



# 第11章 网络应用



# 网络编程技术

## 1. 网络协议概述

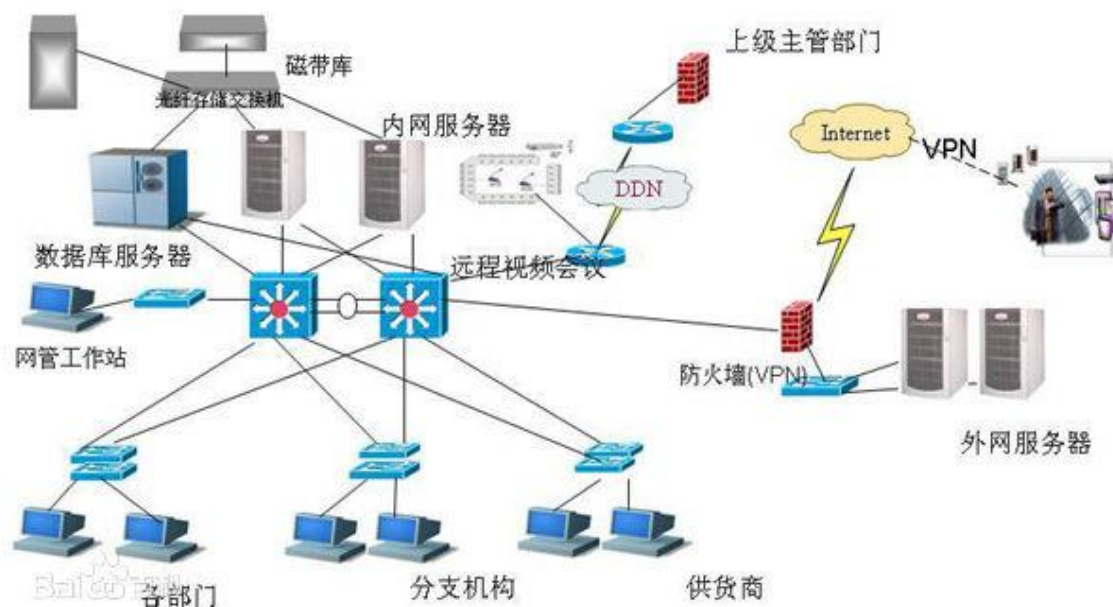
## 2. 网络类及应用

- InetAddress类应用
- ServerSocket类与Socket类实现TCP通信
- DatagramSocket类与Datagram类实现UDP通信

# 1. 网络协议概述

## 计算机网络

► 计算机网络，是指将地理位置不同的具有独立功能的多台计算机及其外部设备，通过通信线路连接起来，在网络操作系统，网络管理软件及网络通信协议的管理和协调下，实现资源共享和信息传递的计算机系统。



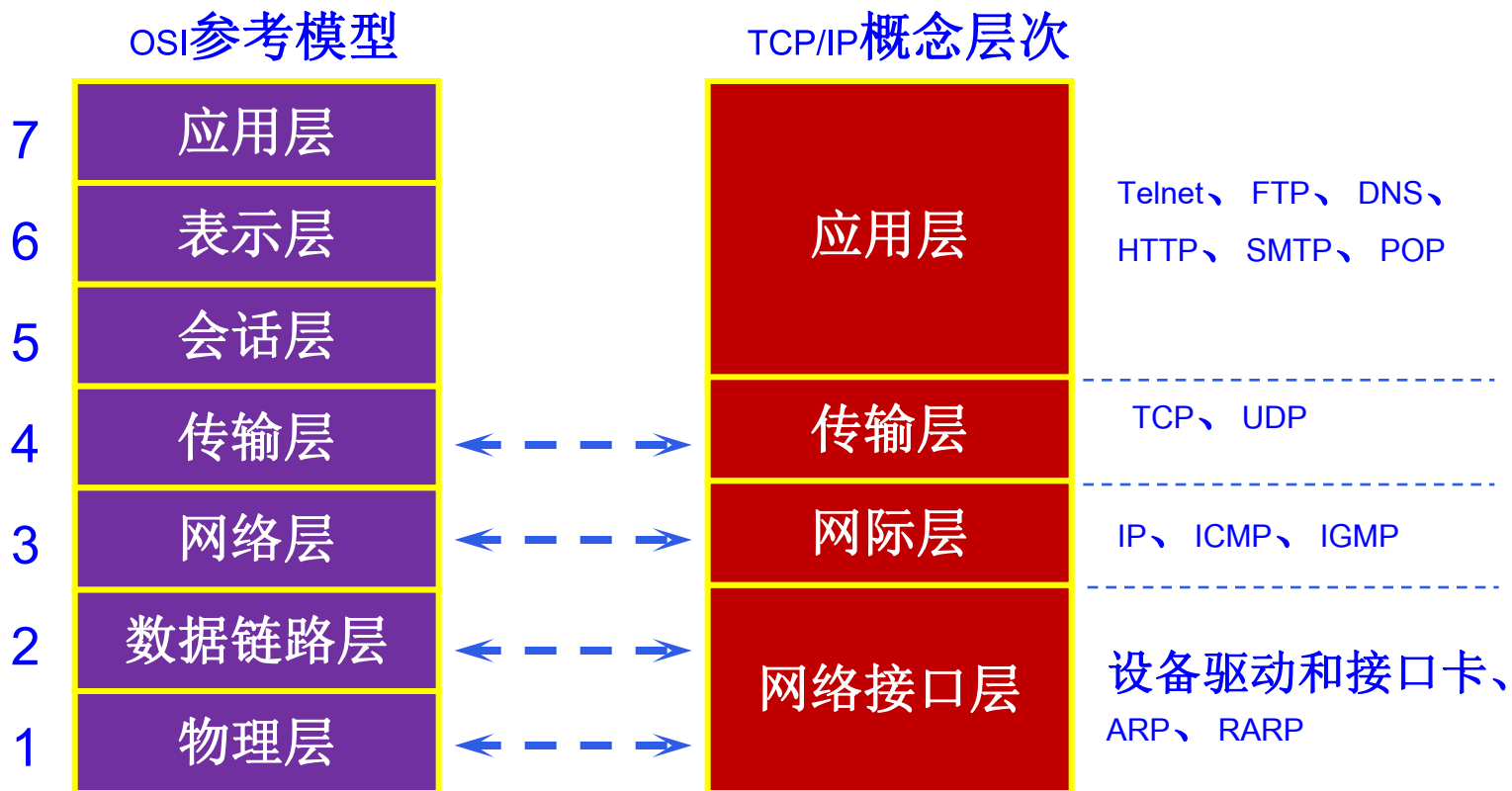


# 1. 网络协议概述

- ❖ **网络通信**是采用网络协议实现计算机之间的数据交换。
- ❖ 所谓**网络协议**是指通讯双方约定好的规则集合。
- ❖ 国际标准化组织提出开放系统互连模型，即**ISO/OSI**七层协议模型
- ❖ 传输控制协议/互联网络协议，即**TCP/IP**四层协议模型起源于美国国防部高级研究规划署(ARPA)的一项研究计划——实现若干台主机的相互通信。



# ISO/OSI与TCP/IP的对应关系







# 1. 网络协议概述

- ❖ **TCP**是面向连接的、可靠的协议；**UDP**是面向非连接、不可靠的连接。
- ❖ 一个网络连接是一个**5元组**：  
(协议名称，本地地址，本地端口，远程地址，远程端口)
- ❖ 通过**端口**确定通信进程，端口取值范围**0~65535**，其中**0~1023**为系统保留端口。



# 1. 网络协议概述

- ❖ **Java**中与网络通信有关的类都在**java.net**包中。
  - ⌘ 基于**TCP**传输协议的类有**URL**、**URLConnection**、**Socket**和**ServerSocket**。
  - ⌘ 基于**UDP**传输协议的类有**DatagramPacket**、**DatagramSocket**和**MulticastSocket**。



## 2. 网络类及应用

- ◆ InetAddress类
- ◆ ServerSocket类
- ◆ Socket类
- ◆ DatagramSocket类
- ◆ DatagramPacket类





# (1) InetAddress类

- ❖ Java使用InetAddress类表示一个32位或者128位的IP地址。
  - Inet4Address, Inet6Address
- ❖ 继承关系
  - java.lang.Object→java.net.InetAddress
- ❖ 常用成员方法
  - **getLocalHost()**: 返回本地主机的主机名/IP地址
  - **getByName()**: 返回主机名/IP地址
  - **getHostName()**: 根据IP地址返回主机名
  - **getHostAddress()**: 返回IP地址字符串



输入的主机名称mail.jlu.edu.cn  
主机mail.jlu.edu.cn为: mail.jlu.edu.cn/202.198.16.56  
程序获取的主机名称mail.jlu.edu.cn  
程序获取的IP地址为202.198.16.56  
本机为: H1-THINK/192.168.1.100  
本机的主机名称为: H1-THINK  
本机的IP地址为: 192.168.1.100

```
    InetAddress address = InetAddress.getByName(args[0]);  
    System.out.println("输入的主机名称" + args[0]);  
    System.out.println("主机"+args[0]+"为: "+address);  
    System.out.println("程序获取的主机名称" + address.getHostName());  
    System.out.println("程序获取的IP地址为" + address.getHostAddress());  
  
    System.out.println("本机为: "+InetAddress.getLocalHost());  
    System.out.println("本机的主机名称为:  
"+InetAddress.getLocalHost().getHostName());  
    System.out.println("本机的IP地址为:  
"+InetAddress.getLocalHost().getHostAddress());  
}  
}
```



## 示例：获取主机的IP地址

**JDK中运行输出：**

```
>java InetAddressDemo mail.jlu.edu.cn
```

```
输入的主机名称mail.jlu.edu.cn  
主机mail.jlu.edu.cn为: mail.jlu.edu.cn/202.198.16.56  
程序获取的主机名称mail.jlu.edu.cn  
程序获取的IP地址为202.198.16.56  
本机为: H1-THINK/192.168.1.100  
本机的主机名称为: H1-THINK  
本机的IP地址为: 192.168.1.100
```

### 用Eclipse来运行命令行程序带参数的怎么办？

右击要运行的程序，选择debug as或者run as，然后选择debug/run configuration，然后在对话框左侧栏中，找到相应的程序，然后在上面的标签中点arguments，输入你希望的命令行参数，每行是一个参数。



## (2) ServerSocket类

- ❖ Java使用ServerSocket类实现服务器端的socket机制。
- ❖ 继承关系
  - `java.lang.Object`→`java.net.ServerSocket`
- ❖ 构造方法
  - `public ServerSocket() throws IOException`
  - `public ServerSocket(int port) throws IOException`
  - `public ServerSocket(int port, int backlog) throws IOException`
- ❖ 常用成员方法
  - `Socket accept()`: 侦听并接受到此套接字的连接
  - `void close()`: 关闭



## (3) Socket类

- ❖ Java使用Socket类实现客户端的socket机制。
- ❖ 继承关系
  - `java.lang.Object`→`java.net.Socket`
- ❖ 构造方法
  - `public Socket()`
  - `public Socket(String host,int port)throws UnknownHostException,IOException`
  - `public Socket(InetAddress address,int port)throws UnknownHostException,IOException`
- ❖ 常用成员方法
  - `InputStream getInputStream()`: 返回Socket输入流
  - `OutputStream getOutputStream()`: 返回Socket输出流
  - `void close()`: 关闭Socket



# 基于TCP的点对点通信

利用Socket开发一个Server-Client通信模型应用包括两个程序：

- ◆首先是**服务器程序Server**，它使用**ServerSocket**类监听指定端口，等待客户连接请求；当有客户请求时，处理请求，关闭连接。
- ◆其次是**客户机程序Client**，它使用**Socket**类对网络上某一个服务器某一个端口发出连接请求；一旦连接成功，打开会话；会话完成后，关闭**Socket**。



Java

# 使用ServerSocket建立一个简单的服务器

第1步：创建一个ServerSocket对象

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

第2步：调用Serversocket的accept()方法监听客户机

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

第3步：获得OutputStream和InputStream对象

```
ObjectInputStream input=new  
    ObjectInputStream(connection.getInputStream());  
  
ObjectOutputStream output=new  
    ObjectOutputStream(connection.getOutputStream());
```

第4步：服务器和客户机通过OutputStream和InputStream对象通信的处理阶段

第5步：如果传输结束，通过调用相关流的close()以及Socket的close()关闭连接





## 使用Socket建立一个简单的客户机

第1步：创建一个Socket用来连接到服务器上

```
Socket connection=new Socket(serverAddress,port);
```

第2步： Socket的getInputStream方法和getOutputStream方法分别用于获得与Socket相关联的InputStream和OutputStream的引用

```
ObjectInputStream input=new  
    ObjectInputStream(connection.getInputStream());  
  
ObjectOutputStream output=new  
    ObjectOutputStream(connection.getOutputStream());
```

第3步： 客户机与服务器通过InputStream 和OutputStream对象通信的处理阶段。

第4步： 如果传输结束，通过调用相关流的close()以及Socket的close()关闭连接。



服务端：

```
public class TCPServer {  
    public static void main(String[] args) throws IOException {  
        // 1.创建服务，指定服务端口  
        ServerSocket serverSocket = new ServerSocket(8081);  
  
        // 2.接受客户端的连接  
        Socket socket = serverSocket.accept();  
        System.out.println("服务端接受连接！");  
  
        // 3.发送数据  
        DataOutputStream dos = new DataOutputStream(new BufferedOutputStream(socket.getOutputStream()));  
        dos.writeUTF("你好！我是服务端！");  
        dos.flush();  
    }  
}
```

客户端：

```
public class TCPClient1 {  
    public static void main(String[] args) throws IOException {  
        // 1.创建服务，指定服务端口  
        Socket client = new Socket("localhost", 8081);  
  
        // 2.接收服务端数据  
        DataInputStream dis = new DataInputStream(new BufferedInputStream(client.getInputStream()));  
        String s = dis.readUTF();  
        System.out.println(s);  
    }  
}
```



# 基于TCP的点对面通信

- ❖ 在Java中，使用实现点对面通信的关键在于服务器端程序的设计。
- ❖ **服务器程序设计**步骤如下：
  - 服务器端程序监听指定端口，等待接收客户端的连接请求。
  - 同时构造一个线程类，准备接管客户端会话。
  - 当建立一个**Socket**会话产生后，将这个会话交给线程处理。
  - 服务器端组件继续监听指定端口。

```
while(true){  
    Socket socket = serverSocket.accept();  
    new ServerThead(socket).start();  
}
```



# 服务端程序

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

public class Server {
    public static final int PORT_NUM = 10000;

    ServerSocket serverSocket;

    public Server() throws IOException {
        serverSocket = new ServerSocket(PORT_NUM);
    }

    public void start() throws IOException {
        while (true) {
            Socket socket = serverSocket.accept();
            new ServerThead(socket).start(); // 创建服务线程
        }
    }

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



# 服务端程序

```
private class ServerThead extends Thread {  
    Socket socket;  
  
    ServerThead(Socket socket) {  
        this.socket = socket;  
    }  
  
    public void run() {  
        try {  
            Writer output = new OutputStreamWriter(socket.getOutputStream());  
            DataInputStream input = new DataInputStream(  
                socket.getInputStream());  
            output.write("WHO_ARE_YOU\n");  
            output.flush();  
            System.out.println("Client connected! id:" + input.readInt());  
            socket.close();  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```



# 客户端程序

```
import java.io.*;
import java.net.*;
import java.util.concurrent.atomic.AtomicInteger;

public class Client {
    int id = 0;

    Socket socket;

    public void connet(InetAddress address) throws IOException {
        socket = new Socket(address, Server.PORT_NUM);
    }

    public void comm() throws IOException {
        BufferedReader input = new BufferedReader(new InputStreamReader(
            socket.getInputStream()));
        DataOutputStream output = new DataOutputStream(socket.getOutputStream());
        String cmd = input.readLine();
        if (cmd.equals("WHO_ARE_YOU")) {
            output.writeInt(id);
        }
        socket.close();
    }
}
```





# 客户端程序

```
static AtomicInteger idGen = new AtomicInteger(1);

public Client() {
    this.id = idGen.getAndIncrement();
    System.out.println("Client Started: " + id);
}

static public void main(String[] args) throws IOException {
    for (int i = 1; i < 5; ++i) {
        new Thread() {
            public void run() {
                Client client = new Client();
                try {
                    client.connet(InetAddress.getLoopbackAddress());
                    client.comm();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }.start();
    }
}
```





### 客户端输出

```
Client Started: 1  
Client Started: 2  
Client Started: 3  
Client Started: 4
```

### 服务器端输出

```
Client connected! id:4  
Client connected! id:3  
Client connected! id:1  
Client connected! id:2
```



# 基于UDP的数据报通信

- ❖ 数据报Datagram是一种**非连接方式**，通讯数据经过不确定的路径传向目的地，可靠性和正确性都不能保证，可能会重复到达目的地，甚至还可能根本到不了目的地。
- ❖ 在java.net包中定义了DatagramSocket和DatagramPacket两个类用来支持数据报通信。
- ❖ **DatagramPacket类**表示数据报包，发送方可以用DatagramPacket构造一个数据报，其中包含拟发送的数据和目的地址及端口；接收方可以用DatagramPacket构造一个数据报用于接收发送方发来的数据报。



# 基于UDP的数据报通信

- **DatagramSocket**类表示用来发送和接收数据报包的套接字，代表数据报传送的发送和接收点，主要用来读/写称为报文的数据报中的数据。
- 发送数据报用该类的**send()**方法，接收数据报用该类的**receive()**方法。
- 创建DatagramSocket类对象实例时如果构造方法不能将DatagramSocket与指定的端口绑定在一起，将抛出SocketException异常，所以程序代码中要有相应的处理措施。



## (4) DatagramSocket类

- ❖ DatagramSocket类表示用来发送和接收数据报包的套接字。
- ❖ 继承关系
  - `java.lang.Object` → `java.net.DatagramSocket`
- ❖ 构造方法
  - `DatagramSocket()`: 常用于客户端编程, 没有特定监听的端口, 系统会随机分配一个可用的端口。
  - `DatagramSocket(int port)`: 使用固定端口进行通讯。
  - `DatagramSocket(int port, InetAddress localAddr)`
- ❖ 常用成员方法
  - `receive(DatagramPacket p)`: 接收数据包。此方法会阻塞调用线程, 线程进入阻塞态。
  - `send(DatagramPacket p)`: 发送数据包p到目的地。



## (5) DatagramPacket类

- ❖ DatagramPacket类表示数据报包。
- ❖ 继承关系
  - java.lang.Object → java.net.DatagramPacket
- ❖ 构造方法
  - DatagramPacket(byte[] buf, int length)
  - DatagramPacket(byte[] buf, int length, InetAddress address, int port)
  - DatagramPacket(byte[] buf, int offset, int length)
  - DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)
  - DatagramPacket(byte[] buf, int offset, int length, SocketAddress address)
  - DatagramPacket(byte[] buf, int length, SocketAddress address)

```

1 import java.net.*;
2 import java.io.*;
3 public class UDPSever {
4     public static void main(String[] args) throws IOException{
5         // TODO Auto-generated method stub
6         new UDPSeverThread().start();
7     }
8 }
9 class UDPSeverThread extends Thread{
10     protected DatagramSocket socket = null;
11     protected BufferedReader in =null;
12     protected boolean moreQuotes = true;
13
14     public UDPSeverThread() throws IOException{
15         super();
16         socket = new DatagramSocket(4445);
17         in = new BufferedReader(new StringReader("one\r two\r three\r four\r five\r six\r"));
18     }
19     public void run(){
20         while(moreQuotes){
21             socket.close();
22         }
23     }
24     protected String getNextQuote(){
25         String returnValue = null;
26         try{
27             if((returnValue = in.readLine()) == null){
28                 in.close();
29                 moreQuotes = false;
30                 returnValue = "No more quotes. Goodbye";
31             }
32         }
33         catch(IOException e){
34             returnValue = "IOException";
35         }
36         return returnValue;
37     }
38 }
39 }

```



# UDPClient

```
import java.io.*;
import java.net.*;
public class UDPClient {
    public static void main(String[] args) throws IOException {
        if (args.length != 1) {
            System.out.println("Usage: java UDPClient <hostname>");
            return;
        }
        DatagramSocket socket = new DatagramSocket();
        byte[] buf = new byte[256];
        InetAddress address = InetAddress.getByName(args[0]);
        DatagramPacket packet = new DatagramPacket(buf, buf.length,
address,4445);
        socket.send(packet);
        packet = new DatagramPacket(buf, buf.length);
        socket.receive(packet);
        String received = new String(packet.getData(), 0, packet.getLength(), "utf-
8");
        System.out.println("Server Say: " + received);
        socket.close();
    }
}
```





服务端：

```
public class UDPServer {
    public static void main(String[] args) throws IOException {
        // 1.创建服务,指定服务端口
        DatagramSocket datagramSocket = new DatagramSocket(8086);
        // 2.准备容器
        byte[] container = new byte[1024];
        // 3.打包容器
        DatagramPacket packet = new DatagramPacket(container, container.length);
        // 4.接收数据
        datagramSocket.receive(packet);
        // 5.从包里取出数据
        byte[] data = packet.getData();
        // 6.转换
        String str = convert(data);
        System.out.println(str);
    }

    public static String convert(byte[] data) throws IOException {
        ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(data);
        DataInputStream dis = new DataInputStream(byteArrayInputStream);
        String s = dis.readUTF();
        return s;
    }
}
```



客户端：

```
public class UDPCClient {
    public static void main(String[] args) throws IOException {
        // 1.创建服务，指定服务端口
        DatagramSocket client = new DatagramSocket(8088);
        // 2.准备数据
        String str = "哈哈！";
        byte[] data = convert(str);
        // 3.数据打包
        DatagramPacket packet = new DatagramPacket(data, data.length, new InetSocketAddress("localhost", 8086));
        // 4.发送数据
        client.send(packet);
        // 5.释放资源
        client.close();
    }

    public static byte[] convert(String str) throws IOException {
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        DataOutputStream dos = new DataOutputStream(bos);
        dos.writeUTF(str);
        dos.flush();
        byte[] data = bos.toByteArray();
        dos.close();
        return data;
    }
}
```



# 小结

## ❖ 网络编程

- InetAddress类
- ServerSocket类和Socket类, TCP: 点对点通信、点对面通信
- DatagramSocket类和DatagramPacket类, UDP: 数据报通信



❖ 在兴趣驱使下学习相关技术的同时，掌握好基础知识，学牢；以系统化的思想将所学知识融会贯通。