Day 11

C/S 结构:

全称为 Client/Server 结构，是指 客户端 和 服务器 结构

常见程序有 QQ，迅雷等软件

B/S 结构:

全称为 Browser/Server 结构，是指 浏览器 和 服务器 结构

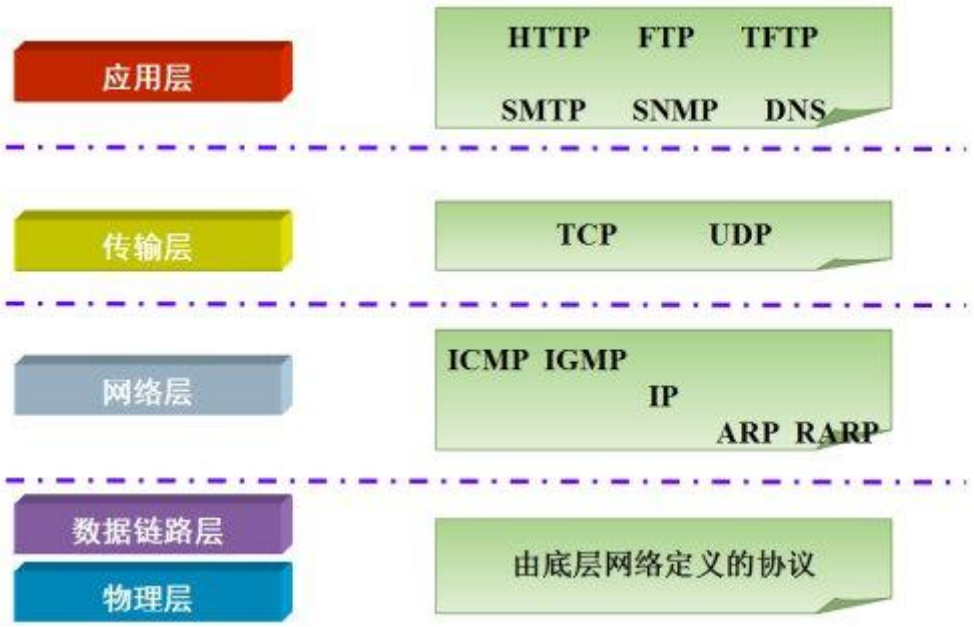
常见浏览器有 IE，谷歌，火狐等

网络通信协议:

通信协议是计算机必须遵守的规则，只有遵守这些规则，计算机之间才能进行通信.

TCP/IP 协议：传输控制协议

它定义了计算机如何连入因特网，以及数据如何在它们之间传输的标准.



UDP：用户数据报协议

特点:

1. 无连接的不可靠协议
2. 数据按包发送，64K 一个包
3. 速度快效率高，容易丢包

用于视频直播，网络电话

TCP：传输控制协议

特点:

1. 需要建立连接的可靠协议　电话
2. 数据传输无大小限制
3. 速度慢效率低　　重发机制

用于文件下载，浏览网页

TCP 通信的三次握手:

1. 客户端向服务端发送验证信息，等待服务器确认
2. 服务端收到验证信息后，回复客户端验证信息，同时发送自己的一条验证信息
3. 客户端收到服务端回复的信息，确认自己之前发的信息无误，并再次向服务器发回服务端的验证信息

网络编程三要素:

1. 通信协议　TCP
2. IP 地址
3. 端口号

IP 地址: 是网络中计算机的唯一标识

端口号: 计算机中进程的唯一标识

注意:

通信的两端是 2 个计算机中的 2 个程序在相互通信，所以 2 个程序都要有端口号.相互之间能识别就可以了

补充：

域名: 便于记忆的主机名称，可以解析到某个 IP 地址，这样通过域名就能找到对应的 IP 地址

模拟 TCP 通信程序

java.net.ServerSocket 类: TCP 服务端

// 构造方法

ServerSocket(int port): 创建一个 TCP 服务端，并监听指定端口

// 成员方法

Socket accept(): 监听数据，会阻塞. 收到数据后返回 Socket 对象

void close(): 关闭服务端 ServerSocket

java.net.Socket 类: TCP 客户端

// 构造方法

Socket(String ip, int port): 创建 TCP 客户端对象

// 成员方法

OutputStream getOutputStream(): 获取输出流对象，用于发送数据

InputStream getInputStream(): 获取输入流，用于接收数据

void shutdownOutput(): 关闭输出流，告知服务端数据发送完毕

void close(): 关闭客户端 Socket

TCP 严格区分为 客户端(Client) 与 服务端(Server)

注意：

服务器程序需要先启动，等待客户端连接

客户端主动连接服务器端，连接成功才能通信，服务器端不可以主动连接客户端

服务器端不需要创建新的 IO 流，通过 accept 得到一个客户端对象，使用客户端创建一个新的流

模拟编写 TCP 服务器端程序：

```
public static void main(String[] args) throws IOException {
    ServerSocket serversocket = new ServerSocket(8888);
    Socket so= serversocket.accept();
    InputStream is = so.getInputStream();
    byte[] bytes = new byte[1024];
    int read = is.read(bytes);
    System.out.println(new String(bytes,0,read));

    OutputStream os = so.getOutputStream();
    os.write("收到了谢谢".getBytes());


    so.close();
    serversocket.close();
}
```

模拟实现 TCP 客户端程序：

```
    public static void main(String[] args) throws IOException
{
        Socket socket = new Socket("127.0.0.1",8888);
        OutputStream os = socket.getOutputStream();
        os.write("你好服务器！".getBytes());

//        4.使用 Socket 对象中的方法 getInputStream() 获取网络字
节输入流 InputStream 对象
        InputStream is = socket.getInputStream();
//        5.使用网络字节输入流 InputStream 对象中的方法 read(),
读取服务器回写的数据
        byte[] bytes = new byte[1024];
        int read = is.read(bytes);
        System.out.println(new String(bytes,0,read));
        socket.close();
    }
```

文件上传案例

注意事项：

1. 上传的图片因为写入文件名相同，每次都会被覆盖，如何解决?
2. 上传一个文件服务端就结束了，如何让服务端不停止，一直接收文件上传
3. 在同一个线程读大文件可能会比较慢，能否利用多线程提高程序效率

具体实现：

服务器端

```
public static void main(String[] args) throws IOException {
    Random r = new Random();
    ServerSocket serverSocket = new ServerSocket(8888);
    System.out.println("图片上传服务器端已启动，等待客户端连
接...");
    while (true){
        Socket socket = serverSocket.accept();
        new Thread(new Runnable() {
            @Override
            public void run() {
                try{
                    File file = new File("e:\\donwload");
                    if (!file.exists()){
                        file.mkdirs();
                    }
                    InputStream is = socket.getInputStream();
                    FileOutputStream fos = new
FileOutputStream("e:\\donwload\\小岳岳
"+System.currentTimeMillis()+r.nextInt(99999)+".jpg");
                    byte[] bs = new byte[1024];
                    int s ;
                    while ((s=is.read(bs))!=-1){
                        fos.write(bs,0,s);
                    }
                    OutputStream os = socket.getOutputStream();
                    os.write("上传成功！".getBytes());
                    fos.close();
                    socket.close();
                }catch (IOException e){
                    System.out.println(e);
                }
```

```
            }
        }).start();
    }
}
```

客户端：

```
public static void main(String[] args) throws IOException {
    Socket socket = new Socket("192.168.14.250",9999);

    FileInputStream fis = new FileInputStream("E:\\小岳岳.jpg");

    byte[] bytes = new byte[1024];
    int c ;
    OutputStream os = socket.getOutputStream();
    while ((c = fis.read(bytes))!=-1){
        os.write(bytes,0,c);
    }
    socket.shutdownOutput();

    InputStream is = socket.getInputStream();
    byte[] bs = new byte[1024];
    int read = is.read(bs);
    System.out.println(new String(bs,0,read));
    os.close();
    fis.close();
}
```

改进：当同时上传的用户太多时，同时创建太多新的进程会耗费很多资源，可以采用**线程池**的方法解决这个问题。

通过代码来实现 BC 模式浏览器与服务器之间的交互：

```
public static void main(String[] args) throws IOException {
    ServerSocket server = new ServerSocket(8080);
    while (true){
        Socket socket = server.accept();
        new Thread(()->{
            FileInputStream fis = null;
            OutputStream os = null;
            try{
                InputStream is = socket.getInputStream();
                BufferedReader br = new BufferedReader(new
InputStreamReader(is));
                String s = br.readLine();
                String[] arr = s.split(" ");
```

```
                String s1 = arr[1].substring(1);
                System.out.println(s1);
                fis = new FileInputStream(s1);
                os = socket.getOutputStream();
                os.write("HTTP/1.1 200 OK\r\n".getBytes());
                os.write("Content-Type:
text/html\r\n".getBytes());
                os.write("\r\n".getBytes());
                byte[] bs = new byte[1024];
                int c;
                while ((c=fis.read(bs))!=-1){
                    os.write(bs,0,c);
                }
            }catch (IOException e){
                e.printStackTrace();
            }finally {
                if (fis!=null){
                    try {
                        fis.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
                if (os!=null){
                    try {
                        os.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }
        }).start();
    }
}
```