

## [s2]网络编程

- [概述](#)
  - [OSI 7层模型](#)
- [网络通信的要素（传输层）](#)
  - [网络编程中2个主要问题](#)
  - [网络编程中的要素](#)
- [IP](#)
  - [作用： 唯一定位一台网络上的计算机](#)
  - [本机（localhost）的IP：127.0.0.1](#)
  - [IP地址分类](#)
    - [IPV4/IPV6](#)
      - [IPV4](#)
      - [IPV6](#)
        - [128位是怎么算出来的？](#)
    - [公网（互联网）/ 私网（局域网）](#)
      - [ABCD类地址](#)
- [port端口](#)
  - [一些概念](#)
  - [端口分类](#)
    - [公有端口](#)
    - [程序注册端口](#)
    - [动态/私有端口](#)
- [实现基于TCP的消息收发](#)
  - [设计思路，实现，测试](#)
    - [客户端实现](#)
    - [服务端实现](#)
    - [测试效果](#)
- [实现基于TCP的文件收发](#)
  - [设计思路及实现](#)
    - [客户端实现](#)

- [服务端实现](#)
  - [思考](#)
- [再谈io](#)
- [初识tomcat](#)
- [UDP](#)
  - [测试](#)
    - [测试数据收发](#)
      - [思路](#)
      - [客户端代码](#)
      - [服务端代码](#)
    - [测试实现聊天](#)
      - [思路](#)
      - [发送端代码](#)
      - [接收端代码](#)
      - [测试效果](#)
    - [测试实现在线互动](#)
      - [思路](#)
      - [代码实现](#)
        - [发送端代码](#)
        - [接收端代码](#)
        - [测试代码](#)
      - [测试效果](#)
- [URL](#)
  - [测试使用URL下载资源](#)
    - [代码](#)
    - [测试](#)

开始时间 07.04.2020

结束时间 07.07

# 概述

---

# OSI 7层模型

OSI（Open System Interconnect），即开放式系统互联

OSI中的层	功能	TCP/IP协议族
应用层	文件传输，电子邮件，文件服务，虚拟终端	TFTP，HTTP，SNMP，FTP，SMTP，DNS，RIP，Telnet
表示层	数据格式化，代码转换，数据加密	没有协议
会话层	解除或建立与别的接点的联系	没有协议
传输层	提供端对端的接口	TCP，UDP
网络层	为数据包选择路由	IP，ICMP，OSPF，BGP，IGMP，ARP，RARP
数据链路层	传输有地址的帧以及错误检测功能	SLIP，CSLIP，PPP，MTU，ARP，RARP
物理层	以二进制数据形式在物理媒体上传输数据	ISO2110，IEEE802，IEEE802.2

## 网络通信的要素（传输层）

### 网络编程中2个主要问题

- 1. 如何准确定位到网络上的1台或多台主机？
- 2. 找到主机之后如何进行通信？

### 网络编程中的要素

- 1. IP和端口号
- 2. 网络通信协议（TCP、UDP）

## IP

**作用： 唯一定位一台网络上的计算机**

**本机（localhost）的IP：127.0.0.1**

## **IP地址分类**

### **IPV4/IPV6**

#### **IPV4**

如127.0.0.1 由4个字节组成，每个字节长度为0-255

共42亿个，其中配额 北美30亿，亚洲4亿，已于2011年用尽

#### **IPV6**

128位

## 表示方法

IPv6的地址长度为128位，是IPv4地址长度的4倍。于是IPv4点分十进制格式不再适用，采用十六进制表示。IPv6有3种表示方法。

### 一、冒分十六进制表示法

格式为X:X:X:X:X:X:X，其中每个X表示地址中的16b，以十六进制表示，例如：

ABCD:EF01:2345:6789:ABCD:EF01:2345:6789

这种表示法中，每个X的前导0是可以省略的，例如：

2001:0DB8:0000:0023:0008:0800:200C:417A → 2001:DB8:0:23:8:800:200C:417A

### 二、0位压缩表示法

在某些情况下，一个IPv6地址中间可能包含很长的一段0，可以把连续的一段0压缩为“::”。但为保证地址解析的唯一性，地址中“::”只能出现一次，例如：

FF01:0:0:0:0:0:0:1101 → FF01::1101

0:0:0:0:0:0:0:1 → ::1

0:0:0:0:0:0:0:0 → ::

### 三、内嵌IPv4地址表示法

为了实现IPv4-IPv6互通，IPv4地址会嵌入IPv6地址中，此时地址常表示为：X:X:X:X:X:d.d.d.d，前96b采用冒分十六进制表示，而最后32b地址则使用IPv4的点分十进制表示，例如::192.168.0.1与::FFFF:192.168.0.1就是两个典型的例子，注意在前96b中，压缩0位的方法依旧适用 [\[9\]](#)。



## 128位是怎么算出来的？

例如这样一个ipv6地址

2001:0db8:85a3:0000:1319:8a2e:0370:7344

一共八组，每组4个数字，一共32个数，每个数字都是十六进制数，一个十六进制数可以写成4个二进制数（就是十六进制数转成二进制数）

可以理解为有多少位就是多少个二进制数

所以， $32 \times 4 = 128$ 位

可以这么理解：

8组32个数字，每个数字上可以有16种可能，32个数字就是 $16^{32}$ 种可能。用二进制的话，每个16进制数需要用4位二进制来表示，所以是 $(2^4)^{32}$ 位

## 公网（互联网） / 私网（局域网）

### ABCD类地址

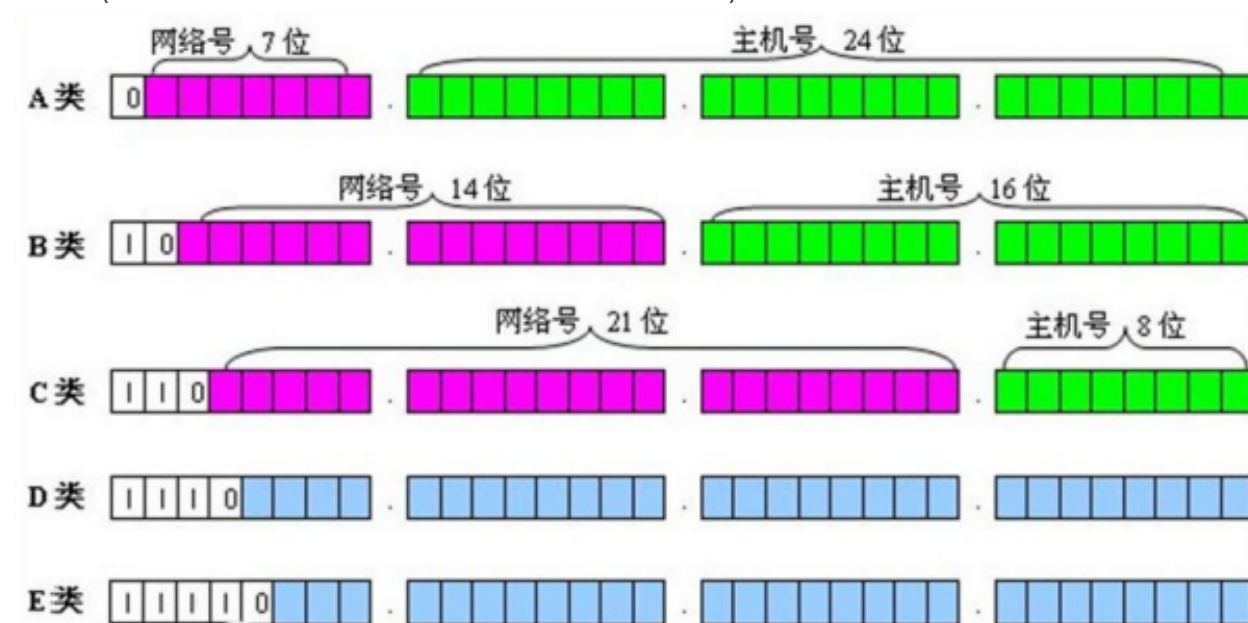
为供不同规模的网络使用，Internet委员会共定义了5种IP地址类型，即A类~E类。

所以IP地址中的ABCD等类指的是不同规模的网络。

区分方法：

每类IP地址都是有标识符的，同时也是有范围的。

标识符(将点分十进制IP转换成二进制值后对照该图即可)：



IP地址范围：

A类IP地址 地址范围1.0.0.1-126.255.255.254（二进制表示为：00000001 00000000 00000000 00000001 - 01111110 11111111 11111111 11111110）。

B类IP地址地址范围128.1.0.1-191.254.255.254（二进制表示为：10000000 00000001 00000000 00000001 - 10111111 11111110 11111111 11111110）

C类IP地址范围192.0.1.1-223.255.254.254（二进制表示为: 11000000 00000000 00000001 00000001 - 11011111 11111111 11111110 11111110）。

D类IP地址范围224.0.0.1-239.255.255.254。

E类IP地址范围240.0.0.0---255.255.255.254。

上述IP地址均为IPv4地址。

# port端口

---

## 一些概念

1. 可理解为表示计算机上的一个程序的进程
2. 被规定在 0-65535
3. 端口号在单个协议下不能冲突，但在不同协议下可以，比如TCP,UDP可以同时使用80端口。

## 端口分类

### 公有端口

范围： 0-1023

举例：

- http : 80
- https : 443
- ftp : 21
- telnet : 23

### 程序注册端口

范围： 1024 - 49151

举例：

- tomcat : 8080
- mysql : 3306

- oracle : 1521

## 动态/私有端口

范围：49152 - 65535

# 实现基于TCP的消息收发

---

## 设计思路，实现，测试

1. 接受消息需要服务端，发送消息需要客户端
2. 客户端要找到服务端，所以服务端必须拥有一个可访问的地址 (ip+port)

## 客户端实现

```
public class DemoTcpClient {  
    public static void main(String[] args) throws IOException {  
        Socket socket = null;  
        OutputStream os = null;  
        try {  
            //1. 使用服务端信息  
            InetAddress serverIp = InetAddress.getLocalHost();  
            int port = 9999;  
            //2. 使用该信息创建一个新的连接  
            socket = new Socket(serverIp, port);  
            //3. 发送消息  
            os = socket.getOutputStream();  
            os.write("hello from sean".getBytes());  
        }  
    }  
}
```



```

        System.out.println("msg sent");
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        os.close();
        socket.close();
    }
}
}

```

## 服务端实现

```

public class DemoTcpServer {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = null;
        Socket socket = null;
        InputStream is = null;
        ByteArrayOutputStream baos = null;
        try {
            //注册服务器地址
            serverSocket = new ServerSocket(9999);
            //等待客户端连接
            socket = serverSocket.accept();//这是一个阻塞式的方法，接受一次后关闭
            //读取客户端消息
            is = socket.getInputStream();
            baos = new ByteArrayOutputStream();
            byte[] buffer = new byte[1024];
            int len;
            while ((len = is.read(buffer)) != -1) {
                baos.write(buffer, 0, len);
            }
            System.out.println("msg received : " + baos.toString());
        }
    }
}

```

```
    } catch (IOException e) {  
        e.printStackTrace();  
    } finally {  
        baos.close();  
        ;  
        is.close();  
        socket.close();  
    }  
}  
}
```

## 测试效果

客户端输出：

```
msg sent
```

服务端输出：

```
msg received : hello from sean
```

# 实现基于TCP的文件收发

---

## 设计思路及实现

增设一个需求，

客户端发送完文件后，通知服务端。

服务端接受完文件后，通知客户端。

和前面发消息比起来，区别就是使用的io流类型不一样。

客户端：发送文件之前，需要先将文件读取到文件输入流，再使用输出流输出。输出

服务端：接受文件时，需要使用文件输出流将输入内容写出。

## 客户端实现

```
public class DemoTcpFileClient {
    public static void main(String[] args) throws IOException {
        //1. 使用服务端信息
        InetAddress serverIp = InetAddress.getLocalHost();
        int port = 9998;
        //2. 使用该信息创建一个新的连接
        Socket socket = new Socket(serverIp, port);
        OutputStream os = socket.getOutputStream();
        //3. 读取文件
        URL url = DemoTcpFileClient.class.getClassLoader().getResource("arrow.jpg");
        FileInputStream fis = new FileInputStream(new File(url.getPath()));
        //4. 写出文件
        byte[] buffer = new byte[1024];
        int len;
        while ((len = fis.read(buffer)) != -1) {
            os.write(buffer, 0, len);
        }
        //5. 关闭输出流，这句话必须写
        socket.shutdownOutput();
        System.out.println("file sent");

        //6. 等待服务端的确认收到的应答
        //6.1 从会话中获取输入流
        InputStream serverMsgReader = socket.getInputStream();
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        byte[] buffer2 = new byte[1024];
        int len2;
        while ((len2 = serverMsgReader.read(buffer2)) != -1) {
            baos.write(buffer2, 0, len2);
        }
    }
}
```

```

    }
    System.out.println("msg from server : " + baos.toString());

    //7.按就近使用顺序关闭流
    baos.close();
    serverMsgReader.close();
    fis.close();
    os.close();
    socket.close();
}
}

```

## 服务端实现

```

public class DemoTcpFileServer {
    public static void main(String[] args) throws IOException {
        //注册服务器地址
        ServerSocket serverSocket = new ServerSocket(9998);
        //等待客户端连接
        Socket socket = serverSocket.accept();//这是一个阻塞式的方法，接受一次后关闭
        System.out.println("connection established");
        //读取客户端消息
        InputStream is = socket.getInputStream();
        FileOutputStream fos = new FileOutputStream(new File("copy.jpg"));
        byte[] buffer = new byte[1024];
        int len;
        while ((len = is.read(buffer)) != -1) {
            fos.write(buffer, 0, len);
        }
        //告知客户端，文件已收到
        //获取会话的输出流
        OutputStream os = socket.getOutputStream();
    }
}

```

```
os.write("file recived".getBytes());  
//关闭流  
os.close();  
fos.close();  
is.close();  
socket.close();  
}  
}
```

## 思考

问：为什么客户端的shutdownOutput必须写？

答：

可以从时序图来分析

- 1.客户端与服务端的会话（socket）建立
- 2.客户端使用socket的输出流输出文件
- 3.服务端使用socket的输入流接受文件
- 4.客户端告知服务端发送完毕
- 5.服务端告知客户端接收完毕
- 6.客户端确认

所以shutdownOutput的作用是上面的第4步，如果客户端的输出流一直不关闭，服务端就会一直等待接受。

## 再谈io

io的作用就像适配器一样，确保在收发过程中确保正常“沟通”。

最上层的inputstream/outputstream虽然可以确保收发无误，但是要“认识”收发的东西，就要使用具体的管道流。

比如，客户端需要发送一个文件，就要先将文件读入到文件输入流里，然后使用输出流，逐个

将文件输入流的内容输出。

服务端需要接受一个文件并产生一个文件副本。

首先要从socket里获取输入流，读取接受到的信息。

然后使用文件输出流，将接受到的信息逐个”翻译“成图片，并输出到指定位置。

# 初识tomcat

---

相比前面的TCP demo

服务端：

DIY

tomcat

客户端：

DIY

浏览器

# UDP

---

用一个比喻来初识：TCP类比打电话，UDP类比写信（只需要地址，然后投递信件(java中的DatagramPacket)即可）。

# 测试

## 测试数据收发

思路

UDP不存在客户端和服务端的界限，这个例子只是测试收发。

发送方只要有地址，就能发送数据包。

服务端若要接受数据包，仍然需要监听。

## 客户端代码

```
public class DemoUdpClient {
    public static void main(String[] args) throws IOException {
        DatagramSocket socket = new DatagramSocket();

        byte[] content = "hello from sean".getBytes();
        DatagramPacket packet
            = new DatagramPacket(content, 0, content.length, InetAddress.getLocalHost(),
19090);

        socket.send(packet);
    }
}
```

## 服务端代码

```
public class DemoUdpServer {
    public static void main(String[] args) throws IOException {
        DatagramSocket socket = new DatagramSocket(9090);

        byte[] buffer = new byte[1024];
        DatagramPacket packet = new DatagramPacket(buffer, 0, buffer.length);
        socket.receive(packet);

        System.out.println(packet.getAddress());
        System.out.println(new String(packet.getData(),0,packet.getLength()));
    }
}
```

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

## 测试实现聊天

### 思路

发送端和接收端都拥有彼此的地址即可。

### 发送端代码

```
public class DemoSender {
    public static void main(String[] args) throws IOException {

        final int ANOTHER_CHATTER_PORT = 6666;

        DatagramSocket socket = new DatagramSocket(7777);
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        while(true) {
            String content = reader.readLine();
            if(content.equals("bye")) {
                break;
            }
            byte[] contentByte = content.getBytes();
            DatagramPacket packet = new DatagramPacket(contentByte, 0, contentByte.length,
                InetAddress.getLocalHost(), ANOTHER_CHATTER_PORT);
            socket.send(packet);
        }
        reader.close();
        socket.close();
    }
}
```

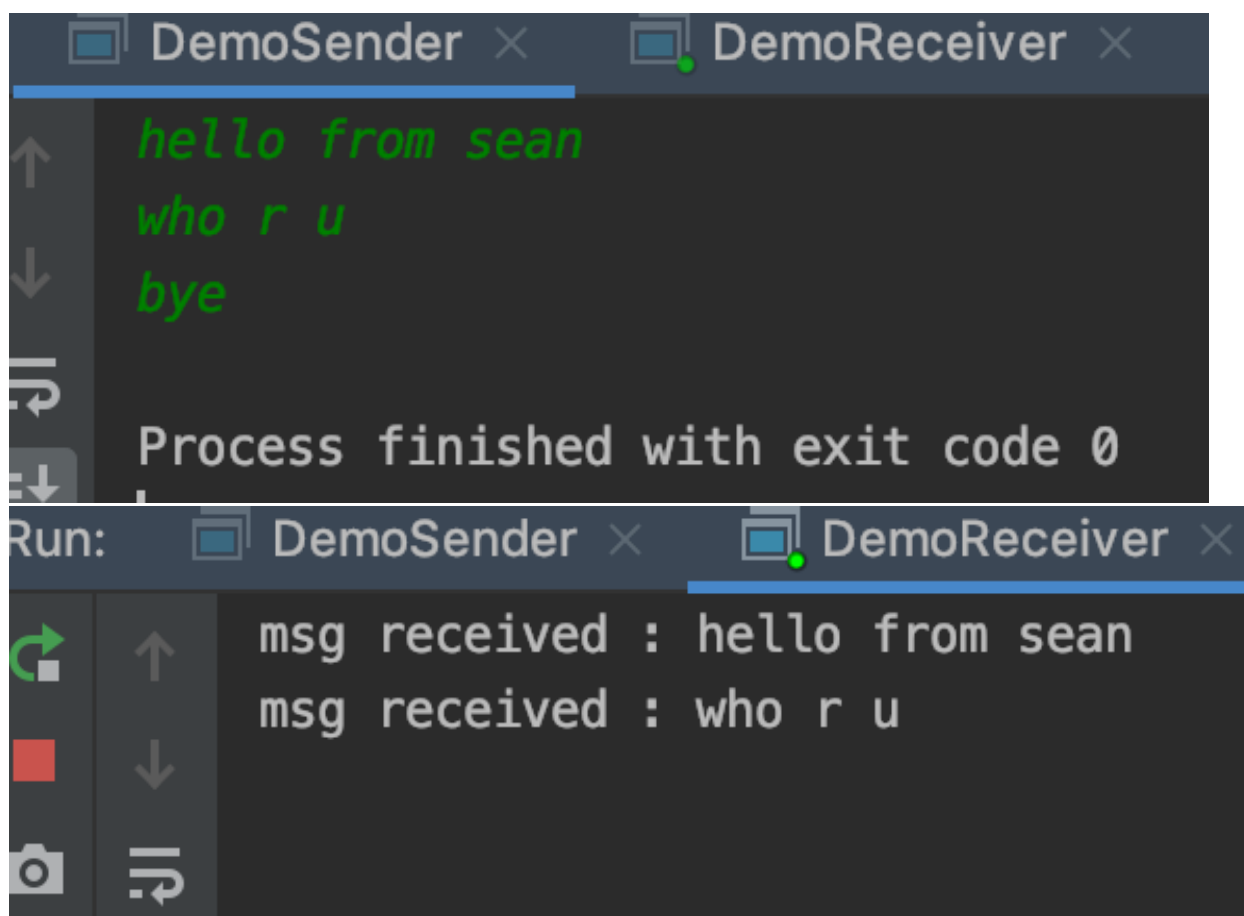


```
}  
}
```

## 接收端代码

```
public class DemoReceiver {  
    public static void main(String[] args) throws IOException {  
        final int ANOTHER_CHATTEr_PORT = 7777;  
        DatagramSocket socket = new DatagramSocket(6666);  
  
        while(true) {  
            byte[] buffer = new byte[1024];  
            DatagramPacket packet = new DatagramPacket(buffer,0,buffer.length);  
            socket.receive(packet);  
  
            String receivedMsg = new String(packet.getData(),0,packet.getLength());  
            System.out.println("msg received : " + receivedMsg );  
  
            if(receivedMsg.equals("bye")){  
                break;  
            }  
        }  
        socket.close();  
    }  
}
```

## 测试效果



The image shows two terminal windows. The top window, titled 'DemoSender', contains the text 'hello from sean', 'who r u', and 'bye' in green, followed by 'Process finished with exit code 0' in white. The bottom window, titled 'Run: DemoSender DemoReceiver', shows 'msg received : hello from sean' and 'msg received : who r u' in white text.

```
hello from sean
who r u
bye

Process finished with exit code 0

Run: msg received : hello from sean
      msg received : who r u
```

## 测试实现在线互动

### 思路

假设A和B互动，即意味着A,B各自都需要实现收发消息。

即A需要有B的地址，B也需要有A的地址，但是收发的端口号不能一样，所以A和B各自有2个端口，共4个端口。

即A发送使用端口，A接收使用端口，B发送使用端口，B接收使用端口。

如果仅仅是让A,B都能接到各自的消息，只需让A,B知道对方的接收使用端口即可。

但是，如果想通过端口号就知道对方身份，则需要将发送方的发送端口号告知对方。

举个具体的例子：

A发送使用1端口，接收使用3端口，  
B发送使用2端口，接收使用4端口

A只需要将消息发送到4端口即可

B收到了来自1端口的消息，但是B并不知道1端口是谁在用，所以需要提前告知B,1端口是A在用。

## 代码实现

### 发送端代码

```
public class DemoConcurrentSender implements Runnable {
    DatagramSocket socket;
    BufferedReader reader;
    InetAddress toIp;
    int toPort;

    public DemoConcurrentSender(int fromPort, InetAddress toIp, int toPort)
        throws SocketException {
        this.socket = new DatagramSocket(fromPort);
        this.reader = new BufferedReader(new InputStreamReader(System.in));
        this.toIp = toIp;
        this.toPort = toPort;
    }

    public void run() {
        try {
            while (true) {
                String content = null;
                content = reader.readLine();
                byte[] contentByte = content.getBytes();
                DatagramPacket packet = new DatagramPacket(contentByte, 0,
                    contentByte.length, this.toIp, this.toPort);
                socket.send(packet);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```

        if (content.equals("bye")) {
            break;
        }
    }
    reader.close();
    socket.close();
} catch (IOException e) {
    e.printStackTrace();
}
}
}

```

接收端代码

```

public class DemoConcurrentReceiver implements Runnable {

    DatagramSocket socket;

    public DemoConcurrentReceiver(int fromPort) throws SocketException {
        this.socket = new DatagramSocket(fromPort);
    }

    public void run() {
        while(true) {
            try {
                byte[] buffer = new byte[1024];
                DatagramPacket packet = new DatagramPacket(buffer,0,buffer.length);
                socket.receive(packet);
                String receivedMsg = new String(packet.getData(),0,packet.getLength());
                String msgFrom;
                switch (packet.getPort()) {
                    case KOBE_SEND_PORT : msgFrom = "kobe"; break;
                    case MJ_SEND_PORT : msgFrom = "mj"; break;
                }
            }
        }
    }
}

```

```

        default: msgFrom = "unknown"; break;
    }
    System.out.println(String.format("msg received from %s : %s",
msgFrom,receivedMsg));
    if(receivedMsg.equals("bye")){
        break;
    }
} catch (IOException e) {
    e.printStackTrace();
}
}
socket.close();
}
}

```

## 测试代码

定义一个常量类来存放端口号

```

public class ConstantUtil {
    public static final int KOBE_SEND_PORT = 24241;
    public static final int KOBE_RECEIVED_PORT = 24242;

    public static final int MJ_SEND_PORT = 23231;
    public static final int MJ_RECEIVED_PORT = 23232;
}

```

定义2个人，分别启用各自的端口号，进行测试通讯

```

public class Mj {
    public static void main(String[] args) throws UnknownHostException, SocketException {
        new Thread(new

```

```

DemoConcurrentSender(MJ_SEND_PORT,InetAddress.getByName("localhost"),KOBE_RECEIVED_PORT)).start();
    new Thread(new DemoConcurrentReceiver(ConstantUtil.MJ_RECEIVED_PORT)).start();
}
}

```

```

public class Kobe {
    public static void main(String[] args) throws UnknownHostException, SocketException {
        new Thread(new
DemoConcurrentSender(KOBE_SEND_PORT,InetAddress.getByName("localhost"),MJ_RECEIVED_PORT)).start();
        new Thread(new
DemoConcurrentReceiver(ConstantUtil.KOBE_RECEIVED_PORT)).start();
    }
}

```

## 测试效果

```

Mj x Kobe x
↑ hi there
↓ msg received from mj : halo kobe

un: Mj x Kobe x
↑ /Library/Java/JavaVirtualMachines/jdk1.8.0_101
↓ objc[3058]: Class JavaLaunchHelper is implemented by two different libraries
msg received from kobe : hi there
halo kobe

```

# URL

---

统一资源定位符：

e.g. <https://www.baidu.com>

dns解析：ip <-> 域名

协议 + ip + 端口号 + 项目 + 资源

## 测试使用URL下载资源

### 代码

```
public class URLdownload {
    public static void main(String[] args) throws IOException {
        String downloadSource =
            "https://ws.stream.qqmusic.qq.com/C400003xuYjZ1llbtO.m4a?
            guid=2210128444&vkey=8242D550CB05C87C6A6A5431766D814E6C67C1A4B743F18F0
            C4A7DC7096853CD4465ED064079AE757968D300B523BF89DDE0A7CD3B5E0DE6&uin=
            0&fromtag=66";
        URL url = new URL(downloadSource);
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();

        InputStream is = connection.getInputStream();

        String outputSource = "output.m4a";
        FileOutputStream fos = new FileOutputStream(outputSource);

        byte[] buffer = new byte[1024];
        int len;
        while((len = is.read(buffer)) != -1) {
```

```
fos.write(buffer,0 ,len);  
}  
}  
}
```

## 测试

随便找一个网页播放器，然后播放一首歌，找到带资源的链接。

