网络编程概述

2018年8月27日 12:22

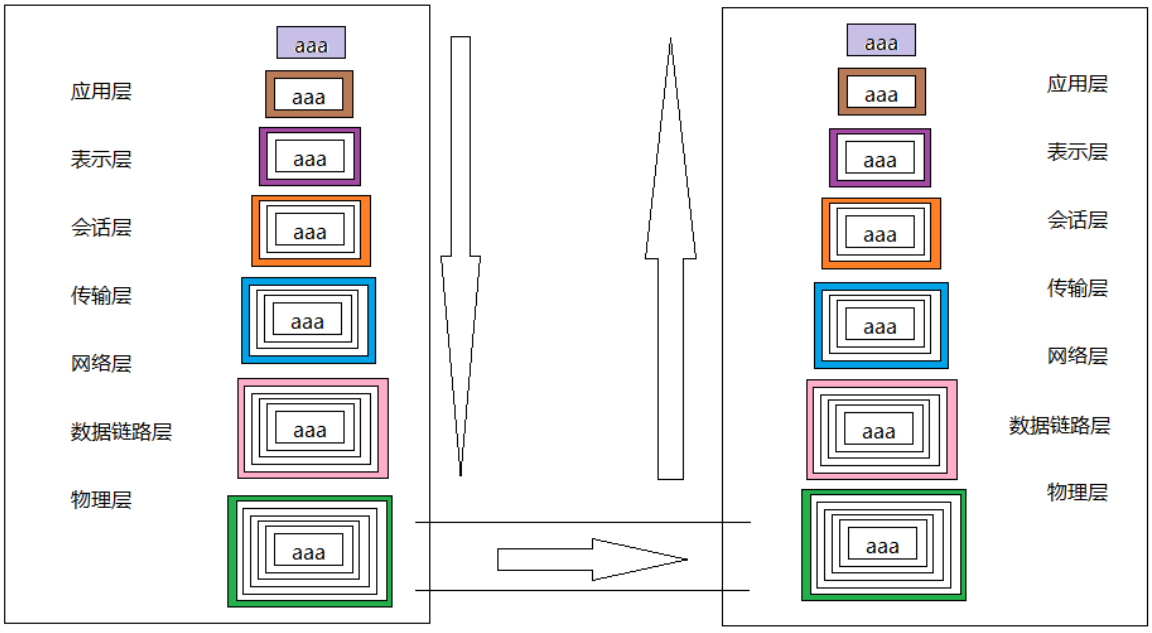
1. 网络编程概述

计算可以通过网络连接，组成计算机网络，计算机之间可以通过网络进行通信，传递信息。很多应用程序也都具有网络通信能力。而java也提供了开发网络程序的编程能力，这就称之为java的网络编程

2. 网络编程的基本概念 - 网络模型

OSI七层网络模型

物理层 数据链路层 网络层 传输层 会话层 表示层 应用层



3. 网络编程的基本概念 - 协议

网络中的计算机想要互相通信，必须遵循相同的沟通方式，需要提前约定，这样提前约定的沟通方式，称之为网络协议，由于网络是分层的，每层之间都有数据要传递，一般的协议都是为某一个层数据的通信来订立的，所以一般来说一个协议通常是归属于某一层的，每一个层也有若干的协议来约定通信规则

协议又可以分为公有协议 和 私有协议

公有协议是由国际化标准组织订立的，全世界的计算机都去遵循

应用层：HTTP HTTPS FTP SMTP POP3

传输层：TPC UDP

网络层：IP协议

私有协议是公司 组织 团队 个人 自己约定的协议 只在遵循该协议的小范围内起作用

4. 网络编程的基本概念 - IP

IP协议目前有两个版本：

IPV4:

0~255 : 0~255 : 0~255 : 0~255

其中如下网段的地址比较特殊，是内网地址：

10.0.0.0 - 10.225.225.255
172.16.0.0 - 172.31.255.255
192.168.0.0 - 192.168.255.255

特殊的ip：

本地回环地址：

127.0.0.1
0.0.0.0

广播地址

xxx.xxx.xxx.255

IPV6:

128位的二进制表示的地址，可以表示2^128个地址

5. 网络编程基本概念 - 端口

每个计算机 除了可以分配到一个IP以外 还会划分出2^16个端口

需要网络通信的软件 可以 来占用一个端口 通过 ip：端口 在指定ip的指定端口上进行通信

虽然ip只有一个，但是端口有很多，所以可以在一个ip上利用不同端口 实现同时进行多个通信的效果

0~65535

其中0~1024的端口是计算机预留的端口 普通程序不可以占用

其他端口应用程序随便占用，先到先得，同一时间一个端口只能有一个程序占

用，所以用完后端口会被释放，其他程序才可以再次占用。

6. 主机名 域名 DNS服务器 Hosts文件

主机名

IP地址可以表示网络中的主机 但是ip不易记忆 所以一帮都会选择为当前主机 指定主机名

域名

主机名是可能重复的 为了防止在公网上主机名重复，有了域名的概念，域名需要统一到域名管理组织中注册，从而防止重复

DNS服务器

网络中有 DNS服务器中可以 帮我们将 主机名或域名翻译成对应ip

Hosts文件

可以在本地的Hosts文件中模拟DNS的功能

windows下：

C:\Windows\System32\drivers\etc\Hosts

Linux下：

/etc/hosts

7. 套接字编程 socket编程

为了能够使开发人员开发网络相关的程序，操作系统为开发者提供了网络编程的接口，通过这套接口可以开发基于网络层 和 传输层的 代码 从而实现网络通信。物理层 和 数据链路层 由操作系统负责，不需要开发人员关注，会话层 表示层 应用层 当中的需求 则需要开发人员根据需要自己来实现。

这套操作系统提供的网络编程的接口 称之为socket - 套接字编程

**套接字不是协议 只是一套编程接口 不要搞混

Java网络开发 - IP地址

2018年8月27日 16:12

1. 代表IP地址的类

继承结构：

java.net

类 InetAddress

在java中代表IP地址

重要方法：

static InetAddress	**getLocalHost**() 返回本地主机。
static InetAddress	**getByName**(String host) 在给定主机名的情况下确定主机的 IP 地址。
static InetAddress	**getByAddress**(byte[] addr) 在给定原始 IP 地址的情况下，返回 InetAddress 对象。

String	**getHostName**() 获取此 IP 地址的主机名。
String	**getHostAddress**() 返回 IP 地址字符串（以文本表现形式）。
byte[]	**getAddress**() 返回此 InetAddress 对象的原始 IP 地址。

2. InetSocketAddress

继承结构

java.net

类 SocketAddress

java.net

类 InetSocketAddress

代表socket通信过程中的IP地址

重要方法

构造方法摘要
InetSocketAddress(InetAddress addr, int port) 根据 IP 地址和端口号创建套接字地址。
InetSocketAddress(int port) 创建套接字地址，其中 IP 地址为通配符地址，端口号为指定值。
InetSocketAddress(String hostname, int port) 根据主机名和端口号创建套接字地址。

InetAddress	**getAddress**() 获取 InetAddress。
String	**getHostName**() 获取 hostname。
int	**getPort**() 获取端口号。

Java网络开发 - Socket编程 - UPD编程

2018年8月27日 16:33

1. UDP协议概述

UDP是TCP协议中非常重要和常用的通信协议，可以实现不可靠的网络通信

特点：

不需要创建连接

数据以独立的数据包的形式发送 每个数据包最大64KB

传输过程中 不保证数据一定可以到达 也不保证接受的到的数据包的顺序和发送时一致

速度比较快

~类似于飞鸽传书

在速度要求比较高 可靠性要求比较低 的场景下优先使用

2. java中的udp实现

继承结构

java.net

类 DatagramSocket

代表UDP通信的一个端

重要方法

构造方法摘要	
	DatagramSocket() 构造数据报套接字并将其绑定到本地主机上任何可用的端口。
	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(SocketAddress bindaddr) 创建数据报套接字，将其绑定到指定的本地套接字地址。

void	**send**(DatagramPacket p) 从此套接字发送数据报包。
void	**receive**(DatagramPacket p) 从此套接字接收数据报包。

继承结构

java.net

类 DatagramPacket

重要方法

构造方法摘要
DatagramPacket(byte[] buf, int length) 构造 DatagramPacket，用来接收长度为 length 的数据包。

byte[]	**getData**() 返回数据缓冲区。
int	**getLength**() 返回将要发送或接收到的数据的长度。
SocketAddress	**getSocketAddress**() 获取要将此包发送到的或发出此数据报的远程主机的 SocketAddress（通常为 IP 地址 + 端口号）。
void	**setSocketAddress**(SocketAddress address) 设置要将此数据报发往的远程主机的 SocketAddress（通常为 IP 地址 + 端口号）。
void	**setData**(byte[] buf) 为此包设置数据缓冲区。
void	**close**() 关闭此数据报套接字。

案例：实现UDP聊天案例

```
package cn.tedu.net.udp.chat;

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetSocketAddress;
import java.util.Scanner;

public class ChatClient {
    public static void main(String[] args) {
        new Thread(new Sender()).start();
```

```
            new Thread(new Receiver()).start();
        }
    }

    /**
     * 聊天消息接受者
     */
    class Receiver implements Runnable{
        @Override
        public void run() {
            DatagramSocket ds = null;
            try {
                //1.创建接收端
                ds = new DatagramSocket(44444);
                while(true){
                    //2.接受数据包
                    byte [] data = new byte[1024];
                    DatagramPacket dp = new DatagramPacket(data, data.length);
                    ds.receive(dp);
                    //3.获取接收到的信息
                    String msg = new String(data,0,dp.getLength());
                    String ip = dp.getAddress().getHostAddress();
                    int port = dp.getPort();
                    System.err.println("==收到来自["+ip+":"+port+"]的消息，消息内容
                    为： “"+msg+"”========");
                }
            } catch (Exception e) {
                e.printStackTrace();
                throw new RuntimeException(e);
            } finally {
                //4.关闭接收端
                if(ds != null){
                    ds.close();
                }
            }
        }
    }

    /**
     * 聊天消息发送者
     */
    class Sender implements Runnable{
        @Override
        public void run() {
            Scanner scan = null;
            DatagramSocket ds = null;
            try {
```

```
            //1.创建发送端
            ds = new DatagramSocket();
            //2.创建控制台扫描器
            scan = new Scanner(System.in);
            while(true){
                //3.读取控制台 消息格式 [ip#端口#消息]
                System.out.println("===请输入要发送的消息，格式为[ip#端口#消息]：
                ================");
                String line = scan.nextLine();
                String attrs [] = line.split("#");
                String ip = attrs[0];
                int port = Integer.parseInt(attrs[1]);
                String msg = attrs[2];
                //4.发送数据
                DatagramPacket dp = new DatagramPacket(msg.getBytes(),
                msg.getBytes().length);
                dp.setSocketAddress(new InetSocketAddress(ip, port));
                ds.send(dp);
            }
        } catch (Exception e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        } finally {
            if(ds!=null){
                ds.close();
            }
            if(scan!=null){
                scan.close();
            }
        }
    }
}
```

Java网络开发 - Socket编程 - TCP编程

2018年8月27日 16:38

1. TCP协议概述

TCP是TCP协议中非常重要和常用的通信协议，可以实现可靠的网络通信

特点：

需要创建连接 需要三次握手

底层建立的连接流 数据包以流的方式传递 没有传输数据量大小的限制

传输过程中 可以保证数据一定不会丢 也不会多 也可以保证 顺序的一致

速度比较慢

在可靠性要求比较高 速度要求比较低 的场景下优先使用

2. Java中实现TCP

在TCP通信中，通信的过程需要两端的参与，其中发起请求的端称之为客户端 被动等待请求的端称之为服务器端

代表TCP通信中的客户端的类

java.net

类 Socket

构造方法摘要	
	Socket() 通过系统默认类型的 SocketImpl 创建未连接套接字
	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。
	Socket(String host, int port) 创建一个流套接字并将其连接到指定主机上的指定端口号。

void	**connect**(SocketAddress endpoint) 将此套接字连接到服务器。
InputStream	**getInputStream**() 返回此套接字的输入流。
OutputStream	**getOutputStream**()

	返回此套接字的输出流。
void	**close**() 关闭此套接字。

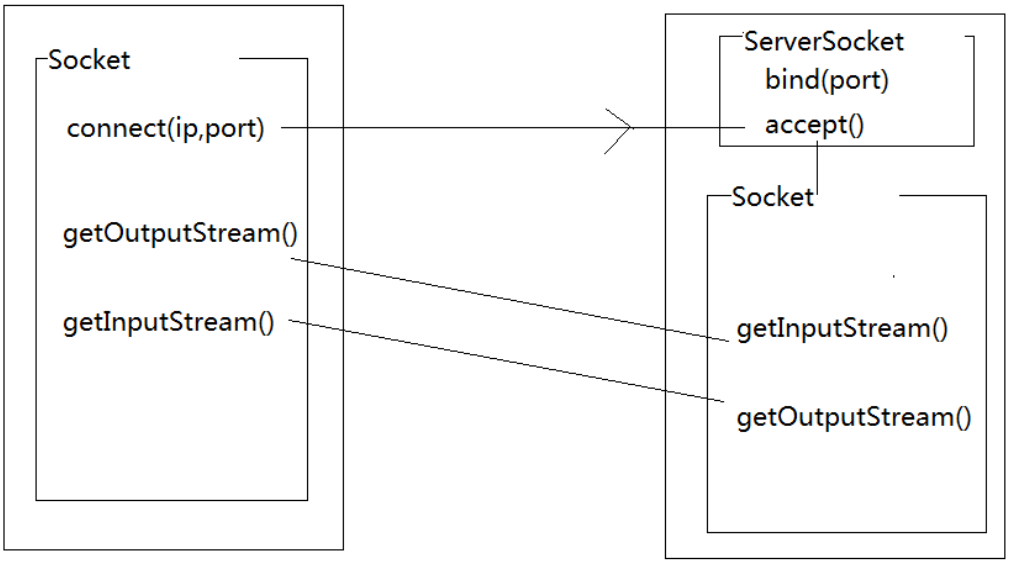
代表TCP通信中服务端的类

java.net

类 ServerSocket

构造方法摘要
ServerSocket() 创建非绑定服务器套接字。
ServerSocket(int port) 创建绑定到特定端口的服务器套接字。

void	**bind**(SocketAddress endpoint) 将 ServerSocket 绑定到特定地址（IP 地址和端口号）。
Socket	**accept**() 侦听并接受到此套接字的连接。
void	**close**() 关闭此套接字。

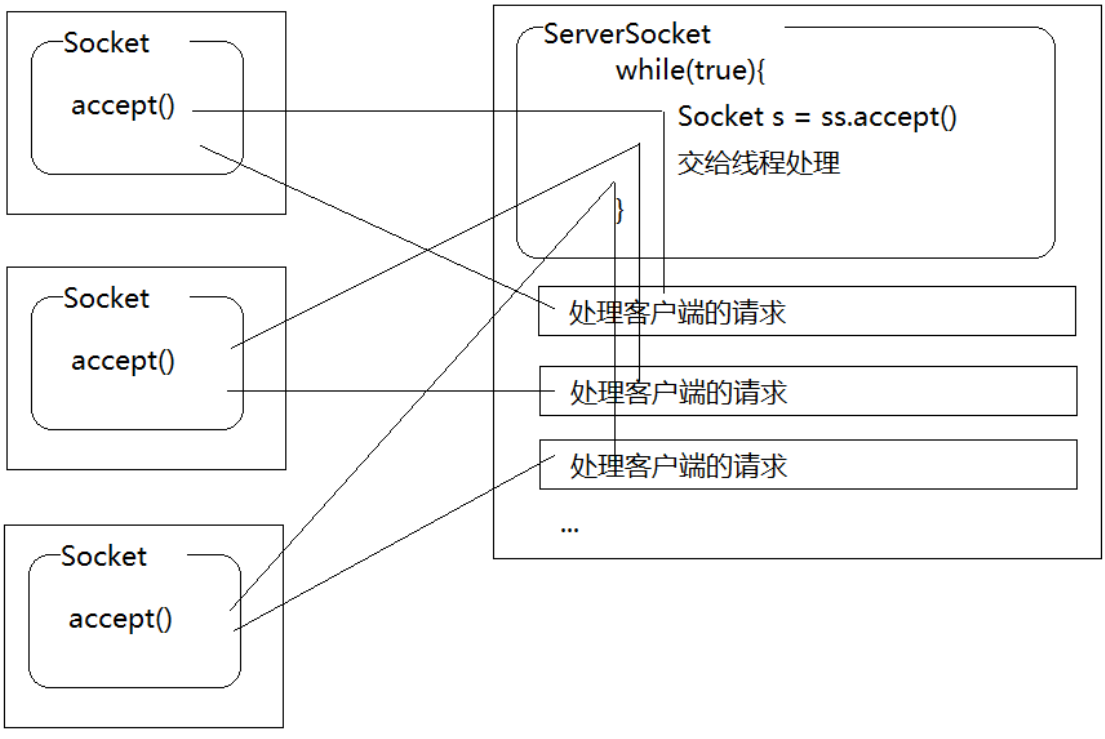


案例：实现TPC通信

```
package cn.tedu.net;

import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.net.Socket;

/**
 * TCP通信的客户端
 */
public class Demo01Client {
    public static void main(String[] args) throws Exception {
        //1.创建Socket
        Socket socket = new Socket();
        //2.连接服务器
        socket.connect(new InetSocketAddress("127.0.0.1", 44444));
        //3.从客户端发送消息给服务端
        OutputStream out = socket.getOutputStream();
        //4.向服务端发送数据
        String msg = "hello world~ ";
        out.write(msg.getBytes());
        out.flush();
        //5.从服务端接受数据
        InputStream in = socket.getInputStream();
        byte [] data = new byte[1024];
        int len = in.read(data);
        String msg2 = new String(data,0,len);
        System.out.println(msg2);
        //6.关闭套接字
        socket.close();
    }
}


package cn.tedu.net;

import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.net.ServerSocket;
import java.net.Socket;

/**
 * TCP通信的服务端
 */
```

```
public class Demo01Server {
    public static void main(String[] args) throws Exception {
        //1.创建服务端
        ServerSocket ss = new ServerSocket();
        //2.绑定指定端口
        ss.bind(new InetSocketAddress(44444));
        //3.等待客户端连接
        Socket socket = ss.accept();
        //4.接受客户端的数据
        InputStream in = socket.getInputStream();
        byte [] data = new byte[1024];
        int len = in.read(data);
        String str = new String(data,0,len);
        System.out.println(str);
        //5.向客户端返回数据
        String msg = "hello net~";
        OutputStream out = socket.getOutputStream();
        out.write(msg.getBytes());
        out.flush();
        //6.关闭套接字
        socket.close();
        ss.close();
    }
}
```

3. 在tcp网络通信中应用多线程技术实现一个服务端为多个客户端提供服务



案例：实现文件上传服务器 并且 通过多线程技术来实现同时处理多个客户端的效果

```
package cn.tedu.net;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.UUID;

/**
 * TCP案例：通过TCP实现文件上传 - 服务端代码
 */

public class Demo02UploadServer {
    public static void main(String[] args) throws Exception {
        //1.创建服务端
        ServerSocket ss = new ServerSocket();
        ss.bind(new InetSocketAddress(44444));
        System.out.println("###千度网盘开始运行了###");
        //2.不停接受客户端连接，一旦连接成功，交给线程处理
        while(true){
            Socket socket = ss.accept();
            new Thread(new UploadRunnable(socket)).start();
        }
    }
}

class UploadRunnable implements Runnable{
    private Socket socket = null;
    public UploadRunnable(Socket socket) {
        this.socket  = socket;
    }

    @Override
    public void run() {
        OutputStream out = null;
        try {
            //1.获取socket输入流
            InputStream in = socket.getInputStream();
            //2.创建文件输出流指向输出位置
            String path = "upload/"+UUID.randomUUID().toString()+".data";
            out = new FileOutputStream(path);
            //3.对接流
```

```
                byte [] data = new byte[1024];
                int i = 0;
                while((i = in.read(data))!=-1){
                    out.write(data,0,i);
                }
                System.out.println("接收到了来自
                ["+socket.getInetAddress().getHostAddress()+"]的上传文件["+path+"]");
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                //4.关闭资源
                if(out!=null){
                    try {
                        out.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    } finally {
                        out = null;
                    }
                }
                if(socket != null){
                    try {
                        socket.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    } finally {
                        socket = null;
                    }
                }
            }
        }
}

package cn.tedu.net;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.util.Scanner;

/**
 * TCP案例：通过TCP实现文件上传 - 客户端代码
 */
```

```
public class Demo02UploadClient {
    public static void main(String[] args) {
        Scanner scanner = null;
        InputStream in = null;
        Socket socket = null;
        try {
            //1.要求用户输入文件路径
            scanner = new Scanner(System.in);
            System.out.println("--请输入要上传的文件的路径：");
            String path = scanner.nextLine();
            File file = new File(path);
            //2.只有文件存在 且 是一个文件才上传
            if(file.exists() && file.isFile()){
                //2.创建连接文件的输入流
                in = new FileInputStream(file);
                //3.创建TCP客户端对象
                socket = new Socket();
                //4.连接TCP服务端
                socket.connect(new InetSocketAddress("127.0.0.1",44444));
                //5.获取到TCP服务端的输出流
                OutputStream out = socket.getOutputStream();
                //6.对接流 发送文件数据给服务端
                byte [] data = new byte[1024];
                int i = 0;
                while((i = in.read(data))!=-1){
                    out.write(data,0,i);
                }
            }else{
                throw new RuntimeException("文件不存在 或者是一个文件夹~~");
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally{
            //7.关闭扫描器 关闭文件输入流 关闭套接字
            if(scanner != null){
                scanner.close();
            }
            if(in != null){
                try {
                    in.close();
                } catch (IOException e) {
                    e.printStackTrace();
                } finally {
                    in = null;
                }
            }
```

```
                if(socket!=null){
                    try {
                        socket.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    } finally {
                        socket = null;
                    }
                }
            }
        }
    }
```

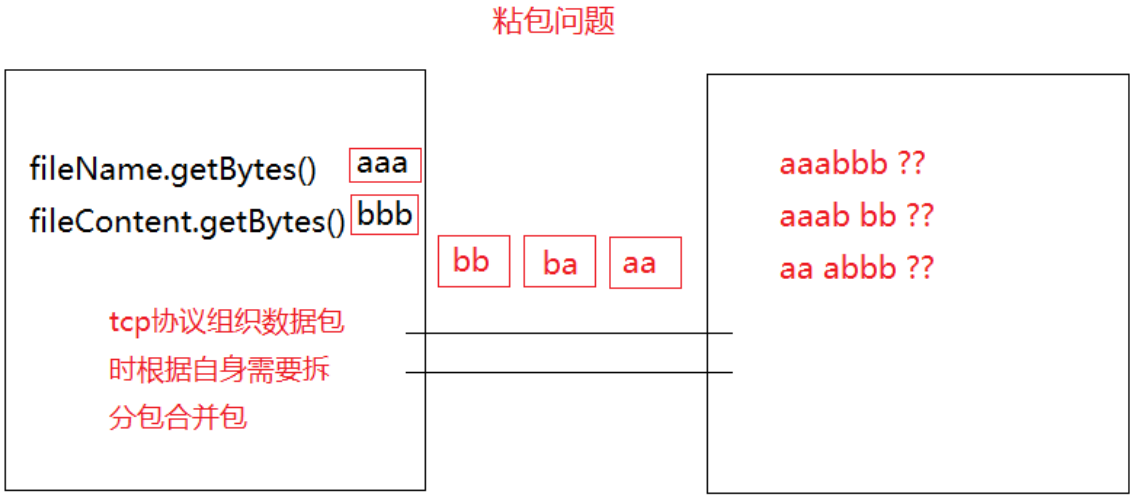
4. TCP通信中的粘包问题

a. 粘包问题概述

TCP通信中 如果连续传输多段数据 ，TCP在传输的过程中，会根据需要自动的拆分包合并包，造成数据边界信息丢失了，在接收端收到数据后无法判断数据的边界在哪里，这样的问题就称之为TCP通信中的粘包问题。

粘包问题的本质在于TCP协议是传输层的协议，而数据边界判断的问题本质上是会话层的问题，TCP协议并没有给予解决方案。

而socket编程是给予网络层 和 传输层的编程，没有会话层的功能提供，所以在socket编程中粘包问题需要程序开发人员自己想办法解决



b. 粘包问题解决方案

i. 只传输固定长度的数据

能解决粘包问题，但是程序的灵活性非常低，只能在每次传输的数据长度都一致的情况下使用，应用的场景比较少

ii. 约定分隔符

能解决粘包问题，但是如果数据本身包含分隔符，则需要进行转义。转义的过程比较麻烦，浪费时间，代码写起来也比较复杂

iii. 使用协议

在通信双发原定数据传输的格式 发送方严格按照格式发送 接收方严格按照格式接收 从而根据格式本身判断数据的边界

协议又分为公有协议 和 私有协议

使用公有协议：

利用会话层 传输层 应用层 的公有协议的规则 来传输数据 判断边界

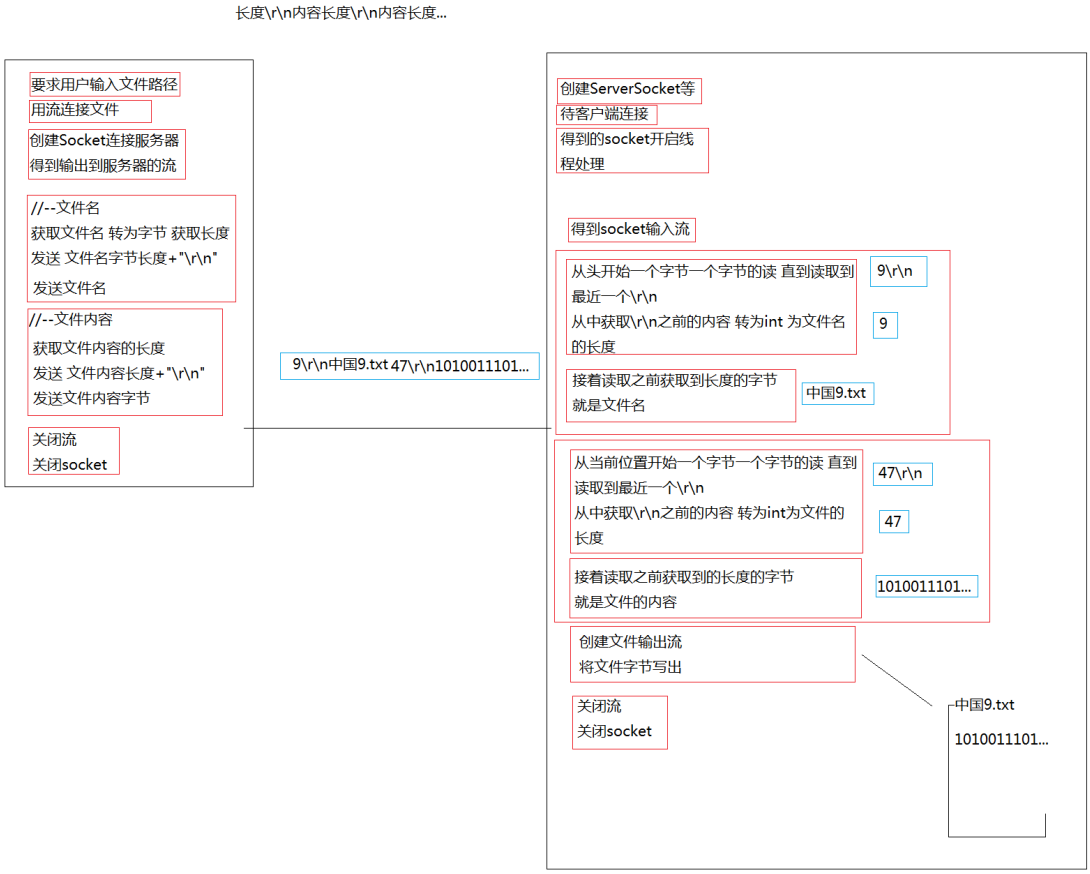
在全世界范围内通用 但协议相对复杂

使用私有协议：

自己来约定传输双方使用的格式 从而来判断边界

协议可以根据需要定制 但只在有限的小范围内有效

案例：改造文件上传案例 通过自定义协议解决粘包问题 实现传输文件同时传输文件名



协议格式： [文件名长度]\r\n[文件名][文件内容长度]\r\n[文件内容]

```
package cn.tedu.net;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
```

```
import java.net.InetSocketAddress;
import java.net.Socket;
import java.util.Scanner;

/**
 * TCP案例：通过TCP实现文件上传 - 客户端代码
 */
public class Demo02UploadClient {
    public static void main(String[] args) {
        Scanner scanner = null;
        InputStream in = null;
        Socket socket = null;
        try {
            //1.要求用户输入文件路径
            scanner = new Scanner(System.in);
            System.out.println("--请输入要上传的文件的路径：");
            String path = scanner.nextLine();
            File file = new File(path);
            //2.只有文件存在 且 是一个文件才上传
            if(file.exists() && file.isFile()){
                //2.创建连接文件的输入流
                in = new FileInputStream(file);
                //3.创建TCP客户端对象
                socket = new Socket();
                //4.连接TCP服务端
                socket.connect(new InetSocketAddress("127.0.0.1",44444));
                //5.获取到TCP服务端的输出流
                OutputStream out = socket.getOutputStream();
                //6.1向服务器发送[文件名字节长度\r\n]
                out.write((file.getName().getBytes().length+"\r\n").getBytes());
                //6.2向服务器发送[文件名字节]
                out.write(file.getName().getBytes());
                //6.3向服务器发送[文件内容字节长度\r\n]
                out.write((file.length()+"\r\n").getBytes());
                //6.4向服务器发送[文件内容字节]
                byte [] data = new byte[1024];
                int i = 0;
                while((i = in.read(data))!=-1){
                    out.write(data,0,i);
                }
            }else{
                throw new RuntimeException("文件不存在 或者是一个文件夹~~");
            }
        } catch (Exception e) {
```

```
                e.printStackTrace();
            } finally{
                //7.关闭扫描器 关闭文件输入流 关闭套接字
                if(scanner != null){
                    scanner.close();
                }
                if(in != null){
                    try {
                        in.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    } finally {
                        in = null;
                    }
                }
                if(socket!=null){
                    try {
                        socket.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    } finally {
                        socket = null;
                    }
                }
            }
        }
}
package cn.tedu.net;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.net.ServerSocket;
import java.net.Socket;

/**
 * TCP案例：通过TCP实现文件上传 - 服务端代码
 */

public class Demo02UploadServer {
    public static void main(String[] args) throws Exception {
        //1.创建服务端
        ServerSocket ss = new ServerSocket();
        ss.bind(new InetSocketAddress(44444));
        System.out.println("###千度网盘开始运行了###");
        //2.不停接受客户端连接，一旦连接成功，交给线程处理
```

```
            while(true){
                Socket socket = ss.accept();
                new Thread(new UploadRunnable(socket)).start();
            }
        }
    }

    class UploadRunnable implements Runnable{
        private Socket socket = null;
        public UploadRunnable(Socket socket) {
            this.socket  = socket;
        }

        /**
         * 通过私有协议传输数据 协议的格式为 [文件名长度\r\n文件名 文件长度\r\n文件内容]
         */
        @Override
        public void run() {
            OutputStream out = null;
            try {
                //1.获取socket输入流
                InputStream in = socket.getInputStream();
                //2.获取文件名 - 读到第一个回车换行之前 截取出文件名的长度 接着读取这个长度的字节 就是文件名
                //--读取数据 直到遇到第一个回车换行
                //----每次从流中读取一个字节 转成字符串 拼到line上 只要line还不是\r\n结尾 就重复这个过程
                String line = "";
                byte [] tmp = new byte[1];
                while(!line.endsWith("\r\n")){
                    in.read(tmp);
                    line += new String(tmp);
                }
                //----读取到了 文件名长度\r\n 截掉\r\n 转成int 就是文件名的长度
                int len = Integer.parseInt(line.substring(0, line.length()-2));
                //----从流中接着读 len个字节 就是文件名
                byte [] data = new byte[len];
                in.read(data);
                String fname = new String(data);

                //3.读取文件内容  - 读到下一个回车换行之前 截取出文件内容的长度 接着读取这个长度的字节 就是文件内容
                String line2 = "";
                byte [] tmp2 = new byte[1];
```

```
                while(!line2.endsWith("\r\n")){
                    in.read(tmp2);
                    line2 += new String(tmp2);
                }
                //----读取到了 文件长度\r\n 截掉\r\n 转成int 就是文件的长度
                int len2 = Integer.parseInt(line2.substring(0, line2.length()-2));
                //4.从流中读取 文件长度个字节 就是文件内容 输出到文件中
                byte data2 [] = new byte[len2];
                in.read(data2);

                //5.创建文件输出流指向输出位置,将数据写出到流中
                String path = "upload/"+fname;
                out = new FileOutputStream(path);
                out.write(data2);
                System.out.println("接收到了来自
                ["+socket.getInetAddress().getHostAddress()+"]的上传文件["+path+"]");
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                //6.关闭资源
                if(out!=null){
                    try {
                        out.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    } finally {
                        out = null;
                    }
                }
                if(socket != null){
                    try {
                        socket.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    } finally {
                        socket = null;
                    }
                }
            }
        }
    }
```