钱进培训是哈法地区资深工程师组成的培训机构，通过各位老师的现身说法，帮助各位学员迅速掌握实战知识，为求职打下坚实的基础。电子邮件：jin.qian.canada@gmail.com钱老师报名、答疑微信号：qianjincanada，或扫描以下二维码添加：

钱老师

加拿大

扫一扫上面的二维码图案，加我微信

钱进老师

钱进老师

Java高级班（JavaEE方向）
讲义4

钱进培训立足Halifax地区，面向在校生提供软件开发技能培训和课程辅导。

本课是钱进培训组织的软件开发系列课程的主打课程，主要征针对已经学习过CSCI1100（JAVA1）的同学，目标是通过大约10次课的学习，掌握JavaEE开发必备的技能，对现代软件企业的开发方式、常用类库、方法论等在校很难学到的知识点进行全面讲解，做到心中有数，提前具备求职的基本技术素质。

I/O 问题是任何编程语言都无法回避的问题，可以说 I/O 问题是整个人机交互的核心问题，因为 I/O 是机器获取和交换信息的主要渠道。在当今这个数据大爆炸时代，I/O 问题尤其突出，很容易成为一个性能瓶颈。

Java 的 I/O 操作类在包 java.io 下，大概有将近 80 个类，但是这些类大概可以分成四组，分别是：

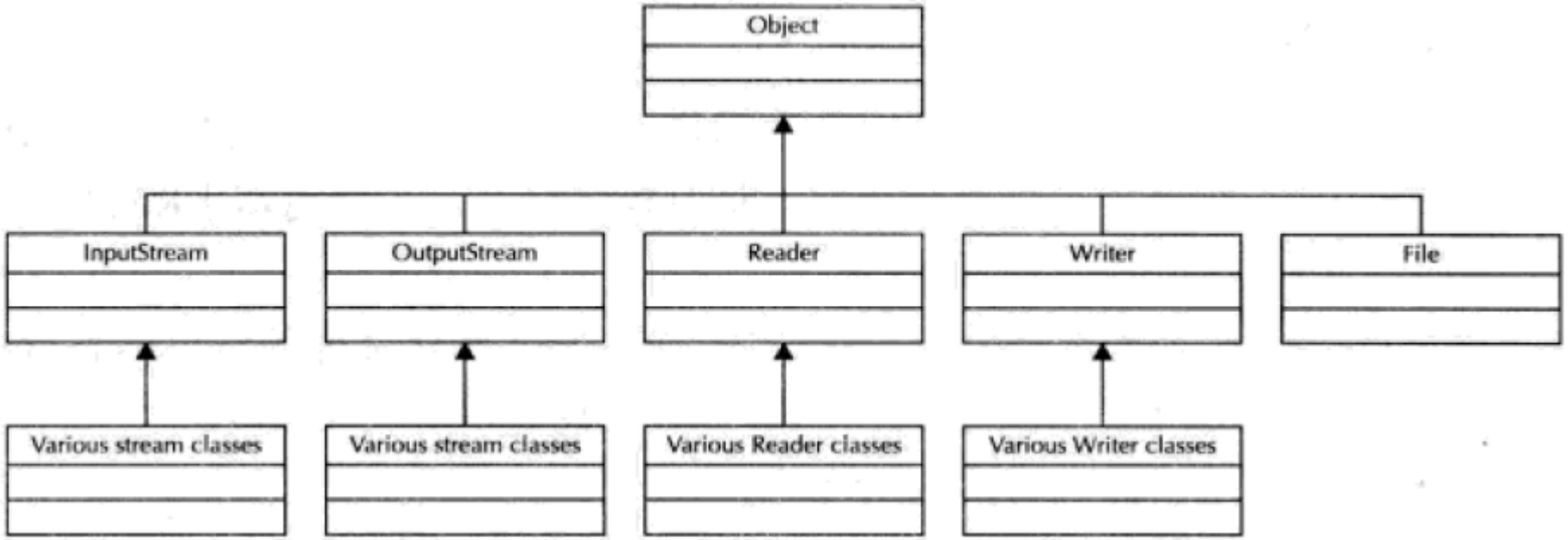
基于字节操作的 I/O 接口：InputStream 和 OutputStream

基于字符操作的 I/O 接口：Writer 和 Reader

基于磁盘操作的 I/O 接口：File

基于网络操作的 I/O 接口：Socket

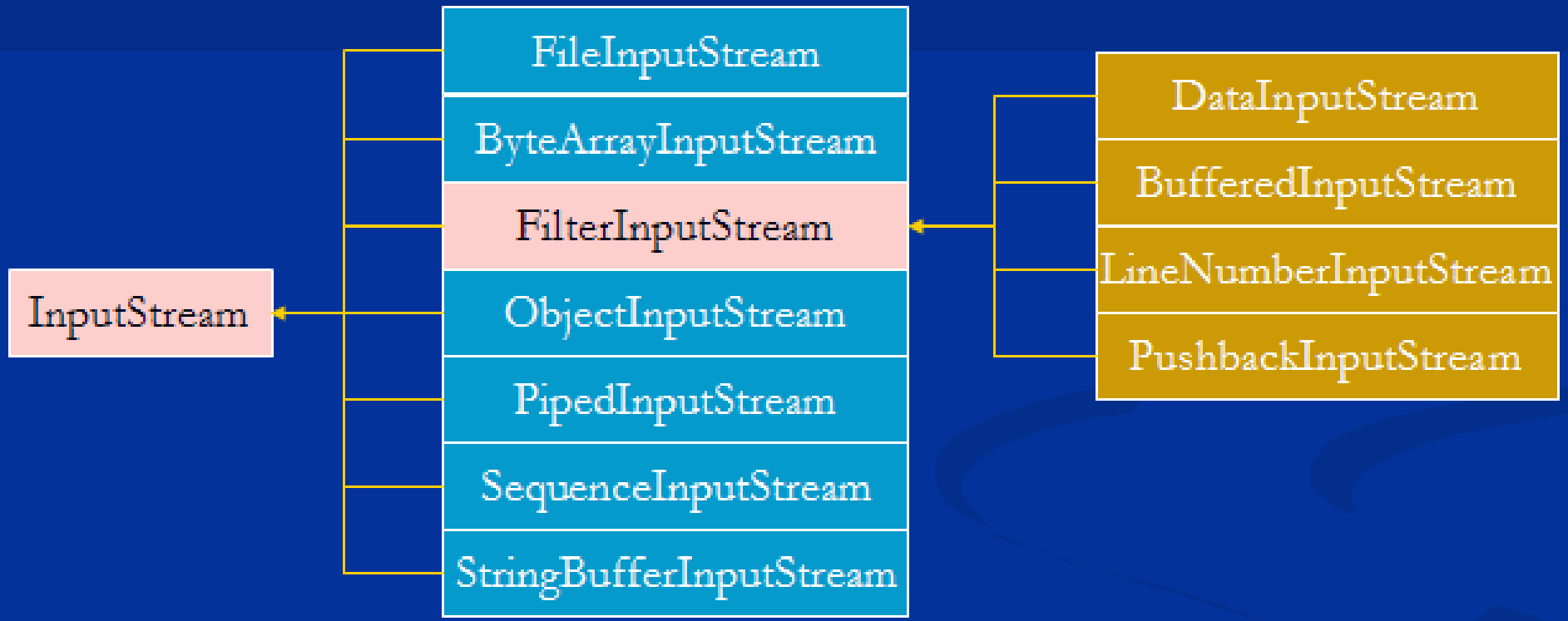
前两组主要是根据传输数据的数据格式，后两组主要是根据传输数据的方式，虽然 Socket 类并不在 java.io 包下



Java中所有的类都继承自Object类，所以各种IO类也不例外。InputStream和OutputStream类操作字节数据。Reader和Writer类工作在字符是上。File类为文件提供接口。

我们很有必要了解IO类的演变历史，如果缺乏历史的眼光，那么我们对什么时候该使用那些类，以及什么时候不该使用那些类而感到迷惑。
JDK1.0的时候，所有与输入相关的类都继承于InputStream，所有与输出相关的类都继承于OutputStream。
JDK1.1的时候，增加了面向字符的IO类，包括Reader和Writer。

java.io包中 InputStream的类层次



字节流

字节流，顾名思义，以字节为单位进行IO操作。字节流最常用的方法是read和write方法。

read方法是从流中读取字节并递增文件指针到一下个位置。在字节流的读取过程中，我们要明白一个重要的概念：文件指针。文件指针指示了流中的当前位置。当文件指针到达文件的末尾时候，读取操作返回-1给调用者。read方法不带任何参数，每次只能读取一个字节。read方法的返回值是读取的字节，并转化为int类型返回。如下所示：

```
FileInputStream reader =new FileInputStream(“filename”);//以字节流格式打开文件

int result =reader.read();
```

read方法还有其他的形式：

int read(byte[] b)，读出的数据被存储在字节数组中，返回读取的字节数，如果提前到达文件的末尾，返回值可能小于数组的长度。

int read(byte[] b, int off, int len)，第一个参数指定数据被存储的字节数组，第二个参数off指定读取的第一个字节将存储在字节数组的偏移量，第三个参数len指定要读取的字节数。

上面三种形式的读取都是直接从文件流里读取的，每次读取只能是一个一个字节的读取。需要说明的是每次读取单个字节或512个字节，需要的io数量都是相同的。为了提高效率，人们设计出了缓存的读取方式：

```
FileInputStream reader =new FileInputStream("filename");
BufferedInputStream bs = new BufferedInputStream(reader);
bs.read();
```

虽然上面仍然读取的是一个字节数据，但是因为缓存的存在，效率已经大大提高了。底层的实现是这样的：BufferedInputStream的实现中有一个用于存储数据的内部缓冲区数组：protected volatile byte[] buf。这个缓冲区数组的作用在于对源进行数据块访问，而不是一字节一字节的访问，也就是进行一次I/O将一块数据存到缓冲区中，再从缓冲区中read，当缓冲区为空时再重新读新的数据块。

注意：

一个是操作数据的方式是可以组合使用的，如这样组合使用

```
OutputStream out = new BufferedOutputStream(new ObjectOutputStream(new FileOutputStream("fileName"))；
```

还有一点是流最终写到什么地方必须要指定，要么是写到磁盘要么是写到网络中

不管是磁盘还是网络传输，最小的存储单元都是字节，而不是字符，所以 I/O 操作的都是字节而不是字符，但是为啥有操作字符的 I/O 接口呢？这是因为我们的程序中通常操作的数据都是以字符形式，为了操作方便当然要提供一个直接写字符的 I/O 接口，如此而已。我们知道字符到字节必须要经过编码转换，而这个编码又非常耗时，而且还会经常出现乱码问题，所以 I/O 的编码问题经常是让人头疼的问题。

字符流

字符流是操作字符文件的。一般情况下我们的使用方式是：

```
FileReader reader =new FileReader("filename");//以字符格式打开文件
BufferedReader buffer = new BufferedReader(reader);//建立缓存
buffer.readLine();//读取缓存中的一行
```

```
/* 读入TXT文件 */
        String pathname = "D:\\twitter\\13_9_6\\dataset\\en\\input.txt"; // 绝对路径或相对路径都可以，这里是绝对路径，写入文件时演示相对路径
        File filename = new File(pathname); // 要读取以上路径的input。txt文件
        InputStreamReader reader = new InputStreamReader(
            new FileInputStream(filename)); // 建立一个输入流对象reader
        BufferedReader br = new BufferedReader(reader); // 建立一个对象，它把文件内容转成计算机能读懂的语言
        String line = "";
        line = br.readLine();
        while (line != null) {
           line = br.readLine(); // 一次读入一行数据
        }

        /* 写入Txt文件 */
        File writename = new File(".\\result\\en\\output.txt"); // 相对路径，如果没有则要建立一个新的output。txt文件
        writename.createNewFile(); // 创建新文件
        BufferedWriter out = new BufferedWriter(new FileWriter(writename));
        out.write("我会写入文件啦\r\n"); // \r\n即为换行
        out.flush(); // 把缓存区内容压入文件
        out.close(); // 最后记得关闭文件
```

协议简介

协议相当于相互通信的程序间达成的一种约定，它规定了分组报文的结构、交换方式、包含的意义以及怎样对报文所包含的信息进行解析。

TCP/IP 协议族有 IP 协议、TCP 协议和 UDP 协议。

TCP 协议和 UDP 协议使用的地址叫做端口号，用来区分同一主机上的不同应用程序。TCP 协议和 UDP 协议也叫端到端传输协议，因为他们将数据从一个应用程序传输到另一个应用程序，而 IP 协议只是将数据从一个主机传输到另一个主机。

在 TCP/IP 协议中，有两部分信息用来确定一个指定的程序：互联网地址和端口号：其中互联网地址由 IP 协议使用，而附加的端口地址信息则由传输协议（TCP 或 UDP 协议）对其进行解析。

现在 TCP/IP 协议族中的主要 socket 类型为流套接字（使用 TCP 协议）和数据报套接字（使用 UDP 协议），其中通过数据报套接字，应用程序一次只能发送最长 65507 个字节长度的信息。

一个 TCP/IP 套接字由一个互联网地址，一个端对端协议（TCP 协议或 UDP 协议）以及一个端口号唯一确定。

TCP与UDP区别：

TCP特点：

1、TCP是面向连接的协议，通过三次握手建立连接，通讯完成时要拆除连接，由于TCP是面向连接协议，所以只能用于点对点的通讯。而且建立连接也需要消耗时间和开销。

2、TCP传输数据无大小限制，进行大数据传输。

3、TCP是一个可靠的协议，它能保证接收方能够完整正确地接收到发送方发送的全部数据。

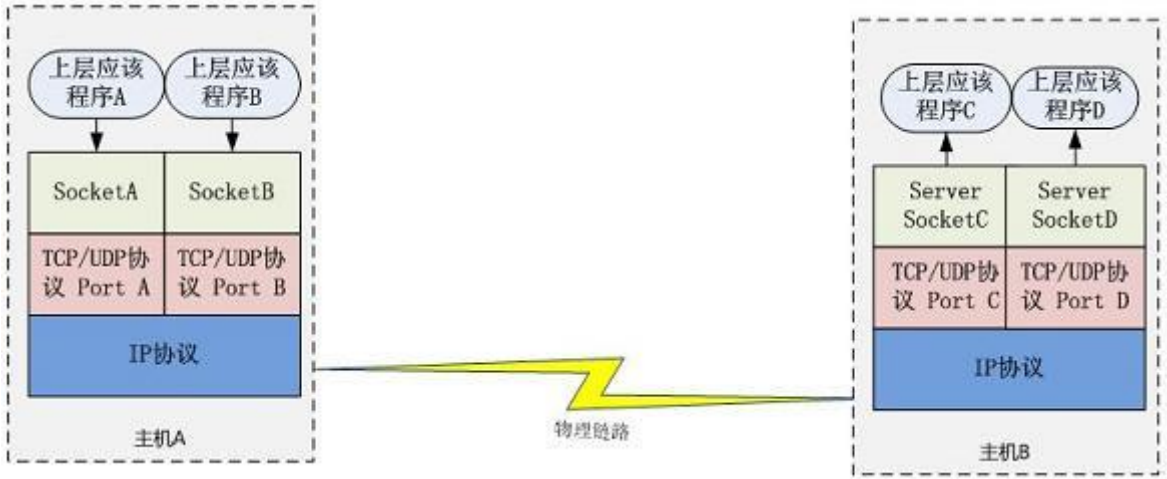
UDP特点：

1、UDP是面向无连接的通讯协议，UDP数据包括目的端口号和源端口号信息，由于通讯不需要连接，所以可以实现广播发送。

2、UDP传输数据时有大小限制，每个被传输的数据报必须限定在64KB之内。

3、UDP是一个不可靠的协议，发送方所发送的数据报并不一定以相同的次序到达接收方。

主机 A 的应用程序要能和主机 B 的应用程序通信，必须通过 Socket 建立连接，而建立 Socket 连接必须需要底层 TCP/IP 协议来建立 TCP 连接。建立 TCP 连接需要底层 IP 协议来寻址网络中的主机。我们知道网络层使用的 IP 协议可以帮助我们根据 IP 地址来找到目标主机，但是一台主机上可能运行着多个应用程序，如何才能与指定的应用程序通信就要通过 TCP 或 UPD 的地址也就是端口号来指定。这样就可以通过一个 Socket 实例唯一代表一个主机上的一个应用程序的通信链路了。



Socket通常也称作"套接字"，用于描述IP地址和端口，是一个通信链的句柄。网络上的两个程序通过一个双向的通讯连接实现数据的交换，这个双向链路的一端称为一个Socket，一个Socket由一个IP地址和一个端口号唯一确定。应用程序通常通过"套接字"向网络发出请求或者应答网络请求。 Socket是TCP/IP协议的一个十分流行的编程界面，但是，Socket所支持的协议种类也不光TCP/IP一种，因此两者之间是没有必然联系的。在Java环境下，Socket编程主要是指基于TCP/IP协议的网络编程。

Socket通讯过程：服务端监听某个端口是否有连接请求，客户端向服务端发送连接请求，服务端收到连接请求向客户端发出接收消息，这样一个连接就建立起来了。客户端和服务端都可以相互发送消息与对方进行通讯。

Socket的基本工作过程包含以下四个步骤：

1、创建Socket；

2、打开连接到Socket的输入输出流；

3、按照一定的协议对Socket进行读写操作；

4、关闭Socket。

Socket可以说是一种针对网络的抽象，应用通过它可以来针对网络读写数据。就像通过一个文件的file handler就可以都写数据到存储设备上一样。根据TCP协议和UDP协议的不同，在网络编程方面就有面向两个协议的不同socket，一个是面向字节流的一个是面向报文的。

对socket的本身组成倒是比较好理解。既然是应用通过socket通信，肯定就有一个服务器端和一个客户端。所以它必然就包含有一个对应的IP地址。另外，在这个地址上server要提供一系列的服务，于是就需要有一系列对应的窗口来提供服务。所以就有一个对应的端口号(Port)。端口号是一个16位的二进制数字，那么范围就是从（0-65535）。IP地址加端口号基本上就构成了socket。

在java.net包下有两个类：Socket和ServerSocket。ServerSocket用于服务器端，Socket是建立网络连接时使用的。在连接成功时，应用程序两端都会产生一个Socket实例，操作这个实例，完成所需的会话。对于一个网络连接来说，套接字是平等的，并没有差别，不因为在服务器端或在客户端而产生不同级别。

TCP 协议提供面向连接的服务，通过它建立的是可靠地连接。Java 为 TCP 协议提供了两个类：Socke 类和 ServerSocket 类。一个 Socket 实例代表了 TCP 连接的一个客户端，而一个 ServerSocket 实例代表了 TCP 连接的一个服务器端，一般在 TCP Socket 编程中，客户端有多个，而服务器端只有一个，客户端 TCP 向服务器端 TCP 发送连接请求，服务器端的 ServerSocket 实例则监听来自客户端的 TCP 连接请求，并为每个请求创建新的 Socket 实例，由于服务端在调用 accept（）等待客户端的连接请求时会阻塞，直到收到客户端发送的连接请求才会继续往下执行代码，因此要为每个 Socket 连接开启一个线程。服务器端要同时处理 ServerSocket 实例和 Socket 实例，而客户端只需要使用 Socket 实例。另外，每个 Socket 实例会关联一个 InputStream 和 OutputStream 对象，我们通过将字节写入套接字的 OutputStream 来发送数据，并通过从 InputStream 来接收数据。

客户端向服务器端发送连接请求后，就被动地等待服务器的响应。典型的 TCP 客户端要经过下面三步操作：

创建一个 Socket 实例：构造函数向指定的远程主机和端口建立一个 TCP 连接；

通过套接字的 I/O 流与服务端通信；

使用 Socket 类的 close 方法关闭连接。

服务端的工作是建立一个通信终端，并被动地等待客户端的连接。

典型的 TCP 服务端执行如下两步操作：

创建一个 ServerSocket 实例并指定本地端口，用来监听客户端在该端口发送的 TCP 连接请求；

重复执行：

调用 ServerSocket 的 accept（）方法以获取客户端连接，并通过其返回值创建一个 Socket 实例；

为返回的 Socket 实例开启新的线程，并使用返回的 Socket 实例的 I/O 流与客户端通信； 通信完成后，使用 Socket 类的 close（）方法关闭该客户端的套接字连接。

```
public class ClientSocket {
  public static void main(String args[]) {
        String host = "127.0.0.1";
        int port = 8919;
        try {
          Socket client = new Socket(host, port);
          Writer writer = new OutputStreamWriter(client.getOutputStream());
          writer.write("Hello From Client");
          writer.flush();
          writer.close();
          client.close();
        } catch (IOException e) {
          e.printStackTrace();
        }
    }

}
```

```
public class ServerClient {
  public static void main(String[] args) {
      int port = 8919;
      try {
         ServerSocket server = new ServerSocket(port);
            Socket socket = server.accept();
         Reader reader = new InputStreamReader(socket.getInputStream());
         char chars[] = new char[1024];
         int len;
         StringBuilder builder = new StringBuilder();
         while ((len=reader.read(chars)) != -1) {
           builder.append(new String(chars, 0, len));
         }
         System.out.println("Receive from client message=: " + builder);
         reader.close();
         socket.close();
         server.close();
      } catch (Exception e) {
         e.printStackTrace();
      }
  }
}
```

```
public class Client1 {
    public static void main(String[] args) throws IOException {
        //客户端请求与本机在20006端口建立TCP连接
        Socket client = new Socket("127.0.0.1", 20006);
        client.setSoTimeout(10000);
        //获取键盘输入
        BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
        //获取Socket的输出流，用来发送数据到服务端
        PrintStream out = new PrintStream(client.getOutputStream());
        //获取Socket的输入流，用来接收从服务端发送过来的数据
        BufferedReader buf =  new BufferedReader(new
InputStreamReader(client.getInputStream()));
```

```
boolean flag = true;
      while(flag){
         System.out.print("输入信息：");
         String str = input.readLine();
         //发送数据到服务端
         out.println(str);
         if("bye".equals(str)){
            flag = false;
         }else{
            try{
               //从服务器端接收数据有个时间限制（系统自设，也可以自己设置），超过了这个时
间，便会抛出该异常
               String echo = buf.readLine();
               System.out.println(echo);
            }catch(SocketTimeoutException e){
               System.out.println("Time out, No response");
            }
         }
      }
      input.close();
```

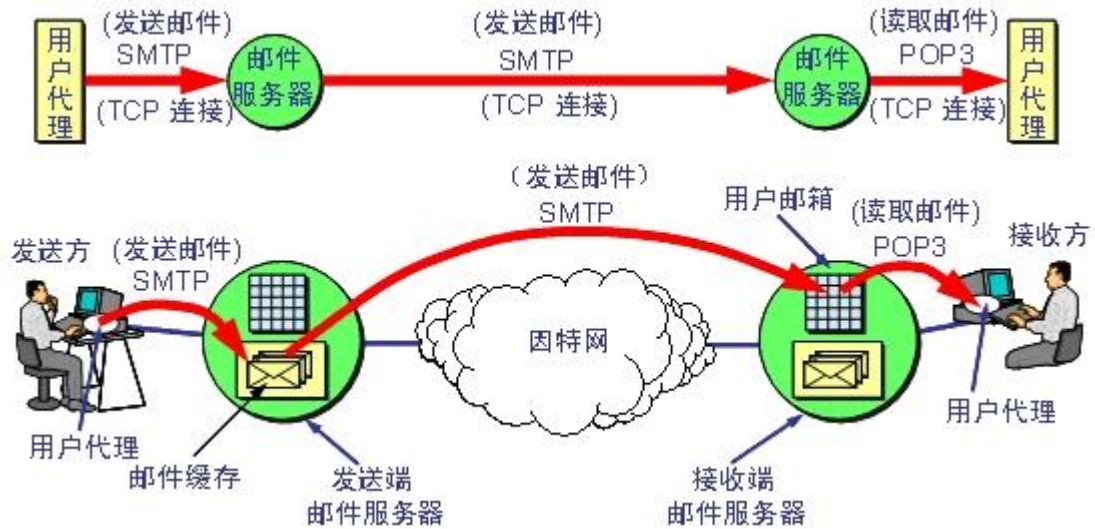
```
public class ServerThread implements Runnable {

    private Socket client = null;
    public ServerThread(Socket client){
        this.client = client;
    }

    @Override
    public void run() {
        try{
            //获取Socket的输出流，用来向客户端发送数据
            PrintStream out = new PrintStream(client.getOutputStream());
            //获取Socket的输入流，用来接收从客户端发送过来的数据
            BufferedReader buf = new BufferedReader(new
InputStreamReader(client.getInputStream()));
            boolean flag =true;
```

```
while(flag){
            //接收从客户端发送过来的数据
            String str =  buf.readLine();
            if(str == null || "".equals(str)){
               flag = false;
            }else{
               if("bye".equals(str)){
                  flag = false;
               }else{
                  //将接收到的字符串前面加上echo，发送到对应的客户端
                  out.println("echo:" + str);
               }
            }
         }
         out.close();
         client.close();
      }catch(Exception e){
         e.printStackTrace();
      }
   }

}
```

```
public class Server1 {
    public static void main(String[] args) throws Exception{
        //服务端在20006端口监听客户端请求的TCP连接
        ServerSocket server = new ServerSocket(20006);
        Socket client = null;
        boolean f = true;
        while(f){
            //等待客户端的连接，如果没有获取连接
            client = server.accept();
            System.out.println("与客户端连接成功！");
            //为每个客户端连接开启一个线程
            new Thread(new ServerThread(client)).start();
        }
        server.close();
    }
}
```

```
public class ClientSocket {
  public static void main(String args[]) {
        String host = "127.0.0.1";
        int port = 8919;
        try {
          Socket client = new Socket(host, port);
          Writer writer = new OutputStreamWriter(client.getOutputStream());
          writer.write("Hello From Client");
          writer.flush();
          writer.close();
          client.close();
        } catch (IOException e) {
          e.printStackTrace();
        }
    }

}
```

SMTP，即简单邮件传送协议，所对应RFC文档为RFC821。同http等多数应用层协议一样，它工作在C/S模式下，用来实现因特网上的邮件传送。SMTP在整个电子邮件通信中所处的位置如图 1所示。



一个具体的SMTP通信（如发送端邮件服务器与接收端服务器的通信）的过程如下。

1)　发送端邮件服务器（以下简称客户端）与接收端邮件服务器（以下简称服务器）的25号端口建立TCP连接。

2)　客户端向服务器发送各种命令，来请求各种服务（如认证、指定发送人和接收人）。

3)　服务器解析用户的命令，做出相应动作并返回给客户端一个响应。

4)　2)和3)交替进行，直到所有邮件都发送完或两者的连接被意外中断。

从这个过程看出，命令和响应是SMTP协议的重点，下面将予以重点讲述。

SMTP的命令不多（14个），它的一般形式是：COMMAND [Parameter] <CRLF>。其中COMMAND是ASCII形式的命令名，Parameter是相应的命令参数，<CRLF>是回车换行符(0DH, 0AH)。

SMTP的响应也不复杂，它的一般形式是：XXX Readable Illustration。XXX是三位十进制数；Readable Illustration是可读的解释说明，用来表明命令是否成功等。XXX具有如下的规律：以2开头的表示成功，以4和5开头的表示失败，以3开头的表示未完成（进行中）。

SMTP命令不区分大小写，但参数区分大小写，有关这方面的详细说明请参考RFC821。常用的命令如下。

HELO <domain> <CRLF>。向服务器标识用户身份发送者能欺骗，说谎，但一般情况下服务器都能检测到。

MAIL FROM: <reverse-path> <CRLF>。<reverse-path>为发送者地址，此命令用来初始化邮件传输，即用来对所有的状态和缓冲区进行初始化。

RCPT TO：<forward-path> <CRLF>。 <forward-path>用来标志邮件接收者的地址，常用在MAIL FROM后，可以有多个RCPT TO。

DATA <CRLF>。将之后的数据作为数据发送，以<CRLF>.<CRLF>标志数据的结尾。

REST <CRLF>。重置会话，当前传输被取消。

NOOP <CRLF>。要求服务器返回OK应答，一般用作测试。

QUIT <CRLF>。结束会话。

VRFY <string> <CRLF>。验证指定的邮箱是否存在，由于安全方面的原因，服务器大多禁止此命令。

EXPN <string> <CRLF>。验证给定的邮箱列表是否存在，由于安全方面的原因，服务器大多禁止此命令。

HELP <CRLF>。查询服务器支持什么命令

常用的响应如下所示，数字后的说明是从英文译过来的。更详细的说明请参考RFC821。

501参数格式错误

502命令不可实现

503错误的命令序列

504命令参数不可实现

211系统状态或系统帮助响应

214帮助信息

220＜domain＞服务就绪

221＜domain＞服务关闭

421＜domain＞服务未就绪，关闭传输信道

250要求的邮件操作完成

251用户非本地，将转发向＜forward-path＞

450要求的邮件操作未完成，邮箱不可用

550要求的邮件操作未完成，邮箱不可用

451放弃要求的操作；处理过程中出错

551用户非本地，请尝试＜forward-path＞

452系统存储不足，要求的操作未执行

552过量的存储分配，要求的操作未执行

553邮箱名不可用，要求的操作未执行

354开始邮件输入，以"."结束

554操作失败

```
SSLSocket socket = (SSLSocket) ((SSLSocketFactory) SSLSocketFactory.getDefault())
.createSocket(“smtp.gmail.com”, 465);
OutputStream socketOut = socket.getOutputStream();
socketOut.write(("HELO " + localhost + "\r\n").getBytes());
socketOut.write(("AUTH LOGIN " + userName + "\r\n").getBytes());
socketOut.write((passWord + "\r\n").getBytes());
socketOut.write(("MAIL FROM:<" + email_from + ">" + "\r\n").getBytes());
socketOut.write(("RCPT TO:<" + email_to + ">" + "\r\n").getBytes());
socketOut.write(("DATA" + "\r\n").getBytes());
socketOut.write(("SUBJECT:" + email_subject + "\r\n").getBytes());
socketOut.write((email_body + "\r\n").getBytes());
socketOut.write(("." + "\r\n").getBytes());
socketOut.write(("QUIT" + "\r\n").getBytes());
```

CCNA全称是Cisco Certified Network Associate，翻译过来就是思科认证网络工程师，而Cisco(思科)公司是全球最大的网络设备公司，CCNA是Cisco认证证书体系中的初级技术证书。

获得CCNA认证标志着具备安装、配置、运行中型路由和交换网络，并进行故障排除的能力。成为CCNA思科认证网络工程师拥有通过广域网与远程站点建立连接，消除基本的安全威胁，了解无线网络接入的技能，并且熟悉和使用IP、EIGRP、串行线路接口协议、帧中继、RIPv2、VLAN、以太网和访问控制列表（ACL）等。

《CCNA学习指南中文》

第一章 第二章

钱进培训是哈法地区资深工程师组成的培训机构，通过各位老师的现身说法，帮助各位学员迅速掌握实战知识，为求职打下坚实的基础。电子邮件：jin.qian.canada@gmail.com钱老师报名、答疑微信号：qianjincanada，或扫描以下二维码添加：

钱老师

加拿大

扫一扫上面的二维码图案，加我微信

钱进老师

钱进老师