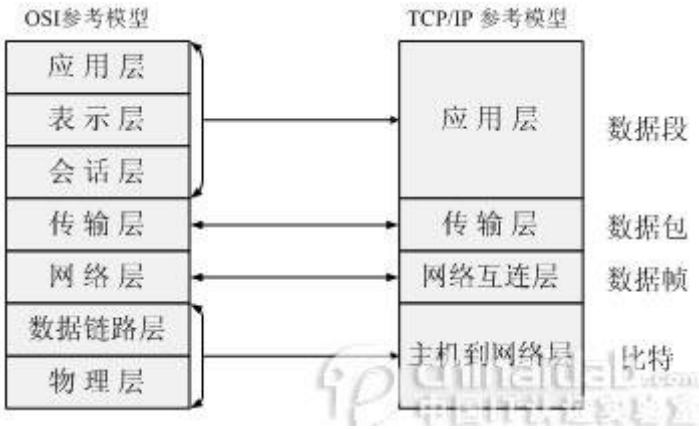
Socket

网络基础

计算机网络分层



OSI七层网络模型(从下往上)：

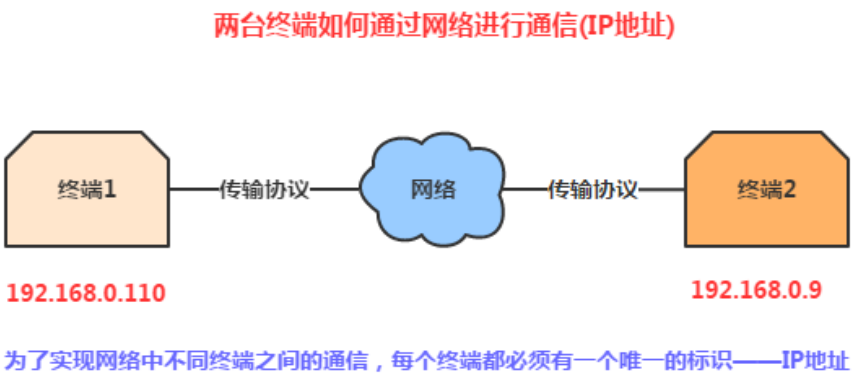
> - *物理层(Physical)：设备之间的数据通信提供传输媒体及互连设备，为数据传输提供可靠的 环境。可以理解为网络传输的物理媒体部分，比如**网卡，网线，集线器，中继器，调制解调器**等！在这一层，数据还没有被组织，仅作为原始的位流或电气电压处理，这一层的单位是:**bit比特***
> - *数据链路层(Datalink)：可以理解为数据通道，主要功能是如何在不可靠的物理线路上进行 数据的可靠传递，改层作用包括：物理地址寻址，数据的成帧，流量控制，数据检错以及重发等！ 另外这个**数据链路指的是**：物理层要为终端设备间的数据通信提供传输媒体及其连接。媒体是 长期的，连接是有生存期的。在连接生存期内，收发两端可以进行不等的一次或多次数据通信。 每次通信都要经过建立通信联络和拆除通信联络两过程！这种建立起来的**数据收发关系**~ 该层的设备有：**网卡，网桥，网路交换机**，另外该层的单位为：**帧***
> - *网络层(Network)：主要功能是将网络地址翻译成对应的物理地址，并决定如何将数据从发 送方路由到接收方，所谓的路由与寻径：一台终端可能需要与多台终端通信，这样就产生的了 把任意两台终端设备数据链接起来的问题！简单点说就是：建立网络连接和为上层提供服务！ 该层的设备有：**路由**！该层的单位为：**数据包**，另外IP协议就在这一层！*
> - *传输层(Transport)：向上面的应用层提供通信服务，面向通信部分的最高层，同时也是 用户功能中的最低层。接收会话层数据，在必要时将数据进行分割，并将这些数据交给网络 层，并且保证这些数据段有效的到达对端！所以这层的单位是：**数据段**；而这层有两个很重要 的协议就是：**TCP传输控制协议**与**UDP用户数据报协议**，这也是本章节核心讲解的部分！*
> - *会话层(Session)：负责在网络中的两节点之间建立、维持和终止通信。建立通信链接， 保持会话过程通信链接的畅通，同步两个*

> *节点之间的对话，决定通信是否被中断以及通信中断时 决定从何处重新发送，即不同机器上的用户之间会话的建立及管理！*
> - *表示层(Presentation)：对来自应用层的命令和数据进行解释，对各种语法赋予相应 的含义，并按照一定的格式传送给会话层。其主要功能是"处理用户信息的表示问题，如编码、 数据格式转换和加密解密，压缩解压缩"等*
> - *应用层(Application)：OSI参考模型的最高层，为用户的应用程序提供网络服务。 它在其他6层工作的基础上，负责完成网络中应用程序与网络操作系统之间的联系，建立与结束使用者之间的联系，并完成网络用户提出的各种网络服务及应用所需的监督、管理和服务等各种协议。此外，该层还负责协调各个应用程序间的工作。应用层为用户提供的服务和协议有：文件服务、目录服务、文件传输服务（FTP）、远程登录服务（Telnet）、电子邮件服务（E-mail）、打印服务、安全服务、网络管理服务、数据库服务等。*

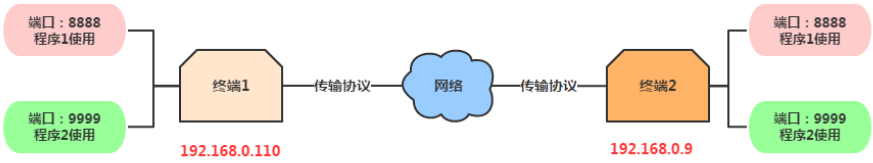
> *TCP/IP是一组协议的代名词，它还包括许多协议，组成了TCP/IP协议簇。 TCP/IP协议簇分为四层，IP位于协议簇的第二层(对应OSI的第三层)，TCP位于协议簇的第三层(对应OSI的第四层)。TCP/IP通讯协议采用了4层的层级结构，每一层都呼叫它的下一层所提供 的网络来完成自己的需求。这4层分别为：*
>
> - *应用层：应用程序间沟通的层，如简单电子邮件传输（SMTP）、文件传输协议（FTP）、 网络远程访问协议（Telnet）等。*
> - *传输层：在此层中，它提供了节点间的数据传送服务，如传输控制协议（TCP）、 用户数据报协议（UDP）等，TCP和UDP给数据包加入传输数据并把它传输到下一层中， 这一层负责传送数据，并且确定数据已被送达并接收。*
> - *网络互连层：负责提供基本的数据封包传送功能，让每一块数据包都能够到达目 的主机（但不检查是否被正确接收），如网际协议（IP）。*
> - *主机到网络层：对实际的网络媒体的管理，定义如何使用实际网络 （如Ethernet、Serial Line等）来传送数据。*

IP地址&端口号

IP地址



端口号



端口号规定为16位，即允许一个IP主机有2的16次方65535个不同的端口。其中：

- 0~1023：分配给系统的端口号

> *我们不可以乱用 常用协议使用的端口：HTTP:80，FTP：21，TELNET：23*

- 1024~49151：登记端口号，主要是让第三方应用使用

> *但是必须在IANA（互联网数字分配机构）按照规定手续登记，*

- 49152~65535：短暂端口号，是留给客户进程选择暂时使用，一个进程使用完就可以供其他进程使用。

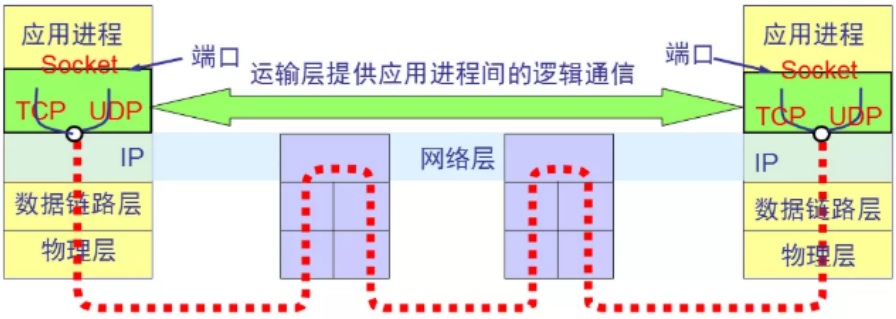
> *在Socket使用时，可以用1024~65535的端口号*

C/S结构

- 定义：即客户端/服务器结构，是软件系统体系结构
- 作用：充分利用两端硬件环境的优势，将任务合理分配到Client端和Server端来实现，降低了系统的通讯开销。

> *Socket正是使用这种结构建立连接的，一个套接字接客户端，一个套接字接服务器。*

如图：



可以看出，Socket的使用可以基于TCP或者UDP协议。

TCP&UDP

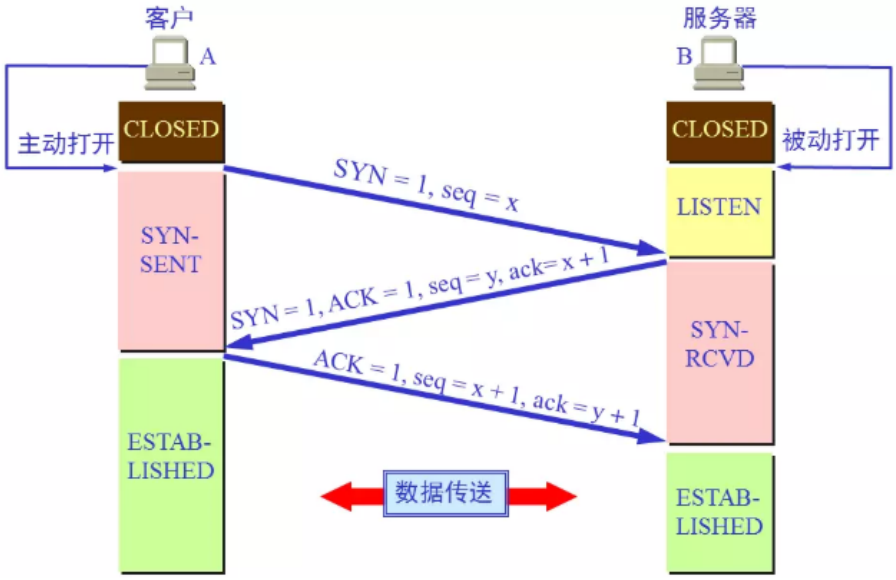
TCP

- 定义：Transmission Control Protocol，即传输控制协议，是一种传输层通信协议

> *基于TCP的应用层协议有FTP、Telnet、SMTP、HTTP、POP3与DNS。*

- 特点：面向连接、面向字节流、全双工通信、可靠
    - **面向连接**：指的是要使用TCP传输数据，必须先建立TCP连接，传输完成后释放连接，就像打电话一样必须先拨号建立一条连接，打完后挂机释放连接。
    - **全双工通信**：即一旦建立了TCP连接，通信双方可以在任何时候都能发送数据。
    - **可靠的**：指的是通过TCP连接传送的数据，无差错，不丢失，不重复，并且按序到达。
    - **面向字节流**：流，指的是流入到进程或从进程流出的字符序列。简单来说，虽然有时候要传输的数据流太大，TCP报文长度有限制，不能一次传输完，要把它分为好几个数据块，但是由于可靠性保证，接收方可以按顺序接收数据块然后重新组成分块之前的数据流，所以TCP看起来就像直接互相传输字节流一样，面向字节流。
- TCP建立连接 必须进行**三次握手**：若A要与B进行连接，则必须
    - 第一次握手：建立连接。客户端发送连接请求报文段，将SYN位置为1，Sequence Number为x；然后，客户端进入SYN_SEND状态，等待服务器的确认。即A发送信息给B
    - 第二次握手：服务器收到客户端的SYN报文段，需要对这个SYN报文段进行确认。即B收到连接信息后向A返回确认信息
    - 第三次握手：客户端收到服务器的（SYN+ACK）报文段，并向服务器发送ACK报文段。即A收到确认信息后再次向B返回确认连接信息

> *此时，A告诉自己上层连接建立；B收到连接信息后告诉上层连接建立。*



这样就完成TCP三次握手 = 一条TCP连接建立完成 = 可以开始发送数据

> 1. *三次握手期间任何一次未收到对面回复都要重发。*
> 2. *最后一个确认报文段发送完毕以后，客户端和服务器端都进入ESTABLISHED状态。*

为什么TCP建立连接需要三次握手？

答：防止服务器端因为接收了**早已失效的连接请求报文**从而一直等待客户端请求，从而浪费资源

> - *“已失效的连接请求报文段”的产生在这样一种情况下：Client发出的第一个连接请求报文段并没有丢失，而是在某个网络结点长时间的滞留了，以致延误到连接释放以后的某个时间才到达server。*
> - *这是一个早已失效的报文段。但Server收到此失效的连接请求报文段后，就误认为是Client再次发出的一个新的连接请求。*
> - *于是就向Client发出确认报文段，同意建立连接。*
> - *假设不采用“三次握手”：只要Server发出确认，新的连接就建立了。*
> - *由于现在Client并没有发出建立连接的请求，因此不会向Server发送数据。*
> - *但Server却以为新的运输连接已经建立，并一直等待Client发来数据。>- 这样，Server的资源就白白浪费掉了。*

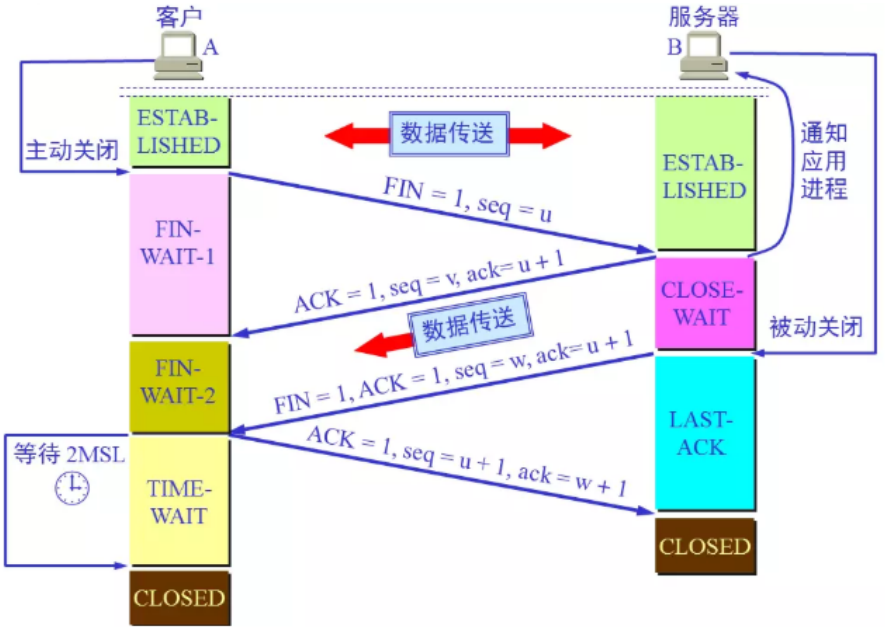
以上答案只是表象，没有说到本质上去

> *那么本质是 因为tcp是全双工，为保证传输的可靠性，需要给每次传输的数据段添加序号，那么初始的序列号就是tcp三次握手真正的意义所在，而为了确保交换双方的初始序号，最少需要三次才行*

采用“三次握手”的办法可以防止上述现象发生：

- Client不会向Server的确认发出确认
- Server由于收不到确认，就知道Client并没有要求建立连接
- 所以Server不会等待Client发送数据，资源就没有被浪费
- TCP释放连接 TCP释放连接需要**四次挥手**过程，现在假设A主动释放连接：（数据传输结束后，通信的双方都可释放连接）
    - 第一次挥手：A发送释放信息到B；（发出去之后，A->B发送数据这条路径就断了）
    - 第二次挥手：B收到A的释放信息之后，回复确认释放的信息：我同意你的释放连接请求
    - 第三次挥手：B发送“请求释放连接“信息给A
    - 第四次挥手：A收到B发送的信息后向B发送确认释放信息：我同意你的释放连接请求

        > *B收到确认信息后就会正式关闭连接；A等待2MSL后依然没有收到回复，则证明B端已正常关闭，于是A关闭连接*



为什么TCP释放连接需要四次挥手？

为了保证双方都能通知对方“需要释放连接”，即在释放连接后都无法接收或发送消息给对方

- 需要明确的是：TCP是全双工模式，这意味着是双向都可以发送、接收的
- 释放连接的定义是：双方都无法接收或发送消息给对方，是双向的
- 当主机1发出“释放连接请求”（FIN报文段）时，只是表示主机1已经没有数据要发送 / 数据已经全部发送完毕；

> *但是，这个时候主机1还是可以接受来自主机2的数据。*

- 当主机2返回“确认释放连接”信息（ACK报文段）时，表示它已经知道主机1没有数据发送了 但此时主机2还是可以发送数据给主机1
- 当主机2也发送了FIN报文段时，即告诉主机1我也没有数据要发送了 此时，主机1和2已经无法进行通信：主机1无法发送数据给主机2，主机2也无法发送数据给主机1，此时，TCP的连接才算释放

三次握手&四次挥手面试题总结

1. 三次握手是什么或者流程？四次握手呢？答案前面分析就是。
2. 为什么建立连接是三次握手，而关闭连接却是四次挥手呢？

> *这是因为服务端在LISTEN状态下，收到建立连接请求的SYN报文后，把ACK和SYN放在一个报文里发送给客户端。而关闭连接时，当收到对方的FIN报文时，仅仅表示对方不再发送数据了但是还能接收数据，己方也未必全部数据都发送给对方了，所以己方可以立即close，也可以发送一些数据给对方后，再发送FIN报文给对方来表示同意现在关闭连接，因此，己方ACK和FIN一般都会分开发送。*

UPD

- 定义：User Datagram Protocol，即用户数据报协议，是一种传输层通信协议。

  > *基于UDP的应用层协议有TFTP、SNMP与DNS。*

- 特点：无连接的、不可靠的、面向报文、没有拥塞控制
  - **无连接的**：和TCP要建立连接不同，UDP传输数据不需要建立连接，就像写信，在信封写上收信人名称、地址就可以交给邮局发送了，至于能不能送到，就要看邮局的送信能力和送信过程的困难程度了。
  - **不可靠的**：因为UDP发出去的数据包发出去就不管了，不管它会不会到达，所以很可能会出现丢包现象，使传输的数据出错。
  - **面向报文**：数据报文，就相当于一个数据包，应用层交给UDP多大的数据包，UDP就照样发送，不会像TCP那样拆分。
  - **没有拥塞控制**：拥塞，是指到达通信子网中某一部分的分组数量过多，使得该部分网络来不及处理，以致引起这部分乃至整个网络性能下降的现象，严重时甚至会导致网络通信业务陷入停顿，即出现死锁现象，就像交通堵塞一样。TCP建立连接后如果发送的数据因为信道质量的原因不能到达目的地，它会不断重发，有可能导致越来越塞，所以需要一个复杂的原理来控制拥塞。而UDP就没有这个烦恼，发出去就不管了。
- 应用场景 很多的实时应用（如IP电话、实时视频会议、某些多人同时在线游戏等）要求源主机以很定的速率发送数据，并且允许在网络发生拥塞时候丢失一些数据，但是要求不能有太大的延时，UDP就刚好适合这种要求。

Java中对于网络提供的几个关键类

针对不同的网络通信层次，Java给我们提供的网络功能有四大类：

- InetAddress： 用于标识网络上的硬件资源
- URL： 统一资源定位符，通过URL可以直接读取或者写入网络上的数据
- Socket和ServerSocket： 使用TCP协议实现网络通信的Socket相关的类
- Datagram： 使用UDP协议，将数据保存在数据报中，通过网络进行通信

Socket

什么是**Socket**？

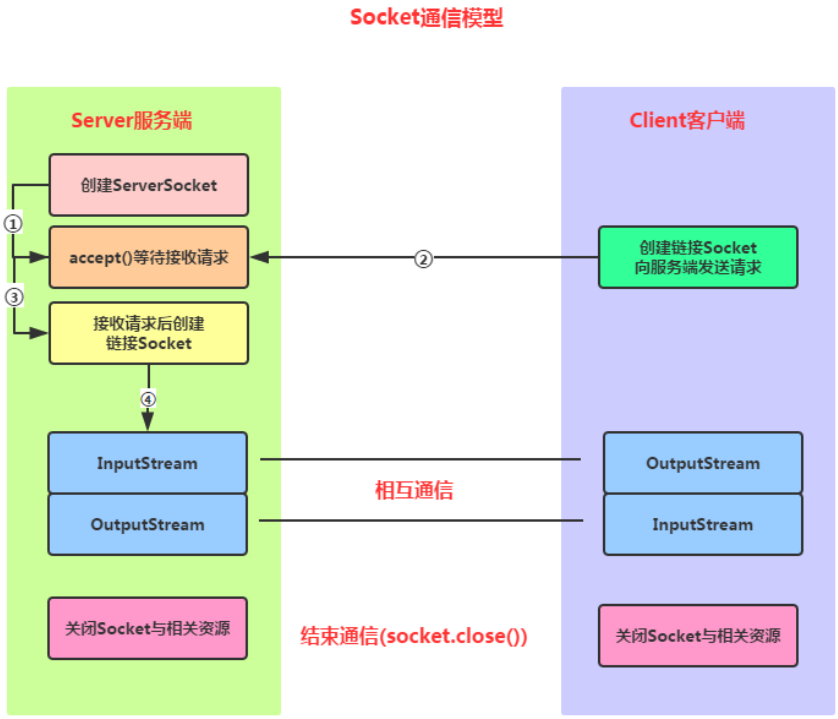
- 即套接字，**是一个对 TCP / IP协议进行封装 的编程调用接口（API）** 用来描述IP地址和端口，是通信链的句柄，应用程序可以通过Socket向网络发送请求或者 应答网络请求！Socket是支持TCP/IP协议的网络通信的基本操作单元，是对网络通信过程 中端点的抽象表示，包含了进行网络通信所必须的五种信息

  a. 连接所使用的的协议

b. 本地主机的IP地址
c. 本地远程的协议端口
d. 远程主机的IP地址
e. 远地进程的协议端口

> *a. 即通过`Socket`，我们才能在Andorid平台上通过`TCP/IP`协议进行开发*
> *b. `Socket`不是一种协议，而是一个编程调用接口（`API`），属于传输层（主要解决数据如何在网络中传输）*
> *c. 成对出现，一对套接字*

Socket通信模型



Socket通信步骤

- Step 1：创建ServerSocket和Socket
- Step 2：打开连接到的Socket的输入/输出流
- Step 3：按照协议对Socket进行读/写操作
- Step 4：关闭输入输出流，以及Socket

Socket服务端的编写

- Step 1：创建ServerSocket对象，绑定监听的端口
- Step 2：调用accept()方法监听客户端的请求
- Step 3：连接建立后，通过输入流读取客户端发送的请求信息
- Step 4：通过输出流向客户端发送响应信息
- Step 5：关闭相关资源

```
 public static void main(String[] args) throws IOException
{
        //1.创建一个服务器端Socket，即ServerSocket，指定绑定的端
口，并监听此端口
        ServerSocket serverSocket = new
ServerSocket(12345);
        InetAddress address = InetAddress.getLocalHost();
        String ip = address.getHostAddress();
        Socket socket = null;
        //2.调用accept()等待客户端连接
        System.out.println("~~~服务端已就绪，等待客户端接入~，
服务端ip地址: " + ip);
        socket = serverSocket.accept();
        //3.连接后获取输入流，读取客户端信息
        InputStream is=null;
        InputStreamReader isr=null;
        BufferedReader br=null;
        OutputStream os=null;
        PrintWriter pw=null;
        is = socket.getInputStream();     //获取输入流
        isr = new InputStreamReader(is,"UTF-8");
        br = new BufferedReader(isr);
        String info = null;
        while((info=br.readLine())!=null){//循环读取客户端的
信息
            System.out.println("客户端发送过来的信息" +
info);
        }
        socket.shutdownInput();//关闭输入流
        socket.close();
    }
```

Socket客户端的编写

- Step 1：创建Socket对象，指明需要链接的服务器的地址和端号
- Step 2：链接建立后，通过输出流向服务器发送请求信息
- Step 3：通过输出流获取服务器响应的信息
- Step 4：关闭相关资源

```
public static void main(String ... args) throws Exception{
        //1.创建客户端Socket，指定服务器地址和端口
        Socket socket = new Socket("127.0.0.1", 12345);
        //2.获取输出流，向服务器端发送信息
        OutputStream os = socket.getOutputStream();//字节输
出流
        PrintWriter pw = new PrintWriter(os);//将输出流包装
为打印流
        //获取客户端的IP地址
        InetAddress address = InetAddress.getLocalHost();
        String ip = address.getHostAddress();
        pw.write("客户端: ~" + ip + "~ 接入服务器! ! ");
```

```
            pw.flush();
            socket.shutdownOutput();//关闭输出流
            socket.close();
        }
```