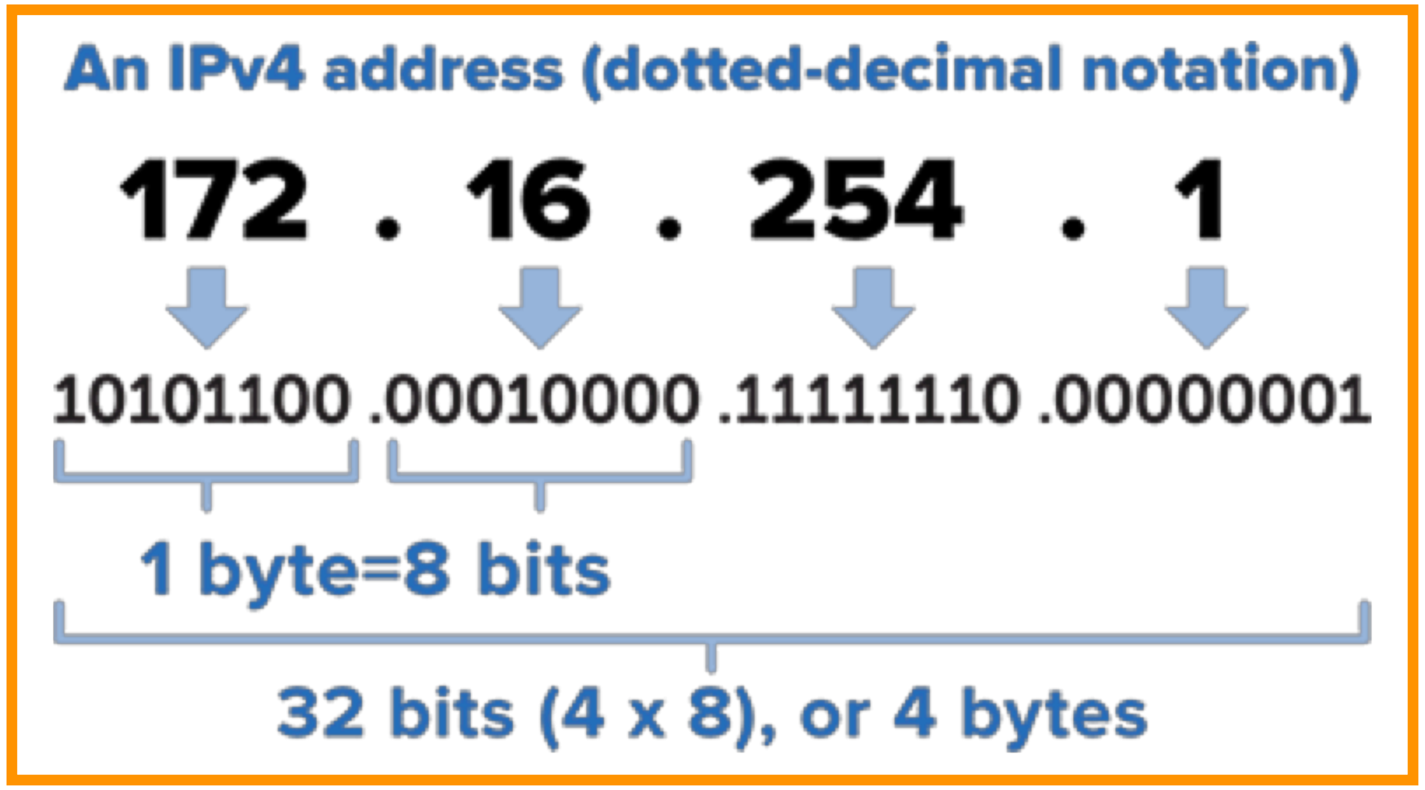
网络模型


OSI七层模型
TCP/IP四层模型
TCP/IP协议栈
应用层
表示层
会话层
传输层
网络层
数据链路层
物理层
应用层
传输层
网络互联层
网络访问层
主机到网络
DNS
FTP
HTTP
TELNET
NTP
TFTP
SNMP
TCP
UDP
IP
以太网
令牌环网
802.2
802.3
HDLC、PPP、FRAME-RELAY
EIA/TIA232,449
V.35、V.21

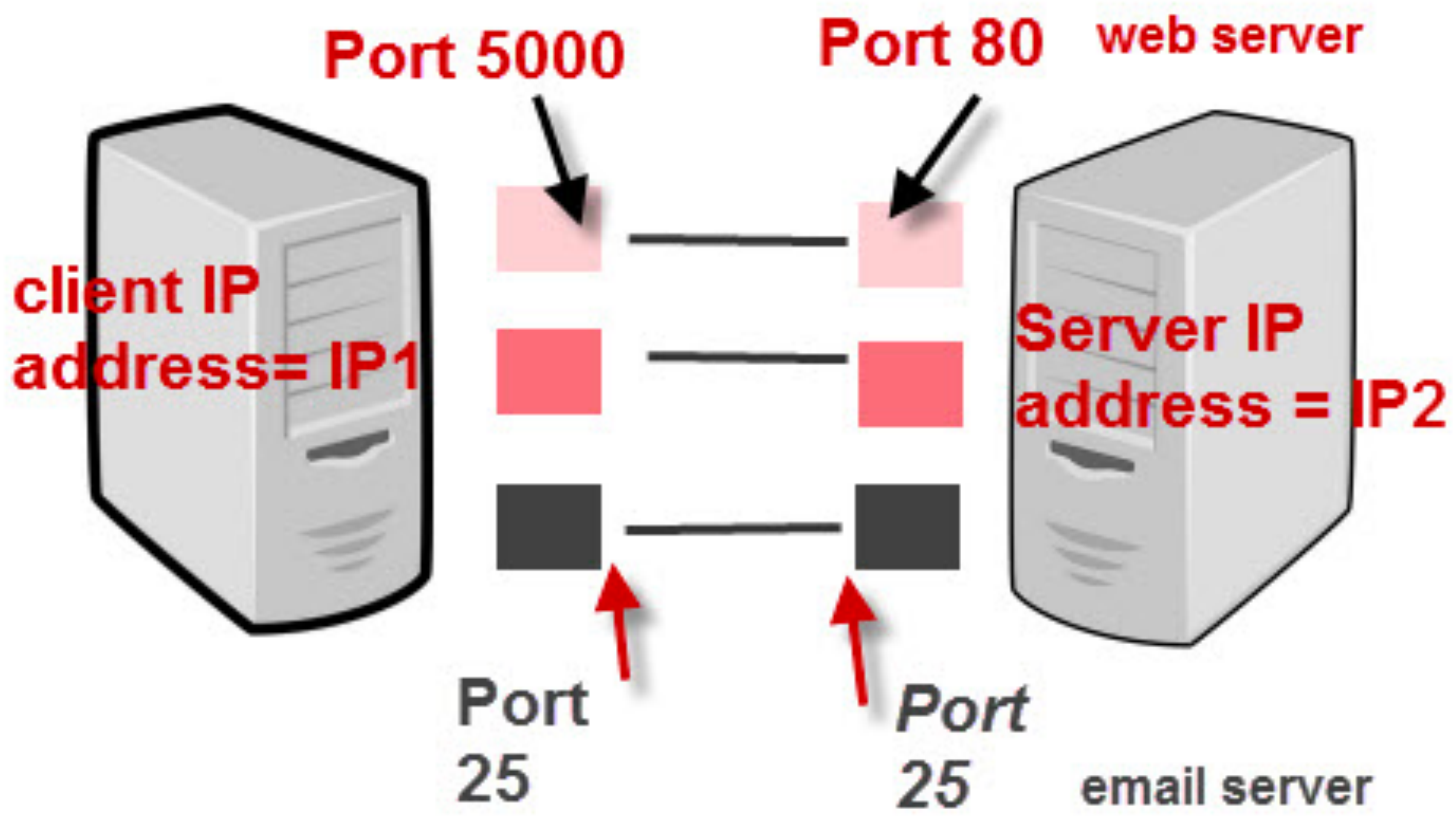
访问网络


TCP/IP
Application Layer
Transport Layer
Internet Layer
Network Access Layer


Anatomy of a URL
https://shop.example.com/directory/file-name?param=1234#fragment
protocol
subdomain
domain
top level domain (TLD)
path
page
parameter
fragment


An IPv4 address (dotted-decimal notation)
172 . 16 . 254 . 1
10101100 .00010000 .11111110 .00000001
1 byte=8 bits
32 bits (4 x 8), or 4 bytes

Socket



IP Address + Port number = Socket

TCP/IP Ports And Sockets

InetAddress 类

public static InetAddress getByName(String host)
public static InetAddress getLocalHost()
public static InetAddress[] getAllByName(String host)

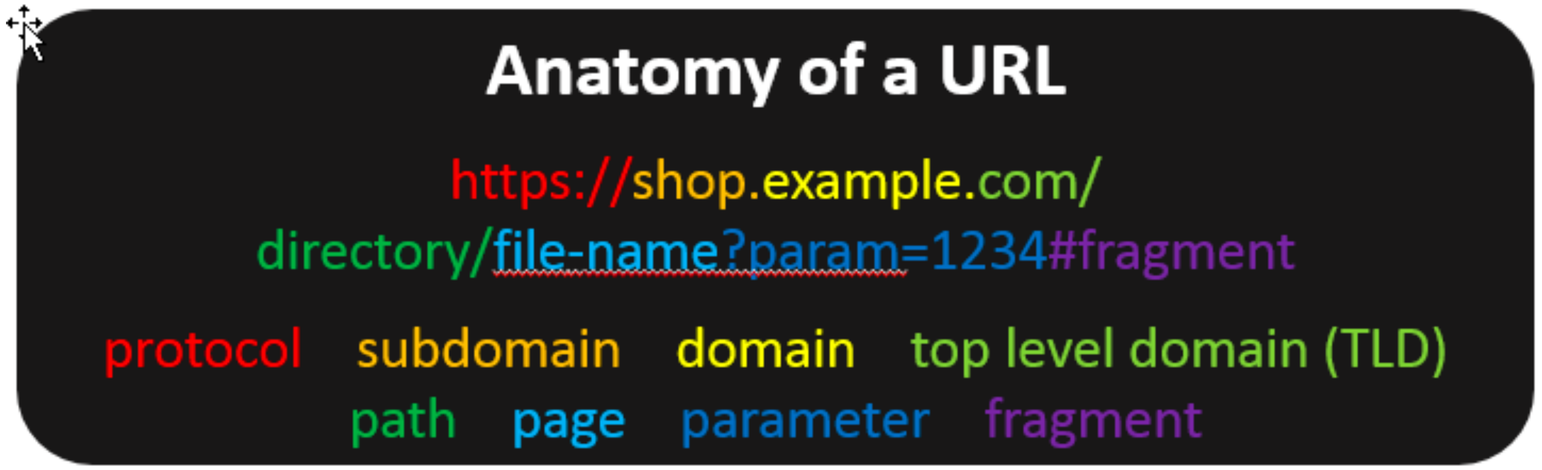
```
public class OreillyByName {
    public static void main(String[] args) {
        try {
            InetAddress address = InetAddress.getByName("www.oreilly.com");
            System.out.println(address.getHostAddress());
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }
    }
}
```

```
public class ByName {
    public static void main(String[] args) {
        try {
            InetAddress address = InetAddress.getByName("208.201.239.37");
            System.out.println(address.getHostName());
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }
    }
}
```

URL 类

- public URL(String spec)
- public URL(URL context, String spec)
- public URL(String protocol, String host, String file)
- public URL(String protocol, String host, int port, String file)

```
public class URLDemo
{
   public static void main(String [] args)
   {
      try
      {
         URL url = new URL("http://www.runoob.com/index.html?language=cn#j2se");
         System.out.println("URL 为：" + url.toString());
         System.out.println("协议为：" + url.getProtocol());
         System.out.println("验证信息：" + url.getAuthority());
         System.out.println("文件名及请求参数：" + url.getFile());
         System.out.println("主机名：" + url.getHost());
         System.out.println("路径：" + url.getPath());
         System.out.println("端口：" + url.getPort());
         System.out.println("默认端口：" + url.getDefaultPort());
         System.out.println("请求参数：" + url.getQuery());
         System.out.println("定位位置：" + url.getRef());
      }catch(IOException e)
      {
         e.printStackTrace();
      }
   }
}
```



URLConnection

URL对象的openConnection()方法

```java
public class URLConnDemo
{
   public static void main(String [] args)
   {
      try
      {
         URL url = new URL("http://www.runoob.com");
         URLConnection urlConnection = url.openConnection();

         BufferedReader in = new BufferedReader(
         new InputStreamReader(connection.getInputStream()));
         String urlString = "";
         String current;
         while((current = in.readLine()) != null)
         {
            urlString += current;
         }
         System.out.println(urlString);
      }catch(IOException e)
      {
         e.printStackTrace();
      }
   }
}
```

URL对象的openstream()方法

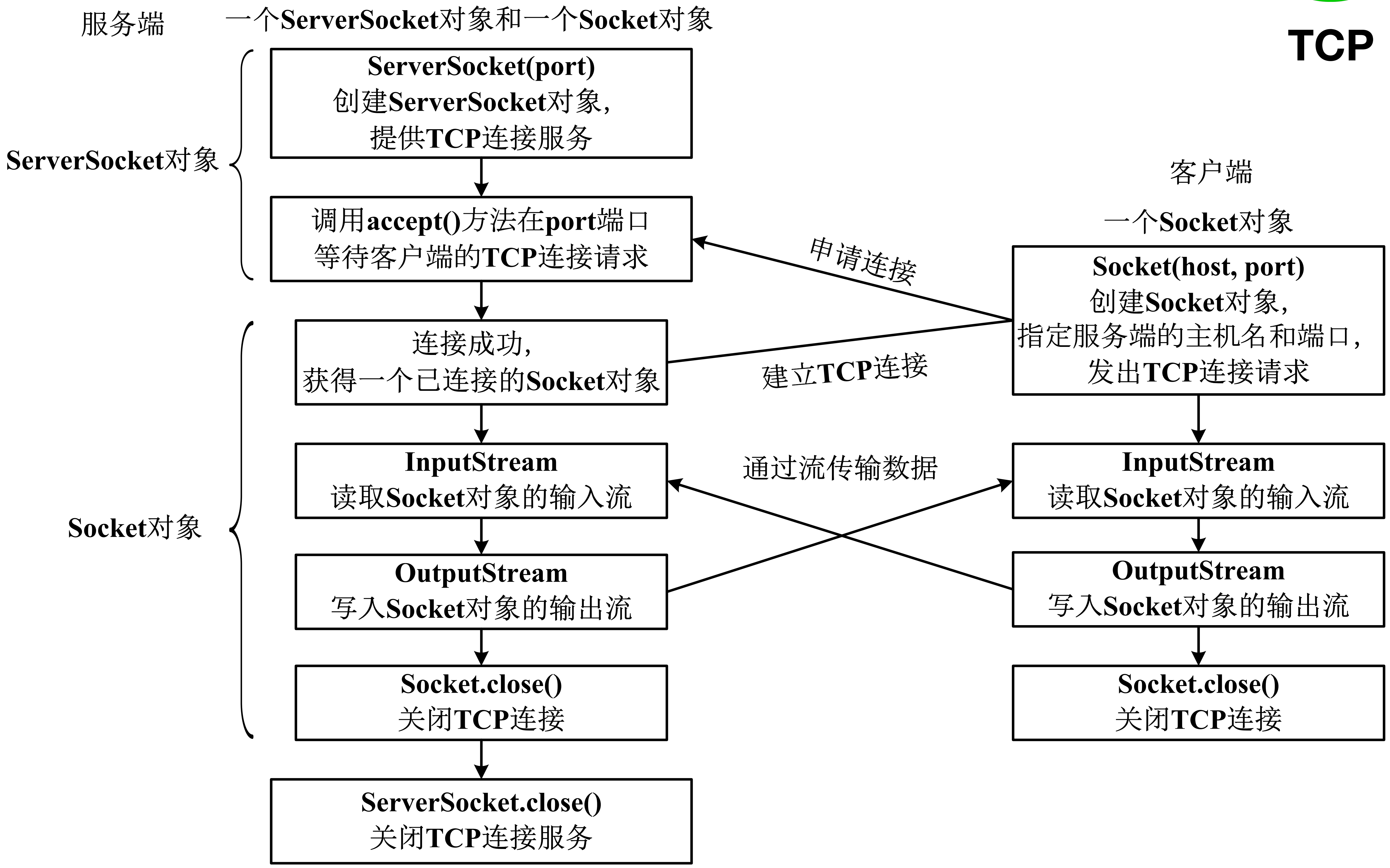
```java
public final InputStream openStream() throws java.io.IOException {
        return openConnection().getInputStream();
}
```

||

URLConnection对象的getInputStream()方法

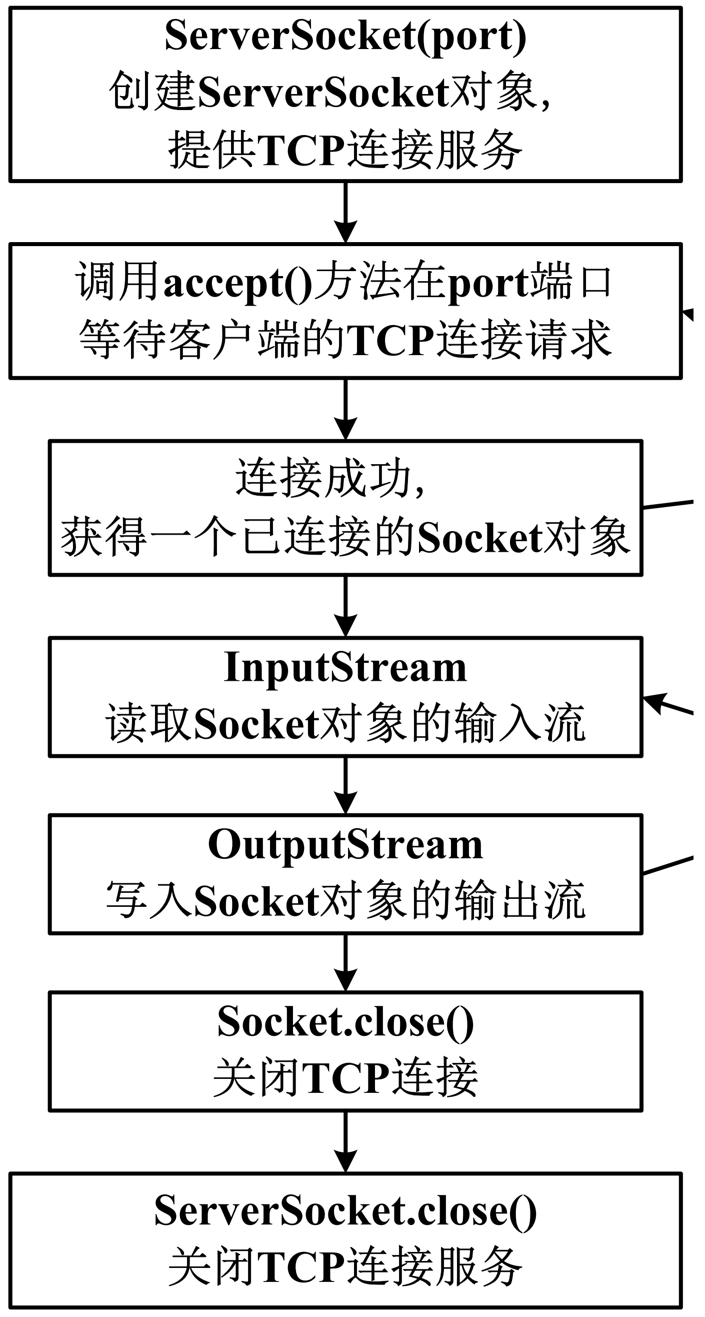
socket中包含了所有的地址信息

TCP



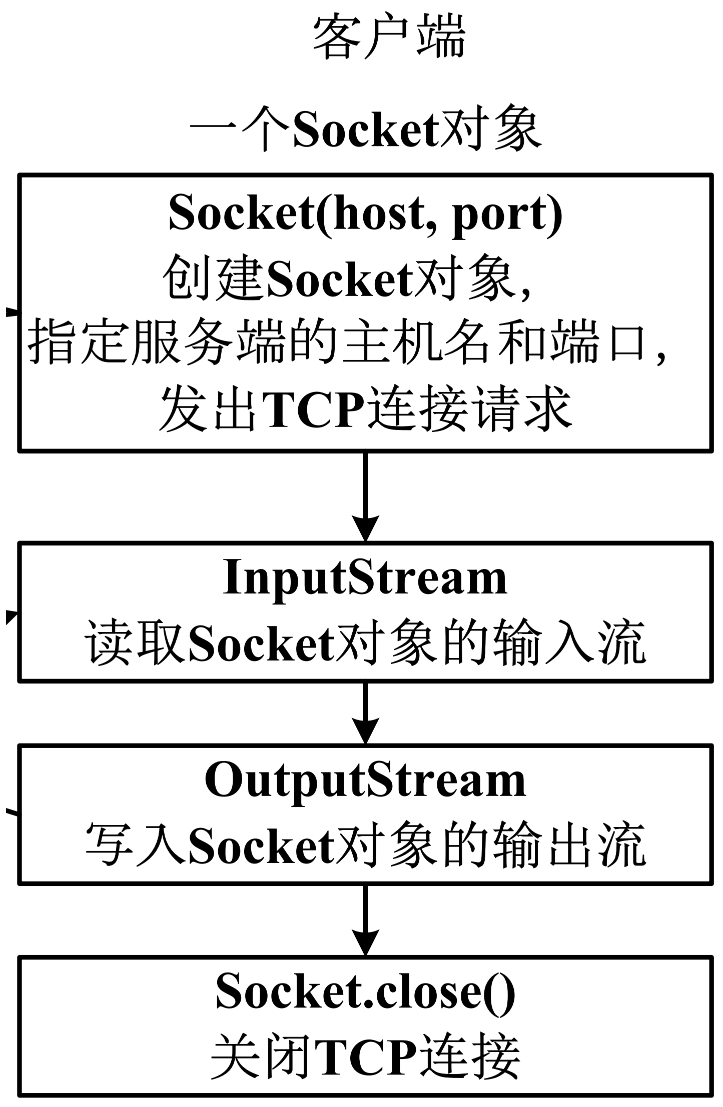
```
public class SocketServer {
  public static void main(String[] args) throws Exception {
    // 监听指定的端口
    int port = 55533;
    ServerSocket server = new ServerSocket(port);

    // server将一直等待连接的到来
    System.out.println("server将一直等待连接的到来");
    Socket socket = server.accept();
    // 建立好连接后，从socket中获取输入流，并建立缓冲区进行读取
    InputStream inputStream = socket.getInputStream();
    byte[] bytes = new byte[1024];
    int len;
    StringBuilder sb = new StringBuilder();
    //只有当客户端关闭它的输出流的时候，服务端才能取得结尾的-1
    while ((len = inputStream.read(bytes)) != -1) {
      // 注意指定编码格式，发送方和接收方一定要统一，建议使用UTF-8
      sb.append(new String(bytes, 0, len, "UTF-8"));
    }
    System.out.println("get message from client: " + sb);

    OutputStream outputStream = socket.getOutputStream();
    outputStream.write("Hello Client,I get the message.".getBytes("UTF-8"));

    inputStream.close();
    outputStream.close();
    socket.close();
    server.close();
  }
}
```

个ServerSocket对象和一个Socket对



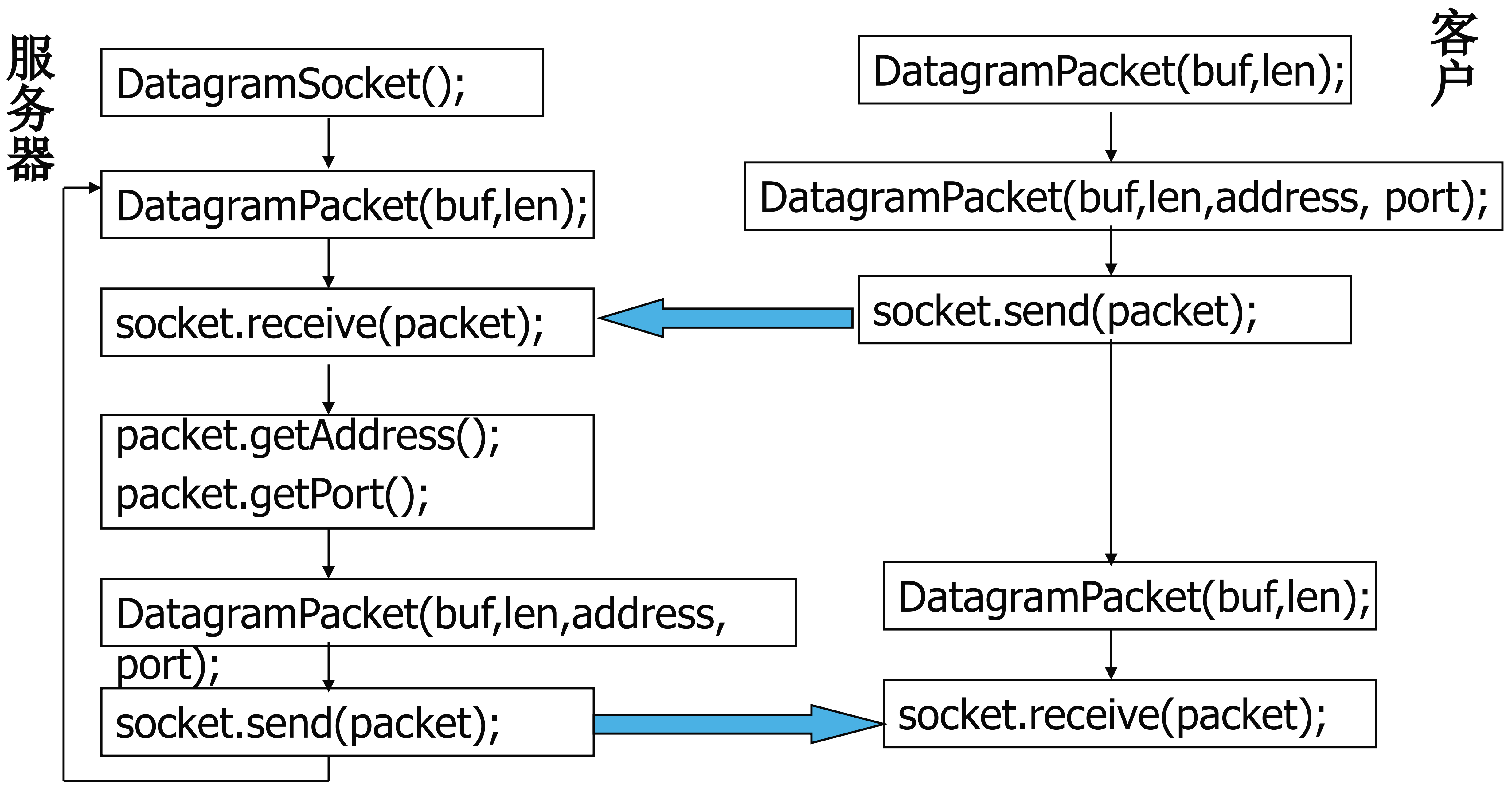
```java
public class SocketClient {
  public static void main(String args[]) throws Exception {
    // 要连接的服务端IP地址和端口
    String host = "127.0.0.1";
    int port = 55533;
    // 与服务端建立连接
    Socket socket = new Socket(host, port);
    // 建立连接后获得输出流
    OutputStream outputStream = socket.getOutputStream();
    String message = "你好  yiwangzhibujian";
    socket.getOutputStream().write(message.getBytes("UTF-8"));
    //通过shutdownOutput高速服务器已经发送完数据，后续只能接受数据
    socket.shutdownOutput();

    InputStream inputStream = socket.getInputStream();
    byte[] bytes = new byte[1024];
    int len;
    StringBuilder sb = new StringBuilder();
    while ((len = inputStream.read(bytes)) != -1) {
      //注意指定编码格式，发送方和接收方一定要统一，建议使用UTF-8
      sb.append(new String(bytes, 0, len,"UTF-8"));
    }
    System.out.println("get message from server: " + sb);

    inputStream.close();
    outputStream.close();
    socket.close();
  }
}
```

public Socket(String host, int port)
public Socket(InetAddress address, int port)

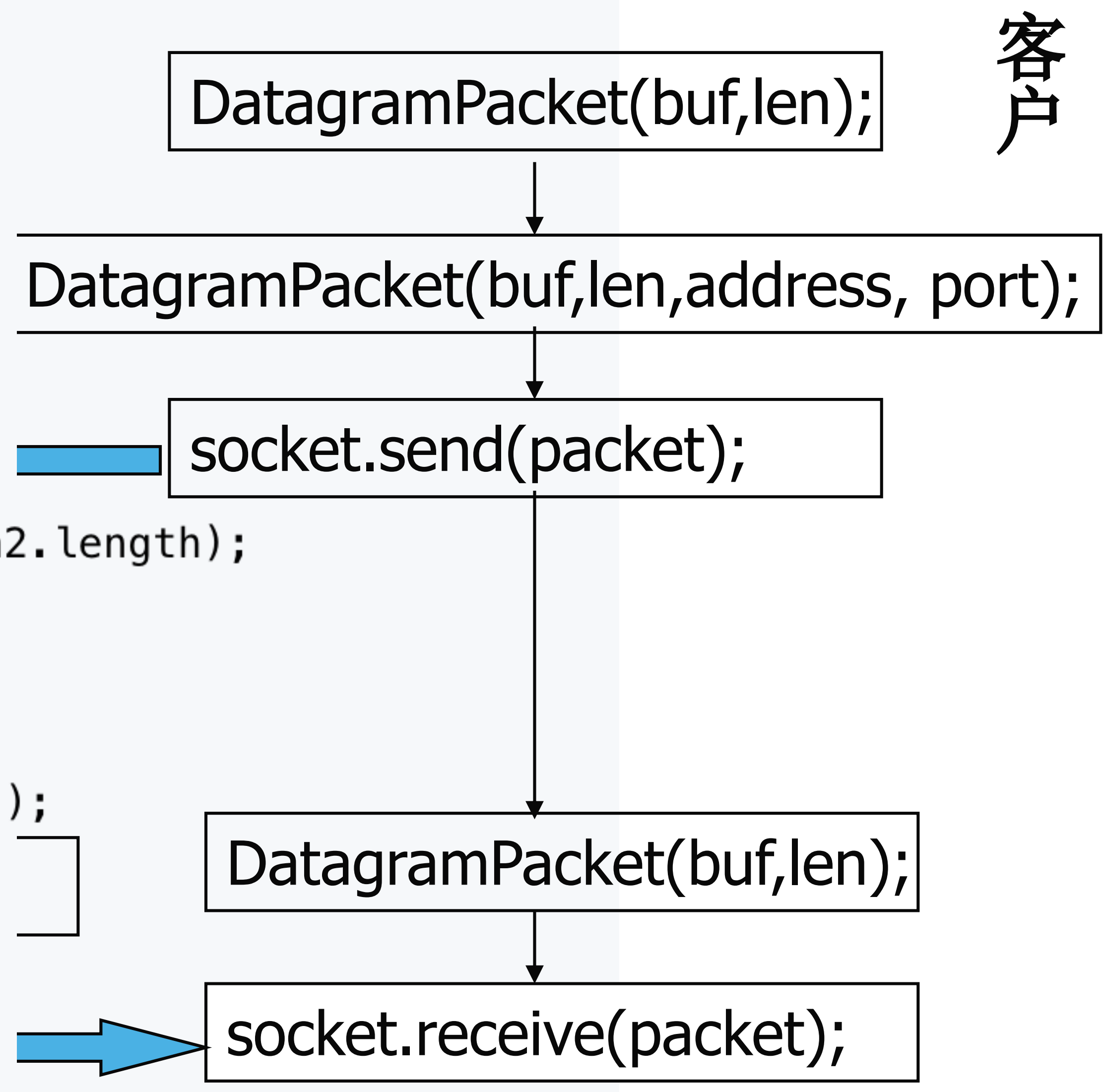


package中包含了所有的地址信息

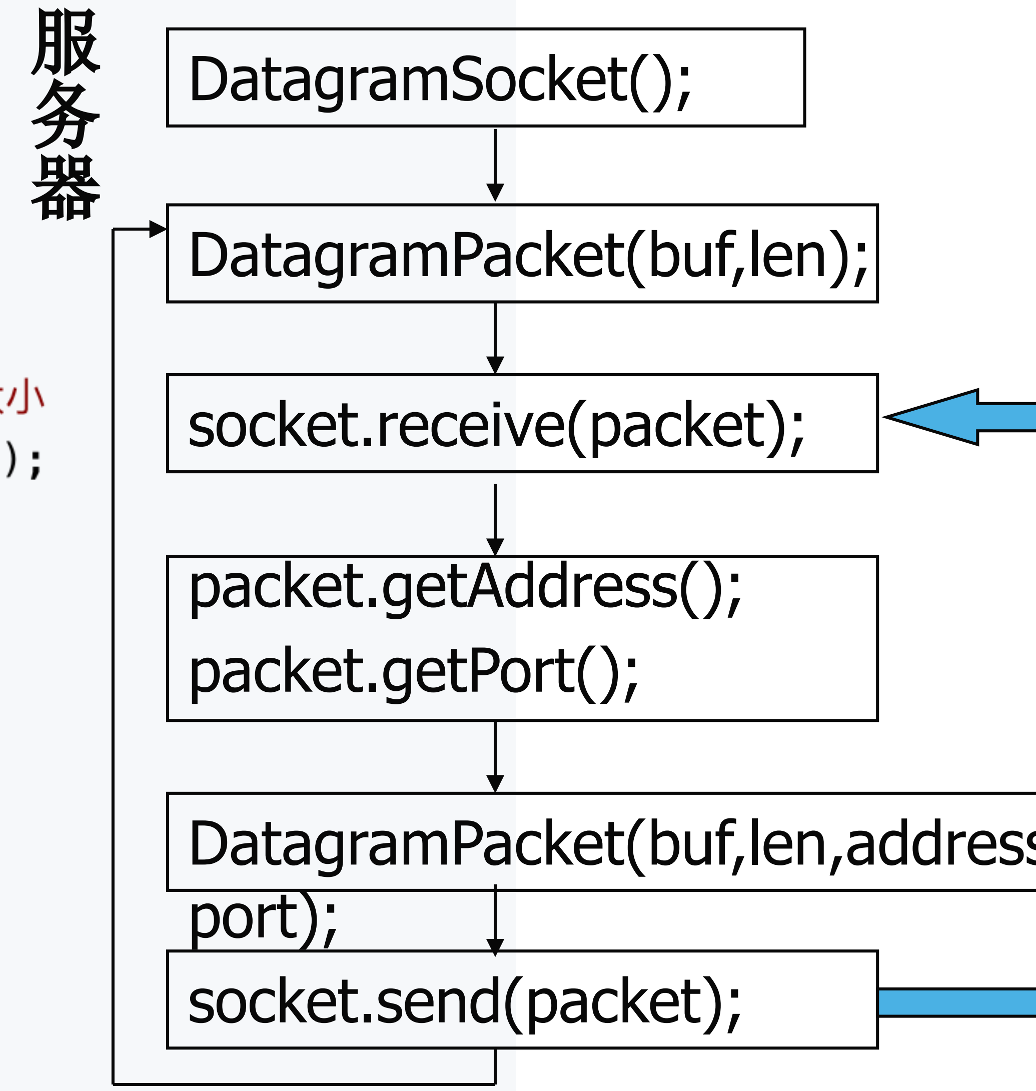
UDP



```java
public class UDPClient {
    public static void main(String[] args) throws IOException {
        /*
         * 向服务器端发送数据
         */
        // 1.定义服务器的地址、端口号、数据
        InetAddress address = InetAddress.getByName("localhost");
        int port = 8800;
        byte[] data = "用户名：admin;密码：123".getBytes();
        // 2.创建数据报，包含发送的数据信息
        DatagramPacket packet = new DatagramPacket(data, data.length, address, port);
        // 3.创建DatagramSocket对象
        DatagramSocket socket = new DatagramSocket();
        // 4.向服务器端发送数据报
        socket.send(packet);

        /*
         * 接收服务器端响应的数据
         */
        // 1.创建数据报，用于接收服务器端响应的数据
        byte[] data2 = new byte[1024];
        DatagramPacket packet2 = new DatagramPacket(data2, data2.length);
        // 2.接收服务器响应的数据
        socket.receive(packet2);
        // 3.读取数据
        String reply = new String(data2, 0, packet2.getLength());
        System.out.println("我是客户端，服务器说：" + reply);
        // 4.关闭资源
        socket.close();
    }
}
```



```
public class UDPServer {
    public static void main(String[] args) throws IOException {
        /*
         * 接收客户端发送的数据
         */
        // 1.创建服务器端DatagramSocket，指定端口
        DatagramSocket socket = new DatagramSocket(8800);
        // 2.创建数据报，用于接收客户端发送的数据
        byte[] data = new byte[1024];// 创建字节数组，指定接收的数据包的大小
        DatagramPacket packet = new DatagramPacket(data, data.length);
        // 3.接收客户端发送的数据
        System.out.println("****服务器端已经启动，等待客户端发送数据");
        socket.receive(packet);// 此方法在接收到数据报之前会一直阻塞
        // 4.读取数据
        String info = new String(data, 0, packet.getLength());
        System.out.println("我是服务器，客户端说：" + info);

        /*
         * 向客户端响应数据
         */
        // 1.定义客户端的地址、端口号、数据
        InetAddress address = packet.getAddress();
        int port = packet.getPort();
        byte[] data2 = "欢迎您!".getBytes();
        // 2.创建数据报，包含响应的数据信息
        DatagramPacket packet2 = new DatagramPacket(data2, data2.length, address, port);
        // 3.响应客户端
        socket.send(packet2);
        // 4.关闭资源
        socket.close();
    }
}
```


服务器
DatagramSocket();
DatagramPacket(buf,len);
socket.receive(packet);
packet.getAddress();
packet.getPort();
DatagramPacket(buf,len,address
port);
socket.send(packet);