# 网络编程

CS架构：

客户端和服务器端

BS架构：

浏览器和服务器

浏览器其实也是一个客户端，只不过是一个通用的客户端

所以本质上都是CS架构

Java程序可以非常方便的访问互联网上的HTTP服务、FTP服务等，并可以直接取得互联网上的远程资源，还可以向远程资源发送GET、POST请求。

## 15.1 网络编程的基础知识

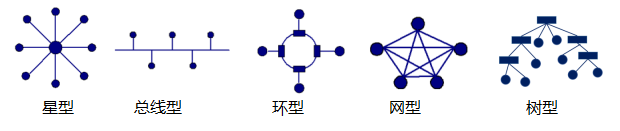
### 15.1.1 网络的概念和分类

* 什么是计算机网络？

就是把分布在不同地理区域的计算机与专门的外部设备用通信线路互连成一个规模大、功能强的网络系统，从而使众多的计算机可以方便地互相传递信息、共享硬件、软件、数据信息等资源。

生活在今天的我们，没有网络那是相当可怕的，我们每天几乎都在用所谓的云计算、云服务、云备份、搜索引擎检索信息、即时通信、在线支付......

* 计算机网络分类
  + 按照网络的传输介质划分：双绞线网、同轴电缆网、光纤网、卫星网等。还有无线传输的，例如Wife的传输介质是红外线，蓝牙的传输介质是无线电波，都是电磁波的一种。
  + 按照网络的拓扑结构来划分，可以分为星型网络、总线网络、环线网络、树型网络、网状网络、混合型网络等。



* + 按照规模来分类
    - 局域网（Local Area Network，LAN）

是指在某一区域内由多台计算机互联成的计算机组。**一般是方圆几千米以内**。局域网可以实现文件管理、应用软件共享、打印机共享、工作组内的日程安排、电子邮件和传真通信服务等功能。局域网是封闭型的，可以由办公室内的两台计算机组成，也可以由一个公司内的上千台计算机组成。

* + - 城域网(Metropolitan Area Network，MAN)

是**在一个城市范围内所建立的计算机通信网，属宽带局域网**。由于采用具有有源交换元件的局域网技术，网中传输时延较小，它的传输媒介主要采用光缆，传输速率在100兆比特/秒以上。MAN的一个重要用途是用作骨干网，通过它将位于同一城市内不同地点的主机、数据库，以及LAN等互相联接起来，这与WAN的作用有相似之处，但两者在实现方法与性能上有很大差别。

* 广域网（Wide Area Network， WAN）

**又称外网、公网，是连接不同地区局域网或城域网计算机通信的远程网。**通常跨接很大的物理范围，所覆盖的范围从几十公里到几千公里，它能连接多个地区、城市和国家，或横跨几个洲并能提供远距离通信，形成国际性的远程网络。

* 互联网、因特网、万维网呢？

凡是由能彼此通信的设备组成的网络就叫互联网，即使仅有两台机器（计算机、手机等），不论用何种技术使其彼此通信，都叫互联网，所以，互联网有广域网、城域网及局域网之分。国际标准的互联网写法是internet，字母i小写！

而因特网是互联网中的一种，它可不是仅有两台机器组成的网络，而是由上千万台设备组成的网络（该网络具备一定规模）。国际标准的因特网写法是Internet，字母I大写！

因特网是基于TCP/IP协议实现的，TCP/IP协议由很多协议组成，不同类型的协议又被放在不同的层，其中，位于应用层的协议就有很多，比如FTP、SMTP、HTTP。所以，因特网提供的服务一般包括有：www（万维网）服务、电子邮件服务（outlook）、远程登录（QQ）服务、文件传输（FTP）服务、网络电话等等。只要应用层使用的是HTTP协议，就称为万维网(World Wide Web)。

### 15.1.2 网络协议

网络通信三要素：

1. 地址 IP地址，定位到设备
2. 端口号 定位到设备上的进程（应用程序）
3. 网络协议 解析数据，控制传输

* 网络协议就是通信协议： 对速率、传输代码、代码结构、传输控制步骤、出错控制等制定标准。
* OSI参考模型

作用：统一网络体系结构；

网络通信必须有硬件和软件方面的支持，由于世界上大型计算机厂商推出各自不同的网络体系结构，影响了网络通信的统一性，因此国际标准化组织ISO于1978年提出了著名的OSI（Open System Interconnection）开放系统互连参考模型。

它把计算机网络分成物理层、数据链路层、网络层、传输层、会话层、表示层、应用层等七层。

1. 应用层：网络服务与最终用户的一个接口。协议有：HTTP、FTP、SMTP、DNS、TELNET、HTTPS、POP3等等。
2. 表示层：数据的表示、安全、压缩。格式有：JPEG、ASCll、DECOIC、加密格式等。
3. 会话层：建立、管理、终止会话。对应主机进程，指本地主机与远程主机正在进行的会话
4. 传输层：定义传输数据的协议端口号，以及流控和差错校验。协议有：TCP、UDP。
5. 网络层：进行逻辑地址寻址，实现不同网络之间的路径选择。协议有：ICMP、IGMP、IP（IPV4 IPV6）、ARP、RARP。
6. 数据链路层：建立逻辑连接、进行硬件地址寻址、差错校验等功能。将比特组合成字节进而组合成帧，用MAC地址访问介质，错误发现但不能纠正。
7. 物理层：建立、维护、断开物理连接。

* TCP/IP协议

IP协议是一种非常重要的协议。

IP（internet protocal）又称为互联网协议。

IP的责任就是把数据从源传送到目的地。

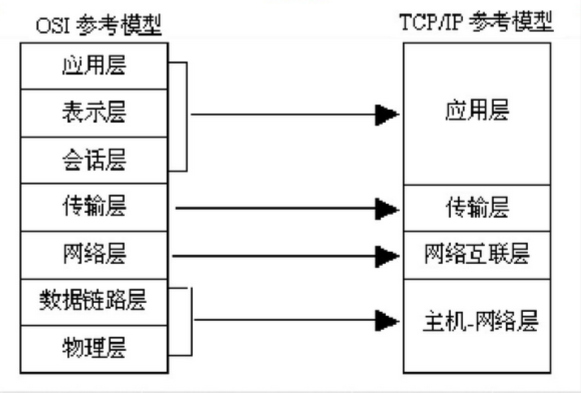
它在源地址和目的地址之间传送一种称之为数据包的东西，它还提供对数据大小的重新组装功能，以适应不同网络对包大小的要求。

经常与IP协议放在一起的还有TCP（Transmission Control Protocol）协议，即传输控制协议**，是一种面向连接的、可靠的、基于字节流的传输层通信协议。**

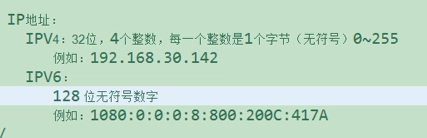
而通常我们说的**TCP/IP协议，其实是指TCP/IP协议族**，因为该协议家族的两个最核心协议：TCP（传输控制协议）和IP（网际协议），为该家族中最早通过的标准，所以简称为TCP/IP协议。

按照TCP/IP协议模型，网络通常被分为四层：网络访问层、互联网层、传输层和应用层。

**UDP:非面向连接，不可靠的，基于用户数据报的，最多支持64K以内的数据的发送、**



### 15.1.3 IP地址





* IP地址用于标识网络中的一个通信实体，这个通信实体可以是一台计算机，也可以是一台打印机，或者是路由器的一个端口。
* 在基于IP协议网络中传输的数据包，都必须使用IP地址来进行标识。

如同我们写信，发快递一样，要标明收件人的通信地址和发件人的通信地址，而邮政人员和物流快递员则通过该地址决定信件、包裹的去向。

* IP地址是一个32位的整数；

为了便于记忆，通常把它分为4个8位的二进制数组成，每8位之间用圆点隔开，格式：X.X.X.X，其中每个X表示地址中的8位，用十进制[0,255]之间值表示，因此我们看到的IP常常是：222.222.88.104。

* IP地址类型

Internet委员会定义了5种IP地址类型以适合不同容量的网络，即A类~E类。其中A、B、C这3类由Internet NIC在全球范围内统一分配，D、E类为特殊地址。



* IPV4 & IPV6

IP地址还分为IPV4和IPV6。由于IPv4最大的问题在于网络地址资源有限，严重制约了互联网的应用和发展。IPv6是IETF（Internet Engineering Task Force）设计的用于替代现行版本IP协议（IPv4）的下一代IP协议，号称可以为全世界的每一粒沙子编上一个网址。IPv4和IPv6地址格式不相同，因此在很长一段时间里，互联网中出现IPv4和IPv6长期共存的局面。2012年6月6日，国际互联网协会举行了世界IPv6启动纪念日，这一天，全球IPv6网络正式启动。多家知名网站，如Google、Facebook和Yahoo等，于当天全球标准时间0点（北京时间8点整）开始永久性支持IPv6访问。2018年6月，三大运营商联合阿里云宣布，将全面对外提供IPv6服务，并计划在2025年前助推中国互联网真正实现“IPv6 Only”。 7月，百度云制定了中国的IPv6改造方案。8月3日，工信部通信司在北京召开IPv6规模部署及专项督查工作全国电视电话会议，中国将分阶段有序推进规模建设IPv6网络，实现下一代互联网在经济社会各领域深度融合。

IPv6的地址长度为128位，是IPv4地址长度的4倍，格式为X:X:X:X:X:X:X:X，其中每个X表示地址中的16位，以十六进制表示。

### 15.1.4 端口号

IP地址可以唯一的确定网络上的一个通信实体，但是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。IP地址就好比通信的街道和门牌号，我们通过IP地址可以找到房子，但是要具体找到某个人，还需要房间号或名字。

**端口号是一个16位的整数，即在[0,65535]之间**，通常它可以分为三类：

1. **公认端口（Well-Known Ports）**

范围从0到1023，这些端口号一般固定分配给一些服务。

比如21端口分配给FTP(文件传输协议)服务，25端口分配给SMTP（简单邮件传输协议）服务，80端口分配给HTTP服务。

1. **注册端口（Registered Ports）**

端口号从1024到49151。

它们松散地绑定于一些服务。例如：Tomcat(8080)，JBOSS(8080)，Oracle（1521）,MySQL(3306)，SQL Server(1433)，QQ（1080）。

1. **动态/私有端口（Dynamic and/Private Ports）**

端口的范围从49152到65535，这些端口号一般不固定分配给某个服务。只要运行的程序向系统提出访问网络的申请，那么系统就可以从这些端口号中分配一个供该程序使用。理论上，不应为服务分配这些端口。

* 域名

然而实际生活中，要常人识记IP地址还是有些困难的，所以就有了域名。例如：尚硅谷的服务器IP地址是222.222.88.104，为了让大家便于记忆和访问，我们注册了域名：www.atguigu.com。域名和IP地址是相对应的，网域名称系统（DNS，Domain Name System）是因特网的一项核心服务，它作为可以将域名和IP地址相互映射的一个分布式数据库，能够使人更方便的访问互联网，而不用去记住能够被机器直接读取的IP地址数串。

## 15.2 基本的网络API

### 15.2.1 使用InetAddress

此类表示互联网协议 (IP) 地址，它有两个子类Inet4Address和Inet6Address，分别对应IPV4和IPV6。

* InetAddress类没有提供公共的构造器，而是提供了如下几个静态方法来获取InetAddress实例。
* public static InetAddress getLocalHost()
* public static InetAddress getByAddress(byte[] addr)
* public static InetAddress getByName(String host)

InetAddress提供了如下几个常用的方法：

* public String getHostAddress()：返回 IP 地址字符串（以文本表现形式）。
* public String getHostName()：获取此 IP 地址的主机名
* public String getCanonicalHostName()：获取此 IP 地址的完全限定域名
* public boolean isReachable(int timeout)：测试是否可以达到该地址。

|  |
| --- |
| **package** com.atguigu.ip;  **import** java.io.UnsupportedEncodingException;  **import** java.net.InetAddress;  **import** java.net.UnknownHostException;  **public** **class** TestInetAddress {  **public** **static** **void** main(String[] args) **throws** Exception{  InetAddress ip1 = InetAddress.*getLocalHost*();  System.***out***.println(ip1);//Irene-PC/192.168.1.107    InetAddress ip2 = InetAddress.*getByName*("www.atguigu.com");  System.***out***.println(ip2);//www.atguigu.com/222.222.88.102    **byte**[] ip = {(**byte**)222,(**byte**)222,88,102};  InetAddress ip3 = InetAddress.*getByAddress*(ip);  System.***out***.println(ip3);//输出ip而不是域名。如果这个IP地址不存在或DNS服务器不允许进行ip-->域名的映射，getHostName()方法就直接返回这个IP地址  }  } |

### 15.2.2 使用URL

URI(Uniform resource identifier)：表示一个统一资源标识符 (URI) 引用，用来唯一的标识一个资源。

URL(Uniform Resource Locator)：类 URL 代表一个统一资源定位符，它是指向互联网“资源”的指针。资源可以是简单的文件或目录，也可以是对更为复杂的对象的引用，例如对数据库或搜索引擎的查询。它是一种具体的URI，即URL可以用来标识一个资源，而且还指明了如何locate这个资源。URI不能用于定位任何资源，它的唯一作用是解析，而URL则包含一个可打开到达该资源的输入流。 URL的基本结构由5部分组成：

|  |
| --- |
| <传输协议>://<主机名>:<端口号>/<文件名>#片段名 |
| <传输协议>://<主机名>:<端口号>/<文件名>?参数列表 |

其中#片段名：即锚点，例如看小说，直接定位到章节

例如：http://java.sun.com/index.html#chapter1

参数列表格式：参数名=参数值&参数名=参数值....

例如: http://192.168.1.100:8080/helloworld/index.jsp?username=chai&password=123

**URL构造方法：**

* public URL (String spec)：通过一个表示URL地址的字符串可以构造一个URL对象。例如：**URL url = new URL ("http://www. atguigu.com/");**
* public URL(URL context, String spec)：通过基 URL 和相对 URL 构造一个 URL 对象。例如：**URL downloadUrl = new URL(url, "download.html")**
* public URL(String protocol, String host, String file); 例如：**new URL("http", "www.atguigu.com", “download. html");**
* public URL(String protocol, String host, int port, String file); 例如: **URL gamelan = new URL("http", "www.atguigu.com", 80, “download.html");**

**URL的常用方法：**

* public String getProtocol()：获取该URL的协议名
* public String getHost()：获取该URL的主机名
* public String getPort()：获取该URL的端口号
* public String getPath()：获取该URL的文件路径，即tomcat的webapps目录下级目录名也就是资源路径名
* public String getFile()：获取该URL的文件名
* public String getRef()：获取该URL在文件中的相对位置
* public String getQuery()：获取该URL的查询名
* public final InputStream openStream()：返回一个用于从该连接读入的 InputStream。

|  |
| --- |
| **package** com.atguigu.url;  **import** java.net.MalformedURLException;  **import** java.net.URL;  **public** **class** TestURL {  **public** **static** **void** main(String[] args) **throws** MalformedURLException {  URL url = **new** URL("http://www.baidu.com:80/index.html?keyword=java");  System.***out***.println("协议：" + url.getProtocol());  System.***out***.println("主机名：" + url.getHost());  System.***out***.println("端口号：" + url.getPort());  System.***out***.println("路径名：" + url.getPath());  System.***out***.println("文件名：" + url.getFile());  System.***out***.println("锚点：" + url.getRef());  System.***out***.println("查询名：" + url.getQuery());//注意如果存在锚点，那么查询名返回null，因为#后面全部当做锚点了  }  } |

案例：获取百度首页的资源

|  |
| --- |
| **package** com.atguigu.url;  **import** java.io.InputStream;  **import** java.net.URL;  **public** **class** TestURLRead {  **public** **static** **void** main(String[] args)**throws** Exception {  URL url = **new** URL("http://www.baidu.com/index.html");  /\*html标签（格式化数据）、css：皮肤（美化数据）、js：互动（交互数据）、数据：\*/  InputStream input = url.openStream();  **byte**[] data = **new** **byte**[1024];  **int** len;  **while**((len=input.read(data))!=-1){  System.***out***.println(**new** String(data,0,len,"UTF-8"));  }  input.close();  }  } |

### 15.2.3 使用URLConnection

URL的方法openStream()，能从网络上读取数据，但是无法给服务器端发送数据，若希望给服务器端发送数据，则需要URLConnection。

它代表应用程序和 URL 之间的通信链接。此类的实例可用于读取和写入此 URL 引用的资源。通常，创建一个到 URL 的连接需要几个步骤：

1. 通过 URL对象调用 openConnection 方法创建URLConnection连接对象。
2. 处理设置参数和一般请求属性。
3. 使用 connect 方法建立到远程对象的实际连接。
4. 远程对象变为可用。远程对象的头字段和内容变为可访问。

URLConnection的常用方法：

* public void setDoOutput(boolean dooutput)如果打算使用 URL 连接进行输出，例如给服务器传递请求参数，则将 DoOutput 标志设置为 true；如果不打算使用，则设置为 false。默认值为 false。 **必须在所有getXXX()和connect()方法之前。**
* public String getContentEncoding()：返回 content-encoding 头字段的值。
* public int getContentLength()：返回 content-length 头字段的值。
* public String getContentType()：返回 content-type 头字段的值。
* public long getDate()：返回 date 头字段的值。
* public long getLastModified()：返回 last-modified 头字段的值。结果为距离格林威治标准时间 1970 年 1 月 1 日的毫秒数。
* public String getHeaderField(String name)：返回指定的头字段的值。
* public InputStream getInputStream()返回从此打开的连接读取的输入流。
* public OutputStream getOutputStream()返回写入到此连接的输出流。

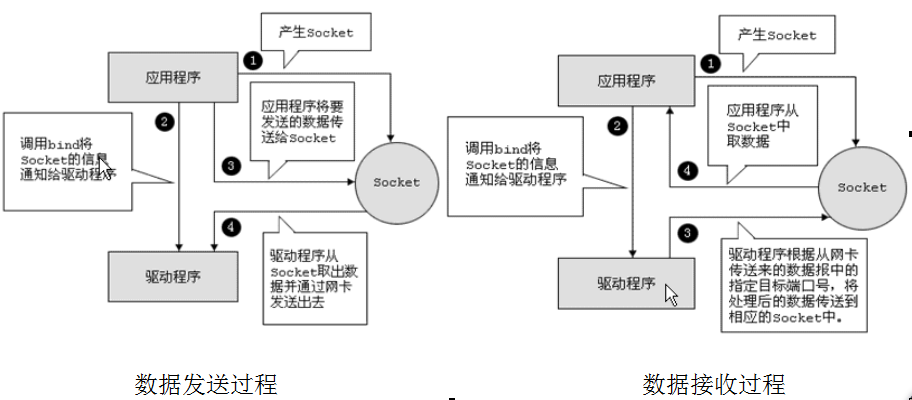
|  |
| --- |
| **package** com.atguigu.url;  **import** java.io.BufferedReader;  **import** java.io.InputStream;  **import** java.io.InputStreamReader;  **import** java.net.URL;  **import** java.net.URLConnection;  **public** **class** TestURLConnection {  **public** **static** **void** main(String[] args) **throws** Exception{  URL url = **new** URL("http://localhost/web1/denglu");  //通过 URL对象调用 openConnection 方法创建URLConnection连接对象  URLConnection uc = url.openConnection();  //处理设置参数  uc.setDoOutput(**true**);  //给服务器发送请求参数  uc.getOutputStream().write("username=admin&password=123".getBytes());  //使用 connect 方法建立到远程对象的实际连接。  uc.connect();  //获取资源  InputStream is = uc.getInputStream();  BufferedReader br = **new** BufferedReader(**new** InputStreamReader(is,"UTF-8"));  String str;  **while**((str=br.readLine())!=**null**){  System.***out***.println(str);  }  br.close();  }  } |

### 15.2.4 Socket

通信的两端都要有Socket（也可以叫“套接字”），是两台机器间通信的端点。网络通信其实就是Socket间的通信。

Socket可以和网卡驱动进行数据交换，把数据交给网卡驱动或者从网卡驱动中提取数据

* Socket分类：
* 流套接字（stream socket）：使用TCP提供可依赖的字节流服务
* 数据报套接字（datagram socket）：使用UDP提供“尽力而为”的数据报服务



Socket类的常用构造方法：

* public 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。
* public 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。

Socket类的常用方法：

* public InputStream getInputStream()：返回此套接字的输入流，可以用于接收消息
* public OutputStream getOutputStream()：返回此套接字的输出流，可以用于发送消息
* public InetAddress getInetAddress()：获取此套接字连接到的远程 IP 地址；如果套接字是未连接的，则返回 null。
* public InetAddress getLocalAddress()：获取套接字绑定的本地地址。
* public int getPort()： 此套接字连接到的远程端口号；如果尚未连接套接字，则返回 0。
* public int getLocalPort()：返回此套接字绑定到的本地端口。如果尚未绑定套接字，则返回 -1。
* public void close()：关闭此套接字。套接字被关闭后，便不可在以后的网络连接中使用（即无法重新连接或重新绑定）。需要创建新的套接字对象。 关闭此套接字也将会关闭该套接字的 InputStream 和 OutputStream。
* public void shutdownInput()：如果在套接字上调用 shutdownInput() 后从套接字输入流读取内容，则流将返回 EOF（文件结束符）。 即不能在从此套接字的输入流中接收任何数据。
* public void shutdownOutput()：禁用此套接字的输出流。对于 TCP 套接字，任何以前写入的数据都将被发送，并且后跟 TCP 的正常连接终止序列。 如果在套接字上调用 shutdownOutput() 后写入套接字输出流，则该流将抛出 IOException。 即不能通过此套接字的输出流发送任何数据。

**注意：**先后调用Socket的shutdownInput()和shutdownOutput()方法，仅仅关闭了输入流和输出流，并不等于调用Socket的close()方法。在通信结束后，仍然要调用Scoket的close()方法，因为只有该方法才会释放Socket占用的资源，比如占用的本地端口号等。

## 15.3 基于TCP协议的网络编程

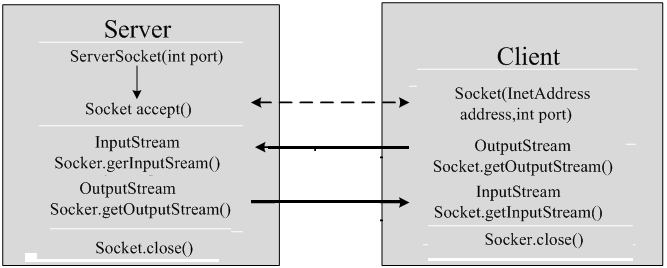
TCP(Transmission Control Protocol，传输控制协议)被称作一种端对端协议。是一种面向连接的、可靠的、基于字节流的传输层的通信协议，可以连续传输大量的数据。类似于打电话的效果。

这是因为它为当一台计算机需要与另一台远程计算机连接时，TCP协议会采用“三次握手”方式让它们建立一个连接，用于发送和接收数据的虚拟链路。数据传输完毕TCP协议会采用“四次挥手”方式断开连接。

TCP协议负责收集这些信息包，并将其按适当的次序放好传送，在接收端收到后再将其正确的还原。TCP协议保证了数据包在传送中准确无误。TCP协议使用重发机制，当一个通信实体发送一个消息给另一个通信实体后，需要收到另一个通信实体确认信息，如果没有收到另一个通信实体确认信息，则会再次重复刚才发送的消息。

### 15.3.1 基于TCP协议的网络通信程序结构

Java语言的基于套接字TCP编程分为服务端编程和客户端编程，其通信模型如图所示：



服务器程序的工作过程包含以下五个基本的步骤：

1. 使用 ServerSocket(int port) ：创建一个服务器端套接字，并绑定到指定端口上。用于监听客户端的请求。
2. 调用 accept()方法：监听连接请求，如果客户端请求连接，则接受连接，创建与该客户端的通信套接字对象。否则该方法将一直处于等待状态。
3. 调用 该Socket对象的 getOutputStream() 和 getInputStream ()：获取输出流和输入流，开始网络数据的发送和接收。
4. 关闭Socket对象：某客户端访问结束，关闭与之通信的套接字。
5. 关闭ServerSocket：如果不再接收任何客户端的连接的话，调用close()进行关闭。

客户端Socket的工作过程包含以下四个基本的步骤：

1. 创建 Socket：根据指定服务端的 IP 地址或端口号构造 Socket 类对象，创建的同时会自动向服务器方发起连接。若服务器端响应，则建立客户端到服务器的通信线路。若连接失败，会出现异常。
2. 打开连接到Socket 的输入/出流：使用 getInputStream()方法获得输入流，使用 getOutputStream()方法获得输出流，进行数据传输。
3. 进行读/写操作：通过输入流读取服务器发送的信息，通过输出流将信息发送给服务器。
4. 关闭 Socket：断开客户端到服务器的连接

注意：客户端和服务器端在获取输入流和输出流时要对应，否则容易死锁。例如：客户端先获取字节输出流（即先写），那么服务器端就先获取字节输入流（即先读）；反过来客户端先获取字节输入流（即先读），那么服务器端就先获取字节输出流（即先写）。

### 15.3.2 案例：一个客户端与服务器单次通信

需求：客户端连接服务器，连接成功后给服务发送“lalala”，服务器收到消息后，给客户端返回“欢迎登录”

服务器端程序示例代码：

|  |
| --- |
| **package** com.atguigu.tcp.one;  **import** java.io.InputStream;  **import** java.io.OutputStream;  **import** java.net.ServerSocket;  **import** java.net.Socket;  **public** **class** Server {  **public** **static** **void** main(String[] args)**throws** Exception {  //1、准备一个ServerSocket对象，并绑定8888端口  ServerSocket server = **new** ServerSocket(8888);  System.***out***.println("等待连接....");   //2、在8888端口监听客户端的连接，该方法是个阻塞的方法，如果没有客户端连接，将一直等待 Socket socket = server.accept();  System.***out***.println("一个客户端连接成功！！");    //3、获取输入流，用来接收该客户端发送给服务器的数据  InputStream input = socket.getInputStream();  //接收数据  **byte**[] data = **new** **byte**[1024];  StringBuilder s = **new** StringBuilder();  **int** len;  **while** ((len = input.read(data)) != -1) {  s.append(**new** String(data, 0, len));  }  System.***out***.println("客户端发送的消息是：" + s);    //4、获取输出流，用来发送数据给该客户端  OutputStream out = socket.getOutputStream();  //发送数据  out.write("欢迎登录".getBytes());  out.flush();    //5、关闭socket，不再与该客户端通信  //socket关闭，意味着InputStream和OutputStream也关闭了  socket.close();    //6、如果不再接收任何客户端通信，可以关闭ServerSocket  server.close();  }  } |

客户端程序示例代码：

|  |
| --- |
| **package** com.atguigu.tcp.one;  **import** java.io.InputStream;  **import** java.io.OutputStream;  **import** java.net.Socket;  **public** **class** Client {  **public** **static** **void** main(String[] args) **throws** Exception {  // 1、准备Socket，连接服务器，需要指定服务器的IP地址和端口号  Socket socket = **new** Socket("127.0.0.1", 8888);  // 2、获取输出流，用来发送数据给服务器  OutputStream out = socket.getOutputStream();  // 发送数据  out.write("lalala".getBytes());  **//会在流末尾写入一个“流的末尾”标记，对方才能读到-1，否则对方的读取方法会一致阻塞**  socket.shutdownOutput();    //3、获取输入流，用来接收服务器发送给该客户端的数据  InputStream input = socket.getInputStream();  // 接收数据  **byte**[] data = **new** **byte**[1024];  StringBuilder s = **new** StringBuilder();  **int** len;  **while** ((len = input.read(data)) != -1) {  s.append(**new** String(data, 0, len));  }  System.***out***.println("服务器返回的消息是：" + s);    //4、关闭socket，不再与服务器通信，即断开与服务器的连接  //socket关闭，意味着InputStream和OutputStream也关闭了  socket.close();  }  } |

运行程序时，必须先启动服务器端程序，主动接收客户端程序的连接请求。

### 15.3.3 案例：多个客户端与服务器之间的多次通信

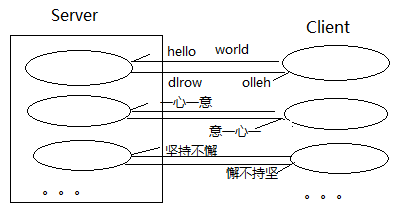
通常情况下，服务器不应该只接受一个客户端请求，而应该不断地接受来自客户端的所有请求，所以Java程序通常会通过循环，不断地调用ServerSocket的accept()方法。

如果服务器端要“同时”处理多个客户端的请求，因此服务器端需要为每一个客户端单独分配一个线程来处理，否则无法实现“同时”。

咱们之前学习IO流的时候，提到过装饰者设计模式，该设计使得不管底层IO流是怎样的节点流：文件流也好，网络Socket产生的流也好，程序都可以将其包装成处理流，甚至可以多层包装，从而提供更多方便的处理。

案例需求：多个客户端连接服务器，并进行多次通信

* 每一个客户端连接成功后，从键盘输入英文单词或中国成语，并发送给服务器
* 服务器收到客户端的消息后，把词语“反转”后返回给客户端
* 客户端接收服务器返回的“词语”，打印显示
* 当客户端输入“stop”时断开与服务器的连接
* 多个客户端可以同时给服务器发送“词语”，服务器可以“同时”处理多个客户端的请求



思考分析：服务器端要“同时”处理多个客户端的请求，那么必须使用多线程，每一个客户端的通信需要单独的线程来处理。

客户端程序示例代码：

|  |
| --- |
| **package** com.atguigu.tcp.echo;  **import** java.io.BufferedReader;  **import** java.io.InputStream;  **import** java.io.InputStreamReader;  **import** java.io.OutputStream;  **import** java.io.PrintStream;  **import** java.net.Socket;  **import** java.util.Scanner;  **public** **class** Client {  **public** **static** **void** main(String[] args) **throws** Exception {  // 1、准备Socket，连接服务器，需要指定服务器的IP地址和端口号  Socket socket = **new** Socket("127.0.0.1", 8888);  // 2、获取输出流，用来发送数据给服务器  OutputStream out = socket.getOutputStream();  PrintStream ps = **new** PrintStream(out);    // 3、获取输入流，用来接收服务器发送给该客户端的数据  InputStream input = socket.getInputStream();  BufferedReader br = **new** BufferedReader(**new** InputStreamReader(input));    Scanner scanner = **new** Scanner(System.***in***);  **while**(**true**){  System.***out***.println("输入发送给服务器的单词或成语：");  String message = scanner.nextLine();  **if**(message.equals("stop")){  socket.shutdownOutput();  **break**;  }    // 4、 发送数据  ps.println(message);  // 接收数据  String feedback = br.readLine();  System.***out***.println("从服务器收到的反馈是：" + feedback);  }    //5、关闭socket，断开与服务器的连接  scanner.close();  socket.close();  }  } |

服务器端程序示例代码：

|  |
| --- |
| **package** com.atguigu.tcp.echo;  **import** java.net.ServerSocket;  **import** java.net.Socket;  **public** **class** Server {  **public** **static** **void** main(String[] args)**throws** Exception {  // 1、准备一个ServerSocket  ServerSocket server = **new** ServerSocket(8888);  System.***out***.println("等待连接...");    **int** count = 0;  **while**(**true**){  // 2、监听一个客户端的连接  Socket socket = server.accept();  System.***out***.println("第" + ++count + "个客户端连接成功！！");    ClientHandlerThread ct = **new** ClientHandlerThread(socket);  ct.start();  }    //这里没有关闭server，永远监听  }  } |

服务器端处理客户端请求的线程类示例代码：

|  |
| --- |
| **package** com.atguigu.tcp.echo;  **import** java.io.BufferedReader;  **import** java.io.IOException;  **import** java.io.InputStreamReader;  **import** java.io.PrintStream;  **import** java.net.Socket;  **public** **class** ClientHandlerThread **extends** Thread{  **private** Socket socket;  **public** ClientHandlerThread(Socket socket) {  **super**();  **this**.socket = socket;  }    **public** **void** run(){    **try**{  //（1）获取输入流，用来接收该客户端发送给服务器的数据  BufferedReader br = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));  //（2）获取输出流，用来发送数据给该客户端  PrintStream ps = **new** PrintStream(socket.getOutputStream());  String str;  // （3）接收数据  **while** ((str = br.readLine()) != **null**) {  //（4）反转  StringBuilder word = **new** StringBuilder(str);  word.reverse();    //（5）返回给客户端  ps.println(word);  }  }**catch**(Exception e){  e.printStackTrace();  }**finally**{  **try** {  //（6）断开连接  socket.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  } |

### 15.3.4 案例：多个客户端上传文件

需求：每一个客户端启动后都可以给服务器上传一个文件，服务器接收到文件后保存到一个upload目录中。可以同时接收多个客户端的文件上传。

思考分析：

1. 服务器端要“同时”处理多个客户端的请求，那么必须使用多线程，每一个客户端的通信需要单独的线程来处理。
2. 服务器保存上传文件的目录只有一个upload，而每个客户端给服务器发送的文件可能重名，所以需要保证文件名的唯一。我们可以使用“时间戳”作为文件名，而后缀名不变
3. 客户端需要给服务器上传文件名（含后缀名）以及文件内容。而文件名是字符串，文件内容不一定是纯文本的，因此选择DataOutputStream和DataInputStream。

客户端的示例代码：

|  |
| --- |
| **package** com.atguigu.tcp.fileupload;  **import** java.io.BufferedReader;  **import** java.io.DataOutputStream;  **import** java.io.File;  **import** java.io.FileInputStream;  **import** java.io.InputStream;  **import** java.io.InputStreamReader;  **import** java.io.OutputStream;  **import** java.net.Socket;  **import** java.util.Scanner;  **public** **class** Client {  **public** **static** **void** main(String[] args) **throws** Exception {  // (1)连接服务器  Socket socket = **new** Socket("192.168.34.53", 8888);  Scanner input = **new** Scanner(System.***in***);    //(2)从键盘输入文件的路径和名称  System.***out***.print("请选择要上传的文件：");  String path = input.nextLine();  File file = **new** File(path);    OutputStream out = socket.getOutputStream();  DataOutputStream dos = **new** DataOutputStream(out);//用它的目的是为了既可以单独传一个字符串，又可以写字节内容    //先发送文件名（含后缀名）  dos.writeUTF(file.getName());//单独发一个字符串    //还需要一个IO流，从文件读取内容，给服务器发过去  FileInputStream fis = **new** FileInputStream(file);    //（3）把文件内容给服务器传过去，类似与复制文件  **byte**[] data = **new** **byte**[1024];  **while**(**true**){  **int** len = fis.read(data);  **if**(len==-1){  **break**;  }  dos.write(data, 0, len);  }  socket.shutdownOutput();    //(4)接收服务器返回的结果  InputStream is = socket.getInputStream();  InputStreamReader isr = **new** InputStreamReader(is);//把字节流转成字符流  BufferedReader br = **new** BufferedReader(isr);  String result = br.readLine();  System.***out***.println(result);    //(5)关闭  fis.close();  input.close();  socket.close();  }  } |

服务器端的示例代码：

|  |
| --- |
| **package** com.atguigu.tcp.fileupload;  **import** java.net.ServerSocket;  **import** java.net.Socket;  **public** **class** Server {  **public** **static** **void** main(String[] args) **throws** Exception{  //服务器在8888端口号监听数据  @SuppressWarnings("resource")  ServerSocket server = **new** ServerSocket(8888);    **while**(**true**){  //(2)等待连接  //这句代码执行一次，意味着一个客户端连接  Socket accept = server.accept();    FileUploadThread ft = **new** FileUploadThread(accept);  ft.start();  }  }  } |

服务器端处理每一个客户端上传文件的线程类代码：

|  |
| --- |
| **package** com.atguigu.tcp.fileupload;  **import** java.io.DataInputStream;  **import** java.io.FileNotFoundException;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **import** java.io.InputStream;  **import** java.io.OutputStream;  **import** java.io.PrintStream;  **import** java.net.Socket;  **import** java.text.SimpleDateFormat;  **import** java.util.Date;  **public class** FileUploadThread **extends** Thread{  **private** Socket socket;  **private** String dir = "upload/";  **public** FileUploadThread(Socket socket) {  **super**();  **this**.socket = socket;  }    **public** **void** run(){  FileOutputStream fos = **null**;  **try** {  InputStream is = socket.getInputStream();  DataInputStream dis = **new** DataInputStream(is);    //读取文件名（含后缀名）  String filename = dis.readUTF();  //截取后缀名  String ext = filename.substring(filename.lastIndexOf("."));  //生成时间戳  SimpleDateFormat sf = **new** SimpleDateFormat("yyyyMMddHHmmssSSS");  filename = sf.format(**new** Date());  //用新文件路径构建文件输出流  fos = **new** FileOutputStream(dir + filename + ext);  //接收文件内容  **byte**[] data = **new** **byte**[1024];  **while**(**true**){  **int** len = is.read(data);  **if**(len==-1){  **break**;  }  fos.write(data, 0, len);  }    //返回结果  OutputStream out = socket.getOutputStream();  PrintStream ps = **new** PrintStream(out);  ps.println(filename + ext + "：已上传完毕");  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }**finally**{  **try** {  fos.close();  socket.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  } |

### 15.3.5 案例：多个客户端群聊

需求：模拟聊天室群聊

客户端要先登录，登录成功之后才能发送和接收消息

思考分析：

* 服务器端，需要为每个客户端开启一个线程通信，这样才能实现多个客户端“同时”与服务器通信
* 客户端，需要把收消息功能与发消息功能分开两个线程，这样才能“同时收发”，即可以发消息，也可以接收其他客户端的聊天消息
* 服务器端要分别处理客户端的“登录”、“退出”、“聊天”的消息，所以这里设计了Code常量类，用状态值区分“登录”、“退出”、“聊天”
* 这里设计Message类，包含code属性，区别是“登录”、“退出”、“聊天”，username属性表示用户名，表明消息是谁发的，content属性，存储消息内容，如果是登录，就用来存储密码。
* 这里消息是Message对象，因此在客户端与服务器之间传输的是对象，所以选择ObjectOutputStream和ObjectInputStream。
* 这里的Message类与Code类是服务器端和客户端“共享”的，要保持一致。特别注意包名和序列化版本ID。

Message类与Code类的示例代码：

|  |
| --- |
| **package** com.atguigu.tcp.chat.bean;  **public** **class** Code {  **public** **static** **final** **int** ***LOGIN*** = 1;  **public** **static** **final** **int** ***CHAT*** = 2;  **public** **static** **final** **int** ***LOGOUT*** = 3;    **public** **static** **final** **int** ***SUCCESS*** = 1;  **public** **static** **final** **int** ***FAIL*** = 2;  } |
| **package** com.atguigu.tcp.chat.bean;  **import** java.io.Serializable;  **public** **class** Message **implements** Serializable{  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **private** **int** code;  **private** String username;  **private** String content;  **public** Message(**int** code, String username, String content) {  **super**();  **this**.code = code;  **this**.username = username;  **this**.content = content;  }  **public** Message() {  **super**();  }  **public** **int** getCode() {  **return** code;  }  **public** **void** setCode(**int** code) {  **this**.code = code;  }  **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** String getContent() {  **return** content;  }  **public** **void** setContent(String content) {  **this**.content = content;  }  } |

服务器端用户管理类代码：

|  |
| --- |
| **package** com.atguigu.tcp.chat.server;  **import** java.util.HashMap;  **public** **class** UserManager {  **public** **static** HashMap<String,String> *allUsers* = **new** HashMap<String,String>();  **static**{  *allUsers*.put("gangge", "123");  *allUsers*.put("xiaobai", "456");  *allUsers*.put("gujie", "789");  }    **public** **static** **boolean** login(String username, String password){  **if**(*allUsers*.get(username)!=**null** && *allUsers*.get(username).equals(password)){  **return** **true**;  }**else**{  **return** **false**;  }  }  } |

服务器端示例代码：

|  |
| --- |
| **package** com.atguigu.tcp.chat.server;  **import** java.net.ServerSocket;  **import** java.net.Socket;  **public** **class** Server {  **public** **static** **void** main(String[] args)**throws** Exception {  @SuppressWarnings("resource")  ServerSocket server = **new** ServerSocket(9999);    **while**(**true**){  Socket socket = server.accept();    ClientHandlerThread ct = **new** ClientHandlerThread(socket);  ct.start();  }  }  } |

服务器端处理消息的线程类代码：

|  |
| --- |
| **package** com.atguigu.tcp.chat.server;  **import** java.io.IOException;  **import** java.io.ObjectInputStream;  **import** java.io.ObjectOutputStream;  **import** java.net.Socket;  **import** java.util.ArrayList;  **import** java.util.Collections;  **import** java.util.HashSet;  **import** java.util.Set;  **import** com.atguigu.tcp.chat.bean.Code;  **import** com.atguigu.tcp.chat.bean.Message;  **public** **class** ClientHandlerThread **extends** Thread{  **public** **static** Set<ObjectOutputStream> *online* = Collections.*synchronizedSet*(**new** HashSet<ObjectOutputStream>());  **private** Socket socket;  **private** String username;  **private** ObjectInputStream ois;  **private** ObjectOutputStream oos;    **public** ClientHandlerThread(Socket socket) {  **super**();  **this**.socket = socket;  }    **public** **void** run(){  Message message = **null**;  **try**{  ois = **new** ObjectInputStream(socket.getInputStream());  oos = **new** ObjectOutputStream(socket.getOutputStream());    //接收数据  **while** (**true**) {  message = (Message) ois.readObject();    **if**(message.getCode() == Code.***LOGIN***){  //如果是登录，则验证用户名密码  username = message.getUsername();  String password = message.getContent();  **if**(UserManager.*login*(username, password)){  message.setCode(Code.***SUCCESS***);  oos.writeObject(message);    //并将该用户添加到在线人员名单中  *online*.add(oos);    message.setCode(Code.***CHAT***);  message.setContent("上线了");  //通知其他人，xx上线了  sendToOther(message);  }**else**{  message.setCode(Code.***FAIL***);  oos.writeObject(message);  }  }**else** **if**(message.getCode() == Code.***CHAT***){  //如果是聊天信息，把消息转发给其他在线客户端  sendToOther(message);  }**else** **if**(message.getCode() == Code.***LOGOUT***){  //通知其他人，xx下线了  message.setContent("下线了");  sendToOther(message);  **break**;  }  }  }**catch**(Exception e){  //通知其他人，xx掉线了  **if**(message!=**null** && username!=**null**){  message.setCode(Code.***LOGOUT***);  message.setContent("掉线了");  sendToOther(message);  }  }**finally**{  //从在线人员中移除并断开当前客户端  **try** {  *online*.remove(oos);  socket.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  **private** **void** sendToOther(Message message) {  ArrayList<ObjectOutputStream> offline = **new** ArrayList<ObjectOutputStream>();  **for** (ObjectOutputStream on : *online*) {  **if**(!on.equals(oos)){  **try** {  on.writeObject(message);  } **catch** (IOException e) {  offline.add(on);  }  }  }    **for** (ObjectOutputStream off : offline) {  *online*.remove(off);  }  }  } |

客户端示例代码：

|  |
| --- |
| **package** com.atguigu.tcp.chat.client;  **import** java.io.ObjectInputStream;  **import** java.io.ObjectOutputStream;  **import** java.net.Socket;  **import** java.util.Scanner;  **import** com.atguigu.tcp.chat.bean.Code;  **import** com.atguigu.tcp.chat.bean.Message;  **public** **class** Client {  **public** **static** **void** main(String[] args) **throws** Exception{  Socket socket = **new** Socket("192.168.1.107", 9999);    ObjectOutputStream oos = **new** ObjectOutputStream(socket.getOutputStream());  ObjectInputStream ois = **new** ObjectInputStream(socket.getInputStream());  //先登录  Scanner scanner = **new** Scanner(System.***in***);  String username;  **while**(**true**){  //输入登录信息  System.***out***.println("用户名：");  username = scanner.nextLine();  System.***out***.println("密码：");  String password = scanner.nextLine();    Message msg = **new** Message(Code.***LOGIN***, username, password);  //发送登录数据  oos.writeObject(msg);  // 接收登录结果  msg = (Message) ois.readObject();  **if**(msg.getCode() == Code.***SUCCESS***){  System.***out***.println("登录成功！");  **break**;  }**else** **if**(msg.getCode() == Code.***FAIL***){  System.***out***.println("用户名或密码错误，登录失败，重新输入");  }  }    //启动收消息和发消息线程  SendThread s = **new** SendThread(oos,username);  ReceiveThread r = **new** ReceiveThread(ois);  s.start();  r.start();    s.join();//不发了，就结束  r.setFlag(**false**);  r.join();    scanner.close();  socket.close();  }  } |

客户端发消息线程类代码：

|  |
| --- |
| **package** com.atguigu.tcp.chat.client;  **import** java.io.IOException;  **import** java.io.ObjectOutputStream;  **import** java.util.Scanner;  **import** com.atguigu.tcp.chat.bean.Code;  **import** com.atguigu.tcp.chat.bean.Message;  **public** **class** SendThread **extends** Thread{  **private** ObjectOutputStream oos;  **private** String username;    **public** SendThread(ObjectOutputStream oos,String username) {  **super**();  **this**.oos = oos;  **this**.username = username;  }  **public** **void** run(){  **try** {  Scanner scanner = **new** Scanner(System.***in***);  **while**(**true**){  System.***out***.println("请输入消息内容：");  String content = scanner.nextLine();  Message msg;  **if**("bye".equals(content)){  msg = **new** Message(Code.***LOGOUT***, username, content);  oos.writeObject(msg);  scanner.close();  **break**;  }**else**{  msg = **new** Message(Code.***CHAT***, username, content);  oos.writeObject(msg);  }  }  } **catch** (IOException e) {  e.printStackTrace();  }  }  } |

客户端收消息线程类代码：

|  |
| --- |
| **package** com.atguigu.tcp.chat.client;  **import** java.io.ObjectInputStream;  **import** com.atguigu.tcp.chat.bean.Message;  **public** **class** ReceiveThread **extends** Thread{  **private** ObjectInputStream ois;  **private** **volatile** **boolean** flag = **true**;    **public** ReceiveThread(ObjectInputStream ois) {  **super**();  **this**.ois = ois;  }  **public** **void** run(){  **try** {  **while**(flag){  Message msg = (Message) ois.readObject();  System.***out***.println(msg.getUsername() + ":" + msg.getContent());  }  } **catch** (Exception e) {  System.***out***.println("请重新登录");  }  }  **public** **void** setFlag(**boolean** flag) {  **this**.flag = flag;  }    } |

以上案例的网络通信程序是基于阻塞式API的，所以服务器必须为每个客户端都提供一条独立线程进行处理，当服务器需要同时处理大量客户端时，这种做法会导致性能下降。如果要开发高性能网络服务器，那么需要使用Java提供的NIO API，可以让服务器使用一个或有限几个线程来同时处理连接到服务器上的所有客户端。

## 15.4 基于UDP协议的网络编程

UDP(User Datagram Protocol，用户数据报协议)：是一个无连接的传输层协议、提供面向事务的简单不可靠的信息传送服务，类似于短信。

UDP协议是一种**面向非连接**的协议，面向非连接指的是在正式通信前不必与对方先建立连接，不管对方状态就直接发送，至于对方是否可以接收到这些数据内容，UDP协议无法控制，因此说，UDP协议是一种**不可靠的**协议。无连接的好处就是快，省内存空间和流量，因为维护连接需要创建大量的数据结构。UDP会尽最大努力交付数据，但不保证可靠交付，没有TCP的确认机制、重传机制，如果因为网络原因没有传送到对端，UDP也不会给应用层返回错误信息。

UDP协议是面向数据报文的信息传送服务。UDP在发送端没有缓冲区，对于应用层交付下来的报文在添加了首部之后就直接交付于ip层，不会进行合并，也不会进行拆分，而是一次交付一个完整的报文。比如我们要发送100个字节的报文，我们调用一次send()方法就会发送100字节，接收方也需要用receive()方法一次性接收100字节，不能使用循环每次获取10个字节，获取十次这样的做法。

UDP协议没有拥塞控制，所以当网络出现的拥塞不会导致主机发送数据的速率降低。虽然UDP的接收端有缓冲区，但是这个缓冲区只负责接收，并不会保证UDP报文的到达顺序是否和发送的顺序一致。因为网络传输的时候，由于网络拥塞的存在是很大的可能导致先发的报文比后发的报文晚到达。如果此时缓冲区满了，后面到达的报文将直接被丢弃。这个对实时应用来说很重要，比如：视频通话、直播等应用。

因此UDP适用于一次只传送少量数据、对可靠性要求不高的应用环境，数据报大小限制在64K以下。

### 15.4.1 基于UDP协议的网络编程

基于UDP协议的网络编程仍然需要在通信实例的两端各建立一个Socket，但这两个Socket之间并没有虚拟链路，这两个Socket只是发送、接收数据报的对象，Java提供了DatagramSocket对象作为基于UDP协议的Socket，使用DatagramPacket代表DatagramSocket发送、接收的数据报。

DatagramSocket 类的常用方法：

* public 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。套接字将被绑定到通配符地址，IP 地址由内核来选择。
* public DatagramSocket(int port,InetAddress laddr)创建数据报套接字，将其绑定到指定的本地地址。本地端口必须在 0 到 65535 之间（包括两者）。如果 IP 地址为 0.0.0.0，套接字将被绑定到通配符地址，IP 地址由内核选择。
* public void close()关闭此数据报套接字。
* public void send(DatagramPacket p)从此套接字发送数据报包。DatagramPacket 包含的信息指示：将要发送的数据、其长度、远程主机的 IP 地址和远程主机的端口号。
* public void receive(DatagramPacket p)从此套接字接收数据报包。当此方法返回时，DatagramPacket 的缓冲区填充了接收的数据。数据报包也包含发送方的 IP 地址和发送方机器上的端口号。 此方法在接收到数据报前一直阻塞。数据报包对象的 length 字段包含所接收信息的长度。如果信息比包的长度长，该信息将被截短。

DatagramPacket类的常用方法：

* public DatagramPacket(byte[] buf,int length)构造 DatagramPacket，用来接收长度为 length 的数据包。 length 参数必须小于等于 buf.length。
* public 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。length 参数必须小于等于 buf.length。
* public int getLength()返回将要发送或接收到的数据的长度。

#### 1、“发送端”

步骤流程：

1、建立发送端的DatagramSocket，需要指定本端的端口号

2、建立数据包DatagramPacket

* 数据
* 接收端的IP地址
* 接收端的端口号

3、调用DatagramSocket的发送方法

4、关闭DatagramSocket

示例代码：

|  |
| --- |
| **package** com.atguigu.udp;  **import** java.net.DatagramPacket;  **import** java.net.DatagramSocket;  **import** java.net.InetAddress;  **import** java.util.ArrayList;  **public** **class** Send {  **public** **static** **void** main(String[] args)**throws** Exception {  // 1、建立发送端的DatagramSocket  DatagramSocket ds = **new** DatagramSocket();    //要发送的数据  ArrayList<String> all = **new** ArrayList<String>();  all.add("尚硅谷让天下没有难学的技术！");  all.add("学高端前沿的IT技术来尚硅谷！");  all.add("尚硅谷让你的梦想变得更具体！");  all.add("尚硅谷让你的努力更有价值！");    //接收方的IP地址  InetAddress ip = InetAddress.*getByName*("127.0.0.1");  //接收方的监听端口号  **int** port = 9999;  //发送多个数据报  **for** (**int** i = 0; i < all.size(); i++) {  // 2、建立数据包DatagramPacket  **byte**[] data = all.get(i).getBytes();  DatagramPacket dp = **new** DatagramPacket(data, data.length, ip, port);  // 3、调用Socket的发送方法  ds.send(dp);  }    // 4、关闭Socket  ds.close();  }  } |

#### 2、“接收端”

步骤流程：

1、建立接收端的DatagramSocket，需要指定本端的IP地址和端口号

2、建立数据包DatagramPacket

* + 需要指定装数据的数组

1. 调用Socket的接收方法
2. 拆封数据
3. 关闭Socket

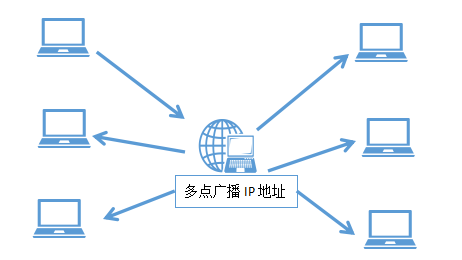
示例代码：

|  |
| --- |
| **package** com.atguigu.udp;  **import** java.net.DatagramPacket;  **import** java.net.DatagramSocket;  **public** **class** Receive {  **public** **static** **void** main(String[] args) **throws** Exception {  // 1、建立接收端的DatagramSocket，需要指定本端的监听端口号  DatagramSocket ds = **new** DatagramSocket(9999);    //一直监听数据  **while**(**true**){  // 2、建立数据包DatagramPacket  **byte**[] buffer = **new** **byte**[1024\*64];  DatagramPacket dp = **new** DatagramPacket(buffer , buffer.length);    // 3、调用Socket的接收方法  ds.receive(dp);    //4、拆封数据  String str = **new** String(buffer,0,dp.getLength());  System.***out***.println(str);  }  }  } |

### 15.4.2 使用MulticastSocket实现多点广播

Datagram只允许数据报发送给指定的目标地址，而MulticastSocket可以将数据报以广播方式发送到数量不等的多个客户端。

IP协议为多点广播提供了这批特殊的IP地址，这些IP地址的范围是224.0.0.0至239.255.255.255。



MulticastSocket常用的方法：

* MulticastSocket(int port) ：创建多播套接字并将其绑定到特定端口。创建一个MulticastSocket对象后，还需要将该MulticastSocket加入到指定的多点广播地址，如果结束也需要脱离多点广播地址。
* void joinGroup(InetAddress mcastaddr) ：加入多播组。
* void leaveGroup(InetAddress mcastaddr) ：离开多播组。
* void setLoopbackMode(boolean disable) ：启用/禁用多播数据报的本地回送。true 表示禁用 LoopbackMode。

案例：群聊

|  |
| --- |
| **package** com.atguigu.udp.multi;  **import** java.io.IOException;  **import** java.net.DatagramPacket;  **import** java.net.InetAddress;  **import** java.net.MulticastSocket;  **import** java.util.Scanner;  **public** **class** Chat {  **private** **volatile** **static** **boolean** *exit* = **false**;  **private** **static** Scanner *input* = **new** Scanner(System.***in***);  **private** **static** String *username*;    **public** **static** **void** main(String[] args) **throws** IOException {  MulticastSocket socket = **new** MulticastSocket(9999);  InetAddress ip = InetAddress.*getByName*("230.0.0.1");  socket.joinGroup(ip);  socket.setLoopbackMode(**false**);    System.***out***.print("请输入用户名：");  *username* = *input*.nextLine();    SendThread s = **new** SendThread(socket,ip);  ReceiveThread r = **new** ReceiveThread(socket);    s.start();  r.start();    **try** {  s.join();  } **catch** (InterruptedException e) {  e.printStackTrace();  }    socket.close();  *input*.close();  }  **static** **class** SendThread **extends** Thread{  **private** MulticastSocket socket;  **private** InetAddress ip;    **public** SendThread(MulticastSocket socket,InetAddress ip) {  **super**();  **this**.socket = socket;  **this**.ip = ip;  }  **public** **void** run(){  **try** {  **while** (!*exit*) {  System.***out***.print("输入广播消息：");  String message = *input*.nextLine();  **if** ("bye".equals(message)) {  *exit* = **true**;  **break**;  }    **byte**[] data = (*username*+":"+message).getBytes();  DatagramPacket dp = **new** DatagramPacket(data, data.length, ip, 9999);    socket.send(dp);  }  } **catch** (IOException e) {  e.printStackTrace();  }  }    }  **static** **class** ReceiveThread **extends** Thread{  **private** MulticastSocket socket;    **public** ReceiveThread(MulticastSocket socket) {  **super**();  **this**.socket = socket;  }    **public** **void** run(){  **try** {  **while** (!*exit*) {  **byte**[] data = **new** **byte**[1024];  DatagramPacket dp = **new** DatagramPacket(data, data.length);  socket.receive(dp);  String str = **new** String(data, 0, dp.getLength());  System.***out***.println(str);  }  } **catch** (IOException e) {  *exit* = **false**;  }  }    }  } |

## 15.5 本章小结

本章重点介绍了Java网络编程的相关知识。通过本章的学习，了解了计算机网络的相关知识，并介绍了IP地址和端口、网络协议的概念，这是进行网络编程的基础。本章介绍了Java提供的InetAddress、URLConnection等，并通过它获取了远程资源。本章还介绍了ServerSocket、Socket类，程序可以通过这两个类实现了TCP服务器、TCP客户端，通过编写案例，我们发现编写基于TCP协议的网络编程对于IO流、多线程知识的应用也有了更好的体会。本章也介绍了DatagramSocket，DatagramPacket和MulticastSocket，并通过一个局域网通信案例示范了如何使用它们。