网络编程

回顾：

今天任务

```
1.网络编程基础
2.IP地址、端口、通信协议
3.相关类
4.TCP编程模型
5.了解UDP编程
```

教学目标

```
1.了解网络概念
2.理解IP地址、端口、通信协议
3.掌握InetAddress、ServerSocket、Socket
4.掌握TCP编程
```

第一节：网络编程基础

1.1 计算机网络

> 所谓计算机网络，就是把分布在不同区域的计算机与专门的外部设备用通信线路相互连接成一个规模大，而且功能强的网络系统，从而使得计算机之间可以相互传递信息，共享数据、软件等资源

1.2 网络编程

> 所谓网络编程，指的就是在同一个网络中不同机器之间的通信

第二节：计算机之间通信需要的条件

2.1 IP地址

> IP地址指的是互联网地址(Internet Protocol Address),是联网设备与互联网之间的唯一标识，在同一个网段中，IP地址是唯一的
>
> IP地址是数字型的，是一个32位的整数，通常将其分成4个8位的二进制数，每8位之间用圆点隔开， 每个8位整数可以转换为一个0~255的十进制整数，例如:202.9.128.88
>
> 分为IPV4和IPV6
>
> IP地址分类
>
> A类：保留给政府结构，1.0.0.1 ~ 126.255.255.254

> B类：分配给中型企业，1.0.0.1 ~ 126.255.255.254
>
> C类：分配给任何需要的个人，192.0.0.1 ~ 223.255.255.254
>
> D类：用于组播，224.0.0.1 ~ 239.255.255.254
>
> E类：用于实验，240.0.0.1 ~ 255.255.255.254
>
> 回收地址：127.0.0.1，指本地机，一般用于测试使用
>
> IP地址可以唯一的确定网络上的一个通信实体，但一个通信实体可以有多个通信程序同时提供网络服务，此时还需要使用端口

2.2 端口

> 数据的发送和接收都需要通过端口出入计算机，端口号用于唯一标识通信实体上进行网络通讯的程序，同一台机器上不能两个程序占用同一个端口
>
> 端口号的取值范围：0~65535
>
> 端口分类：
>
> 公认端口：0~1023
>
> 注册端口：1025~49151
>
> 动态或私有端口：1024~65535
>
> 常用端口：
>
> mysql：3306
>
> oracle：1521
>
> tomcat：8080

2.3 通信协议

> 需要通信的设备之间需要实现相同的通信协议
>
> 网络分层：物理层，数据链路层，网络层，传输层，会话层，表示层，应用层
>
> 通信协议分类：
>
> 传输层协议：TCP和UDP
>
> 网络层IP协议：IPV4和IPV6，互联网协议
>
> 应用层协议：HTTP

第三节：相关类的使用

3.1 InetAddress类

> Java提供了InetAddress类来代表ip地址，是对ip地址的抽取和封装，有两个子类：Inet4Address,Inet6Address，分别表示IPv4和IPv6

常用方法：

```java
//1.获取主机：主机名称和ip地址
/**
 * static InetAddress getLocalHost()
   返回本地主机。
 */
InetAddress id1 = null;
try {
    id1 = InetAddress.getLocalHost();
    //USER-VG9EDR1SST/10.31.165.42
    System.out.println(id1);
} catch (UnknownHostException e) {
    // 未知的主机
    e.printStackTrace();
}

//2.获取ip地址的字符串表示形式
/**
 *  String getHostAddress()
 返回 IP 地址字符串（以文本表现形式）。
 */
String str1 = id1.getHostAddress();
System.out.println(str1);//10.31.165.42

//3.获取主机名
/**
 *  String getHostName()
 获取此 IP 地址的主机名。
 */
String str2 = id1.getHostName();
System.out.println(str2);

//4.根据主机或者ip地址获取InetAddress对象
/**
 * static InetAddress getByName(String host)
 在给定主机名的情况下确定主机的 IP 地址。
 */
try {
    InetAddress id2 = InetAddress.getByName("10.31.165.42");
    ///10.31.165.42
    System.out.println(id2);

    InetAddress id3 = InetAddress.getByName("www.baidu.com");
    //www.baidu.com/115.239.211.112
    System.out.println(id3);
} catch (UnknownHostException e) {
    e.printStackTrace();
}

//5.根据主机或者ip地址获取所有InetAddress对象

/**
```

```
         * static InetAddress[] getAllByName(String host)
         */
        try {
            InetAddress[] arr = InetAddress.getAllByName("www.baidu.com");
            for(InetAddress address:arr) {
                //www.baidu.com/115.239.210.27
                System.out.println(address.toString());
                //115.239.210.27
                System.out.println(address.getHostAddress());
                //www.baidu.com
                System.out.println(address.getHostName());
            }
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }
```

3.2 URLEncoder类和URLDecoder类

> URLEncoder类和URLDecoder类用于完成普通字符串和application/x-www-form-urlencoded MIME字符串之间的转换

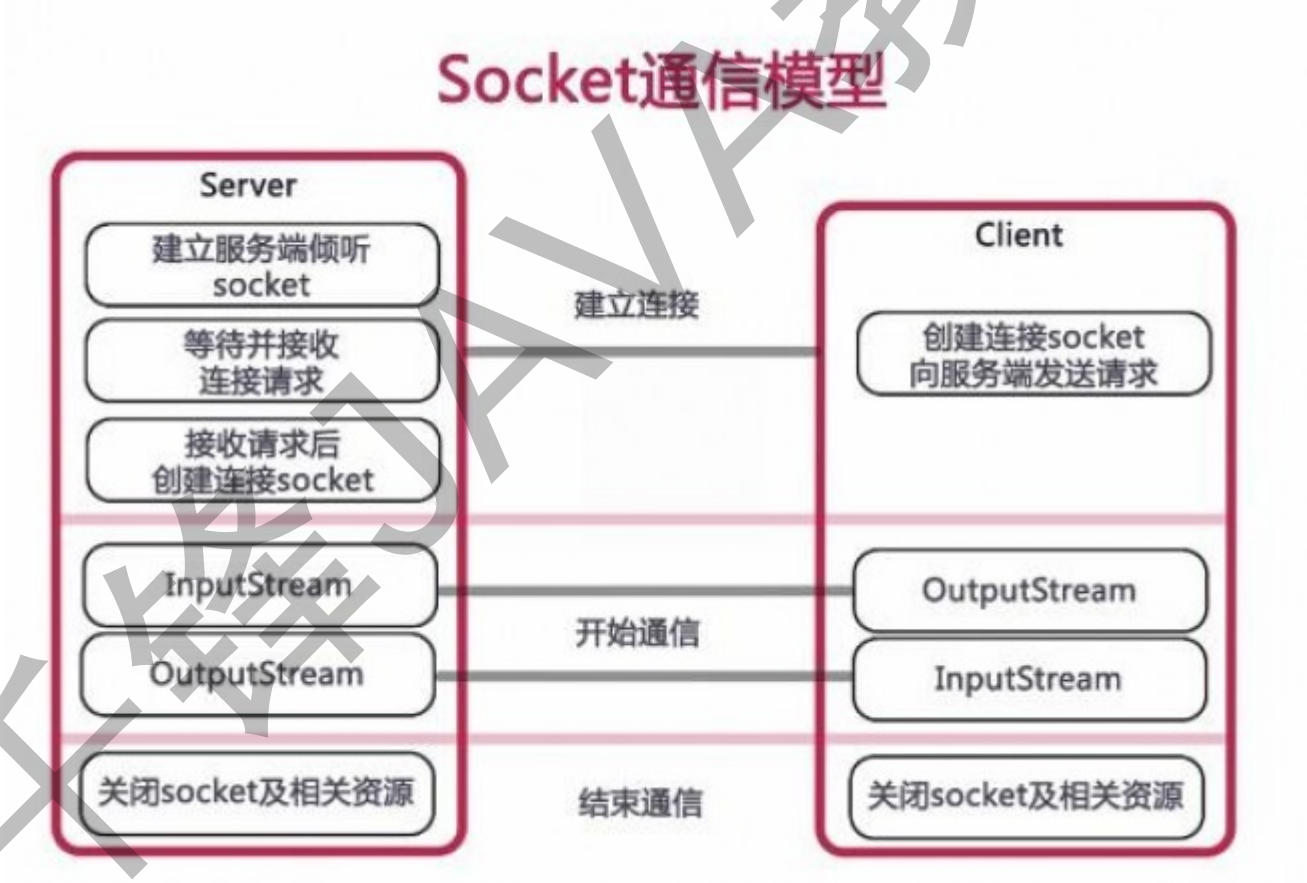
```
        //%E5%8D%83%E9%94%8B
        //URLDecoder: 解码,将特殊中文字符串转换为普通字符串
        String string1 = URLDecoder.decode("%E5%8D%83%E9%94%8B", "utf-8");
        System.out.println(string1);

        //URLEncoder:编码，将普通字符串转换为特殊字符串
        String string2 = URLEncoder.encode("Java的开发指南", "GBK");
        System.out.println(string2);

        //注意：编码和解码需要采用相同的字符集
        String string3 = URLDecoder.decode(string2, "utf-8");
        System.out.println(string3);
```

第四节：基于TCP的网络编程

4.1 概念

> TCP，Transmission Control Protocol，传输控制协议，基于字节流的传输层通信协议
>
> 特点：
>
> a.安全的
>
> b.面向链接的
>
> c.面向链接的
>
> d.传输数据大小限制，一旦连接建立，双方可以按统一的格式传输大的数据
>
> TCP的三次握手
>
> a.客户端向服务端发送一个请求
>
> b.服务端收到请求后，回客户端一个响应
>
> c.客户端向收到服务端的响应后，回服务端一个确认信息

4.2 Socket通信模型



第五节：Socket和ServerSocket

5.1 客户端发送消息，服务端接收消息

客户端

```java
public class Client {
    public static void main(String[] args) {
        //1.建立一个与服务端之间的连接
        /**
         * 面向连接：三次握手
         * 创建一个Socket的对象，就相当于完成了三次握手，建立了一个安全的连接
         */
        /**
         * Socket(InetAddress address, int port)
         创建一个流套接字并将其连接到指定 IP 地址的指定端口号。
         */
        //注意：包含了两个流：InputStream和OutputStream
        Socket socket = null;
        try {
            socket = new Socket(InetAddress.getByName("10.31.165.42"),7777);

            //2.将需要发送的数据写入到网络中
            /**
             *  InputStream getInputStream()
          返回此套接字的输入流 。
           OutputStream getOutputStream()
          返回此套接字的输出流。
             */
            //InputStream input = new BufferedInputStream(socket.getInputStream());
            //InputStream input = socket.getInputStream();
            OutputStream output = socket.getOutputStream();

            //3.写入
            output.write("hello你好吗?".getBytes());
            output.flush();

        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                socket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

服务端

```java
public class Server {
    public static void main(String[] args) {
        //1.实例化一个ServerSocket的对象
        /**
         * ServerSocket(int port)
         创建绑定到特定端口的服务器套接字。
         */
        ServerSocket serverSocket = null;
        try {
            serverSocket = new ServerSocket(7777);

            //2.监听，获取连接到的客户端
            /**
             *  Socket accept()
          侦听并接受到此套接字的连接。
             */
            System.out.println("等待服务端的连接...");
            //注意：在未连接成功之前，将一直处于阻塞状态
            Socket socket = serverSocket.accept();

            System.out.println("连接成功");

            //3.获取网络到内存的一个输入流
            InputStream input = socket.getInputStream();

            //4.读取数据
            byte[] arr = new byte[1024];
            int len = input.read(arr);
            String message = new String(arr, 0, len);

            //5.组织信息
            String ipString = socket.getInetAddress().getHostAddress();
            int port = socket.getPort();

            System.out.println(ipString +":" + port + "说你说：" + message);

        } catch (IOException e) {
            e.printStackTrace();
        }finally {
            try {
                serverSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

5.2 客户端发送消息，服务端回复消息

客户端

```java
public class Client {
    public static void main(String[] args) {
        // 1.建立一个与服务端之间的连接
        /**
         * 面向连接：三次握手 创建一个Socket的对象，就相当于完成了三次握手，建立了一个安全的连接
         */
        // 注意：包含了两个流：InputStream和OutputStream
        Socket socket = null;
        try {
            socket = new Socket(InetAddress.getByName("10.31.165.42"), 7777);

            // 2.将需要发送的数据写入到网络中
            OutputStream output = socket.getOutputStream();

            // 3.写入
            output.write("hello你好吗?".getBytes());
            output.flush();

            // 4.收取服务端发送来的消息
            InputStream input = socket.getInputStream();

            byte[] arr = new byte[1024];
            int len = input.read(arr);
            String message = new String(arr, 0, len);
            System.out.println("来自服务端的回复：" + message);

        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                socket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

服务端

```java
public class Server {
    public static void main(String[] args) {
        //1.实例化一个ServerSocket的对象
        ServerSocket serverSocket = null;
        try {
            serverSocket = new ServerSocket(7777);

            //2.监听，获取连接到的客户端
            System.out.println("等待服务端的连接...");
            //注意：在未连接成功之前，将一直处于阻塞状态
            Socket socket = serverSocket.accept();

            System.out.println("连接成功");

            //3.获取网络到内存的一个输入流
            InputStream input = socket.getInputStream();

            //4.读取数据
            byte[] arr = new byte[1024];
            int len = input.read(arr);
            String message = new String(arr, 0, len);

            //5.组织信息
            String ipString = socket.getInetAddress().getHostAddress();
            int port = socket.getPort();

            System.out.println(ipString +":" + port + "说：" + message);

            //6.服务端给客户端回复消息
            OutputStream output = socket.getOutputStream();
            output.write("你也好，好久不见了".getBytes());
            output.flush();

        } catch (IOException e) {
            e.printStackTrace();
        }finally {
            try {
                serverSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

5.3 客户端上传文件到服务端【以图片为例】

客户端

```
public class Client {
    public static void main(String[] args) {
        //1.创建Socket对象
        Socket socket = null;
        try {
            socket = new Socket(InetAddress.getByName("10.31.165.42"), 8888);

            //2.输入流和输出流
            //输入流：用来读取客户端磁盘上的图片文件
            BufferedInputStream input = new BufferedInputStream(new FileInputStream(new
File("file/client/image.png")));
            //输出流：将图片写入到网络中
            BufferedOutputStream output = new BufferedOutputStream(socket.getOutputStream());

            //3.将读取到的数据写入
            byte[] arr = new byte[1024];
            int len = 0;
            while((len = input.read(arr)) != -1) {
                output.write(arr, 0, len);
                output.flush();
            }

        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            try {
                socket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

服务端

```java
public class Server {
    public static void main(String[] args) {
        //1,实例化一个ServerSocket对象
        ServerSocket serverSocket = null;
        try {
            serverSocket = new ServerSocket(8888);

            //2.等待客户端的连接
            System.out.println("等待连接...");
            Socket socket = serverSocket.accept();
            System.out.println("连接成功");

            //3.读取网络中的数据
            //输入流：读取网络中的数据
            BufferedInputStream input = new BufferedInputStream(socket.getInputStream());
            //输出流：将读取到的数据写入到服务端的磁盘
            BufferedOutputStream output = new BufferedOutputStream(new FileOutputStream(new
File("file/server/image_copy.png")));

            //4.读取和写入
            byte[] arr = new byte[1024];
            int len = 0;
            while((len = input.read(arr)) != -1) {
                output.write(arr, 0, len);
                output.flush();
            }

            System.out.println("上传文件成功");

        } catch (IOException e) {
            e.printStackTrace();
        }finally {
            try {
                serverSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

5.4 客户端从服务端下载文件【以图片为例】

客户端

```java
public class Client {
    public static void main(String[] args) {
        //1.实例化一个Socket对象
        Socket socket  = null;
        try {
            socket = new Socket(InetAddress.getByName("10.31.165.42"), 9999);

            //2.给服务端发送消息，消息的内容就是你需要下载的文件
            OutputStream output = socket.getOutputStream();
            output.write("image_copy.png".getBytes());
            output.flush();

            //3.将将读取出来的文件写入到客户端的磁盘中
            InputStream inputStream = socket.getInputStream();

            OutputStream output1 = new FileOutputStream(new File("file/client/image11.png"));

            byte[] arr1 = new byte[1024];
            int len1 = 0;
            while((len1 = inputStream.read(arr1)) != -1) {
                output1.write(arr1, 0, len1);
                output1.flush();
            }

            System.out.println("下载完成");

        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            try {
                socket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    /**
     * Connection reset:连接重置
     * 出现的原因：
     * a.如果服务端因为某种原因关闭，但是客户端仍然在访问服务端
     * b.如果多个人同时访问同一个ip地址和端口的话
     *
     * 解决办法：
     * 在ServerSocket服务端调用Socket类中shutdownOutput()，作用禁用输出流
     */

}
```

服务端

```java
public class Server {
    public static void main(String[] args) {
        //1.实例化一个ServerSokcet对象
        ServerSocket serverSocket = null;
        try {
            serverSocket = new ServerSocket(9999);

            //2.监听
            Socket socket = serverSocket.accept();

            //3.读取客户端发送来的消息
            InputStream input = socket.getInputStream();
            byte[] arr = new byte[1024];
            int len = input.read(arr);
            String string = new String(arr, 0, len);//image_copy.png

            //4.在服务端的磁盘下查找是否存在指定的文件
            File file = new File("file/server", string);

            //5.判断
            if(file.exists()) {
                //6.先从服务端的磁盘中读取出来
                InputStream inputStream = new FileInputStream(file);

                //7.将读取出来的文件写入到网络中
                OutputStream output = socket.getOutputStream();

                byte[] arr1 = new byte[1024];
                int len1 = 0;
                while((len1 = inputStream.read(arr1)) != -1) {
                    output.write(arr1, 0, len1);
                    output.flush();
                }

                socket.shutdownOutput();

            }else {
                //给客户端一个响应：提示文件不存在
                System.out.println("未找到指定资源");
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            try {
                serverSocket.close();
            } catch (IOException e) {
                e.printStackTrace();

            }
```

```
            }
        }
    }
```

5.5 多个客户端和一个服务端通信【线程版本】

客户端

```java
public class Client {
    public static void main(String[] args) {
        /**
         * 1.客户端对象
         * 2.通过客户端向服务端发送消息【socket.getOutputStream()】
         * 3.接收服务端回复的消息【socket.getInputStream()】
         * 4.借助于Scanner
         * 5.循环的发送和接收消息【进行约定：bye/886则break出循环】
         * 6.释放资源
         */
        //1.构建客户端的对象
        Socket client = null;

        try {
            client = new Socket("10.31.165.42", 65500);

            //2.获取网络输入流和网络输出流
            OutputStream output = client.getOutputStream();
            InputStream input = client.getInputStream();

            //3.从控制台进行获取数据
            Scanner scanner = new Scanner(System.in);

            //4.进行循环的发送和接收消息
            while(true) {
                System.out.println("客户端对服务端说：");
                String message = scanner.nextLine();

                //4.1进行发送
                output.write(message.getBytes());
                output.flush();

                //4.2接收消息
                byte[] arr = new byte[1024];
                int len = input.read(arr);
                String reply = new String(arr, 0, len);
                System.out.println("收到服务端的反馈：" + reply);

                //4.3约定
                if(reply.equals("886") || reply.equals("bye") ||reply.equals("再见")){
                    break;
                }
            }
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        //5.释放资源

        finally {
```

```
            try {
                client.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

服务端

```java
public class Server {
    public static void main(String[] args) {
        /**
         * 1.创建服务端对象
         * 2.通过循环，为不同的客户端进行服务
         * 3.循环体内，创建线程
         * 4.线程的执行体中，进行发送消息和接收消息的操作
         *       读取客户端发送来的消息【socket.getInputStream()】
         *       回复给客户端的消息【socket.getOutputStream()】
         * 5.进行约定：何时停止循环
         * 6.释放资源
         */
        //1.创建服务端对象
        ServerSocket server = null;
        try {
            server = new ServerSocket(65500);

            //2.通过循环，为不同的客户端进行服务：循环体内，创建线程
            while(true) {
                //获取当前的线程对象
                Socket client = server.accept();
                ServerThread thread = new ServerThread(client);
                thread.start();
            }
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

//4.线程的执行体中，进行发送消息和接收消息的操作
//*          读取客户端发送来的消息【socket.getInputStream()】
//*          回复给客户端的消息【socket.getOutputStream()】
class ServerThread extends Thread {
    //定义一个成员变量，就是当前连接到的客户端
    private Socket client;
    public ServerThread(){}
    public ServerThread(Socket client) {
        this.client = client;
    }

    @Override
    public void run() {

        try {
            OutputStream output = client.getOutputStream();
            InputStream input = client.getInputStream();


            //3.从控制台进行获取数据
```

```
            Scanner scanner = new Scanner(System.in);

            //4.进行循环的发送和接收消息
            while(true) {
                //接收客户端发送来的消息
                byte[] arr = new byte[1024];
                int len = input.read(arr);
                String message = new String(arr, 0, len);
                System.out.println("来自客户端的消息：" + message);

                //回复消息
                System.out.println("服务端对客户端说：");
                String message1 = scanner.nextLine();
                output.write(message1.getBytes());
                output.flush();

                //约定
                if(message.equals("886") || message.equals("bye") ||message.equals("再见")){
                    break;
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            try {
                client.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

5.6 TCP实现注册登录

客户端

```
//客户端
public class Client {
    /**
     * 注册
     * @param userName 用户名
     * @param password  密码
     */
    public void doRegister(String userName,String password) {
        realAction(userName, password, "10.31.165.42", 6666);
    }

    /**
     * 登录
     * @param userName 用户名
     * @param password  密码
     */
    public void doLogin(String userName,String password) {
        realAction(userName, password, "10.31.165.42", 7777);
    }

    /**
     * 将用户名和密码发送给服务端
     * @param userName
     * @param password
     * @param ip
     * @param port
     */
    private void realAction(String userName,String password,String ip,int port) {

        Socket client = null;
        try {
            //1.实例化一个Socket的对象
            client = new Socket(ip, port);

            //2.获取输出流
            //缓冲字符流
            BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(client.getOutputStream()));

            //3.写入数据
            writer.write(userName + "," + password);//yang123123abc
            writer.flush();

        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        finally {

            try {
```

```
                client.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

服务端

```
//服务端
public class Server {
    private ServerSocket registerSocket;// 用来注册
    private ServerSocket loginSocket;// 用来登录

    // 将每次需要注册的用户名和密码以键值对的形式存储到配置文件中，结合Properties使用
    // 实例化一个Peroperties的对象
    private Properties userlist = new Properties();

    // 构造代码块
    {
        // 同步配置文件中的内容
        try {
            userlist.load(new BufferedInputStream(new FileInputStream(new
File("file/userlist.properties"))));
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    /**
     * 开启服务
     */
    public void start() {
        // 开启注册服务
        new Thread() {
            @Override
            public void run() {
                startRegisterService();
            }
        }.start();

        // 开启登录服务
        new Thread() {
            @Override
            public void run() {
                startLoginService();
            }
        }.start();
    }

    /**
     * 实现注册功能
     */
    private void startRegisterService() {

        try {
```

```
            //1.实例化一个ServerSocket的对象
            this.registerSocket = new ServerSocket(6666);

            while(true) {
                //2.获取连接到的客户端
                Socket socket = this.registerSocket.accept();
                System.out.println("连接到客户端" + socket);

                //3.获取客户端发送来的数据
                BufferedReader reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

                //4.读取
                String string = reader.readLine();//zhangsan,abc123
                System.out.println(string);
                //5.解析用户名和密码
                String[] arr = string.split(",");

                //6.注册
                //arr[0]:用户名
                //arr[1]:密码
                if(this.userlist.containsKey(arr[0]) && this.userlist.containsValue(arr[1])) {
                    System.out.println("该用户已经存在，无需注册");
                } else {
                    //新增一个用户
                    this.userlist.setProperty(arr[0], arr[1]);
                    //将用户名和密码同步到配置文件中进行持久化
                    this.userlist.store(new BufferedOutputStream(new FileOutputStream(new
File("file/userlist.properties"))), "add an user");
                    System.out.println("注册成功");
                }
            }

        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        finally{
            try {
                this.registerSocket.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }

    /**
     * 实现登录功能
     */
    private void startLoginService() {
```

```
        try {
            //1.实例化一个ServerSocket的对象
            this.loginSocket = new ServerSocket(7777);

            //2.获取连接到的客户端
            Socket socket = this.loginSocket.accept();

            //3.获取客户端发送来的用户名和密码
            BufferedReader reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

            //4.读取
            String string = reader.readLine();

            //5.解析用户名和密码
            String[] arr = string.split(",");

            //6.登录
            if(this.userlist.containsKey(arr[0])) {

                if(arr[1].equals(this.userlist.getProperty(arr[0]))) {
                    System.out.println("登录成功");
                } else {
                    System.out.println("用户名/密码错误");
                }

            } else {
                System.out.println("还未注册，请先注册");
                //startRegisterService();
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

第六节：UDP编程(了解)

6.1 概念

> User Datagram Protocol的简称，用户数据包协议，提供面向事务的简单不可靠信息传送服务
>
> 特点:
>
> a.不安全
>
> b.无连接
>
> c.效率高
>
> d.UDP传输数据时是有大小限制的，每个被传输的数据报必须限定在64KB之内

6.2 DatagramSocket和DatagramPacket

发送方

```
public class Sender {
    public static void main(String[] args) {
        //端口号表示的是指定的接收方的端口号，而发送方的端口是由系统自动分配的
        sendMessage("10.31.165.42", 6666,"你好啊");
    }

    /**
     * 发送数据
     * @param ip     指定接收方的ip地址
     * @param port   指定接收方的端口号
     * @param message 需要发送的信息
     */
    public static void sendMessage(String ip,int port,String message) {
        //1.实例化DatagramSocket的对象
        //注意：和流的使用类似，使用套接字完成之后需要关闭
        DatagramSocket socket = null;

        try {
            socket = new DatagramSocket();

            //2.将需要发送的数据封装为数据报包
            /**
             * DatagramPacket(byte[] buf, int length, InetAddress address, int port)
         构造数据报包，用来将长度为 length 的包发送到指定主机上的指定端口号。
             */
            DatagramPacket packet = new DatagramPacket(message.getBytes(), message.length(),
InetAddress.getByName(ip), port);

            //3.发送
            /**
             *  void send(DatagramPacket p)
         从此套接字发送数据报包。
             */
            //将数据写入网络的过程
            socket.send(packet);

        } catch (SocketException e) {
            // 父类为IOException
            e.printStackTrace();
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

接收方

```java
public class Receiver {
    public static void main(String[] args) {
        //1.实例化DatagramSocket的对象
        //需要进行绑定端口号：由发送方发送来的端口号进行决定
        /**
         *   DatagramSocket(int port)
         创建数据报套接字并将其绑定到本地主机上的指定端口。
         */
        DatagramSocket socket = null;
        try {
            socket = new DatagramSocket(6666);

            //2.将接收到的数据封装到数据报包中
            /**
             * DatagramPacket(byte[] buf, int length)
         构造 DatagramPacket，用来接收长度为 length 的数据包。
             */
            byte[] arr = new byte[1024];
            DatagramPacket packet = new DatagramPacket(arr, arr.length);

            System.out.println("等待接收数据~~~~~~~");

            //3.接收数据
            /**
             *  void receive(DatagramPacket p)
         从此套接字接收数据报包。
             */
            //注意：将数据从网络中读取出来
            socket.receive(packet);

            //4.获取发送方的详细信息
            //信息
            /**
             *  byte[] getData()
         返回数据缓冲区。
             */
            byte[] messages = packet.getData();
            String result = new String(messages, 0, messages.length);

            //获取发送方的ip地址
            /**
             * InetAddress getAddress()
         返回某台机器的 IP 地址，此数据报将要发往该机器或者是从该机器接收到的。
             */
            InetAddress address = packet.getAddress();
            String ip = address.getHostAddress();

            //获取消息是从发送发的哪个端口号发出来的
            /**
             *  int getPort()
         返回某台远程主机的端口号，此数据报将要发往该主机或者是从该主机接收到的。
```

```
             */
            int port = packet.getPort();

            System.out.println(ip + ":" + port + "说：" + result);

        } catch (SocketException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

第七节：课前默写

> 设计程序，使用多线程模拟3个售票员售卖50张票的功能

第八节：作业

> 实现一对多的简易聊天

第九节：面试题

> 1.简述网络编程需要的条件
>
> 2.简述UDP和TCP之间的区别