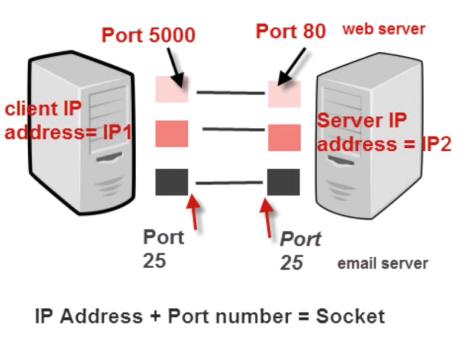
# Socket网络编程进阶与实战

## Socket网络编程入门

### 1.1Socket与TCP、UDP

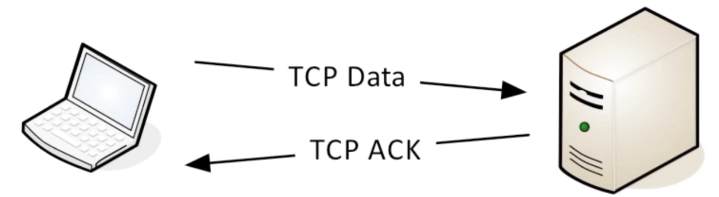


1. TCP

·TCP是面向连接的通信协议

·通过三次握手建立连接，通讯完成时要拆除连接

·由于TCP是面向连接的所以只能用于端到端的通讯

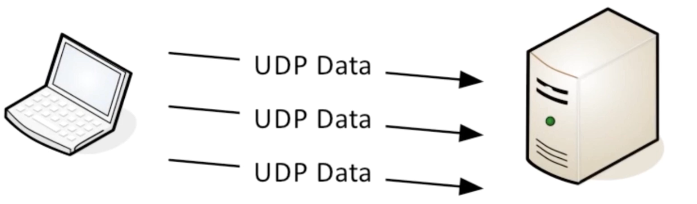


1. UDP

·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，并不局限于端到端



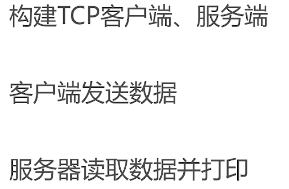
1. Client-Server 应用

·TCP/IP协议中，两个进程间通信的主要模式为：CS模型

·主要目的：协同网络中的计算机资源、服务模式、进程间数据共享

·常见的FTP、SMTP、HTTP

### 1.2Socket TCP牛刀小试



|  |
| --- |
| **public class** Client {  **public static void** main(String[] args) **throws** IOException {  Socket socket = **new** Socket();  socket.setSoTimeout(3000);  socket.connect(**new** InetSocketAddress(Inet4Address.*getLocalHost*(), 2000), 3000);  System.***out***.println(**"已发起服务器连接，并进入后续流程！"**);  System.***out***.println(**"客户端信息："** + socket.getLocalAddress() + **" P: "** + socket.getLocalPort());  System.***out***.println(**"服务端信息："** + socket.getInetAddress() + **" P: "** + socket.getPort());   **try**{  *todo*(socket);  } **catch** (Exception e){  System.***out***.println(**"异常关闭"**);  }   socket.close();  System.***out***.println(**"客户端已退出~"**);   }   **private static void** todo(Socket socket) **throws** IOException {  InputStream in = System.***in***;  BufferedReader br = **new** BufferedReader(**new** InputStreamReader(in));  OutputStream outputStream = socket.getOutputStream();  PrintStream socketPrintStream = **new** PrintStream(outputStream);   InputStream inputStream = socket.getInputStream();  BufferedReader socketBufferedReader = **new** BufferedReader(**new** InputStreamReader(inputStream));   **boolean** flag = **true**;  **do** {  *// 键盘打印一行* String str = br.readLine();  *// 发送到服务器* socketPrintStream.println(str);   *// 从服务器读取一行* String echo = socketBufferedReader.readLine();  **if**(**"bye"**.equals(echo)){  flag = **false**;  } **else** {  System.***out***.println(echo);  }  } **while** (flag);   } } |

|  |
| --- |
| **public class** Server {  **public static void** main(String[] args) **throws** IOException {  ServerSocket server = **new** ServerSocket(2000);   System.***out***.println(**"服务器准备就绪！"**);  System.***out***.println(**"服务端信息："** + server.getInetAddress() + **" P: "** + server.getLocalPort());   *// 等待客户端连接* **for** (;;){  *// 得到客户端* Socket client = server.accept();  *// 客户端构建异步线程* ClientHandler clientHandler = **new** ClientHandler(client);  *// 启动线程* clientHandler.start();  }   }   **private static void** todo(Socket client) {  }   **private static class** ClientHandler **extends** Thread{  **private** Socket **socket**;  **private boolean flag** = **true**;  ClientHandler(Socket socket){  **this**.**socket** = socket;  }   @Override  **public void** run() {  **super**.run();  System.***out***.println(**"新客户端连接："** + **socket**.getInetAddress() +  **" P: "** + **socket**.getPort());  **try**{  *// 得到打印流，用于数据输出；服务器回送数据使用* PrintStream socketOutput = **new** PrintStream(**socket**.getOutputStream());  *// 得到输入流，用于接收数据* BufferedReader socketInput = **new** BufferedReader(**new** InputStreamReader(**socket**.getInputStream()));   **do** {  *// 客户端拿到一条数据* String str = socketInput.readLine();  **if** (**"bye"**.equalsIgnoreCase(str)){  **flag** = **false**;  *// 回送* socketOutput.println(**"bye"**);  } **else** {  System.***out***.println(str);  socketOutput.println(**"回送："** + str.length());  }  } **while** (**flag**);  socketInput.close();  socketOutput.close();  } **catch** (Exception e){  System.***out***.println(**"连接异常断开"**);  } **finally** {  **try** {  **socket**.close();  } **catch** (Exception e){  e.printStackTrace();  }  }   System.***out***.println(**"客户端已退出："** + **socket**.getInetAddress() + **" P: "** + **socket**.getPort());  }  } } |

|  |
| --- |
| > Task :Server.main()  服务器准备就绪！  服务端信息：0.0.0.0/0.0.0.0 P: 2000  新客户端连接：/192.168.0.101 P: 56613  hello  客户端已退出：/192.168.0.101 P: 56613  新客户端连接：/192.168.0.101 P: 56652  hi~!  客户端已退出：/192.168.0.101 P: 56652 |

|  |
| --- |
| > Task :Client.main()  已发起服务器连接，并进入后续流程！  客户端信息：/192.168.0.101 P: 56652  服务端信息：BYF-PC/192.168.0.101 P: 2000  hi~!  回送：4  bye  客户端已退出~ |

### 1.3报文、协议、Mac地址

1. 报文段

·报文段是指TCP/IP协议网络传输过程中，起着路由导航作用

·用以查询各个网络路由网段、IP地址、交换协议等IP数据包

·报文段充当整个TCP/IP协议数据包的导航路由功能

·报文在传输过程中会不断地封装成分组、包、帧来传输

·封装方式就是添加一些控制信息组成的首部，即报文头

1. 传输协议

·一种约束，规定

·约定大于配置，在网络传输中依然使用；网络的传输流程是健壮的稳定的，得益于基础的协议构成

·简单的说：A->B的传输数据，B能识别，繁殖b->A的传输数据A也能识别，这就是协议

1. Mac地址

·媒体访问控制，物理地址、硬件地址；

·定义网络设备的位置

### 1.4 IP、端口、远程服务器

（1）IP地址

·互联网协议地址

·是分配给网络上使用网际协议的设备的数字标签

·常见的IP地址分为IPv4与IPv6两大类

·IP地址由32位二进制数组成，每组十进制值<= 255

·分为A、B、C、D、E五大类，其中E类属于特殊保留地址

·总数量：42亿，最终于2011年2月3日用尽

·如果主机号全是1，那么这个地址为直接广播地址

·IP地址255.255.255.255为受限广播地址

IPv6：

·总长度128位, 32个十六进制数，也可以想象成1632个；

·由两个逻辑部分组成：一个64位网络前缀和一个64位主机地址，主机地址通常根据物理地址自动生成，叫做EUI-64

·8组，每组4个16进制数

·IPv4可以转换成IPv6，IPv6不可以转换成IPv4

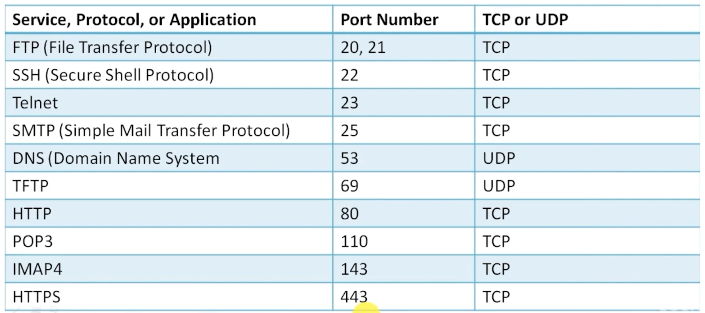
1. 端口

·如果IP地址比作房子，端口就是出入这个房子的窗口

·在不同门窗后有不同的人，房子中的用户与外界交流的出口

·外界鸽子（信息）飞到不同窗口也就给不同的人传递信息

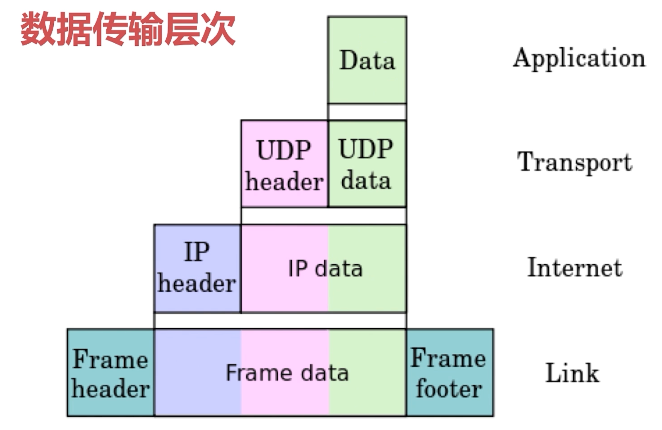
·0-1023号端口以及1024到49151号端口都是特殊端口



·计算机之间依照互联网传输层TCP/IP协议的协议通信，不同的协议都对应不同的端口

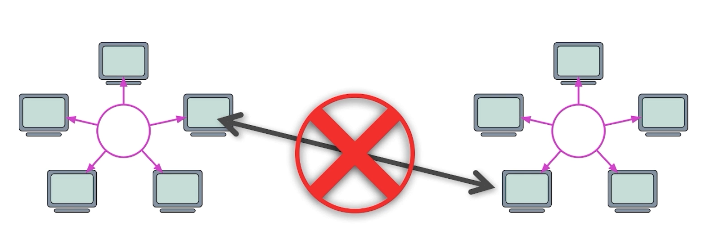
·49152到65535号端口属于“动态端口”范围，没有端口可以被正式地注册占用

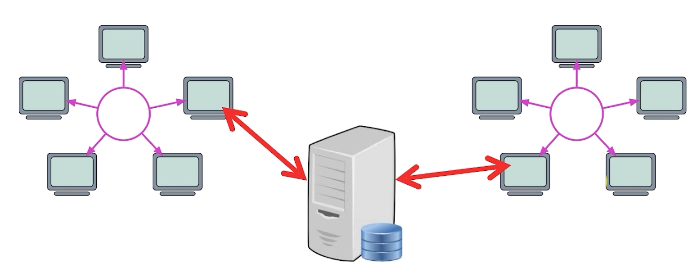
·问题：端口总数：65536，连接能建立多少个？根据操作系统能打开的句柄数决定

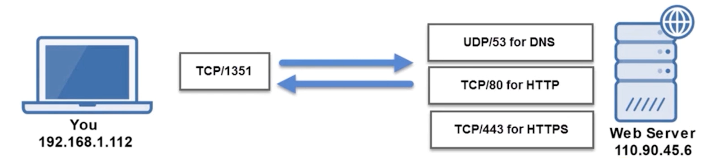


1. 远程服务器

·局域网：一般而言，家里的环境以及公司相互电脑之间环境都属于局域网







## 第2节Socket-UDP入门

### 2.1 UDP是什么

·User Datagram Protocol

·用户数据报协议

·面向数据报的传输层协议，正式规范：RFC768

·用户数据协议、非连接协议

为什么不可靠？

·一旦发送给网络层，就不保留数据备份

·UDP在IP数据报的头部仅仅加入了复用和数据校验（字段）

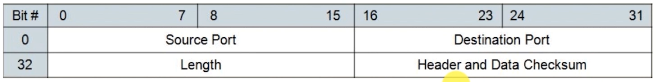
·发送端产生数据，接收端从网络中抓取数据

·结构简单、无校验、速度快、容易丢包、可广播

UDP能做什么？

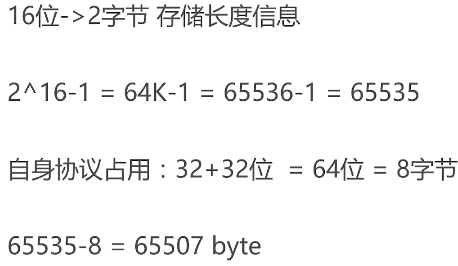
·DNS、TFTP、SNMP

·视频、音频、普通数据（无关紧要的数据）



0-15位：2个字节0-65535

UDP包最大传输长度？



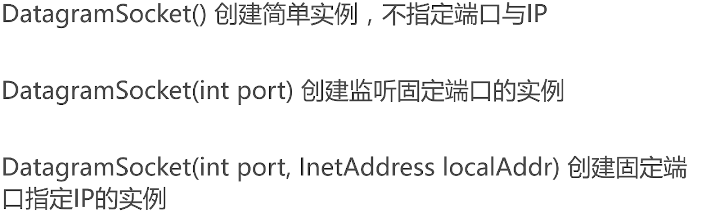
### 2.2 UDP核心API

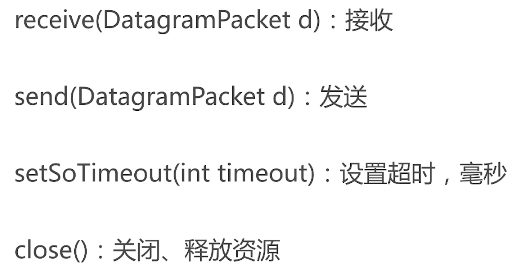
（1）API-DatagramSocket

·用于接收与发送UDP的类

·负责发送某一个UDP包，或者接收UDP包

·不同于TCP， UDP并没有合并到Socket API中



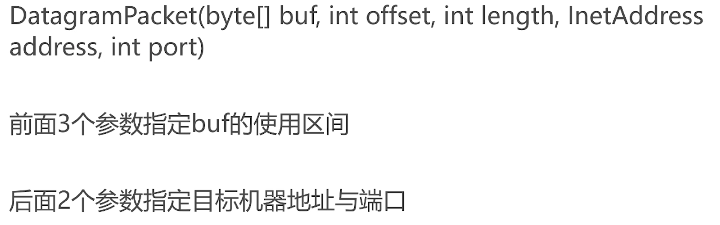


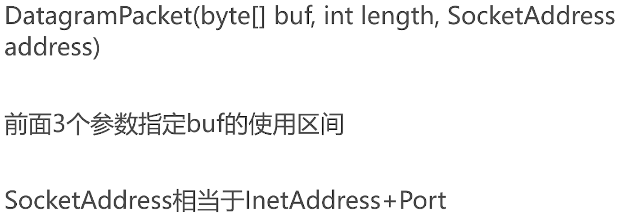
1. API-DatagramPacket

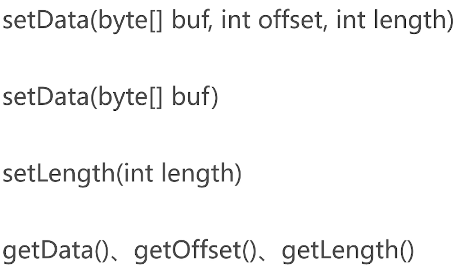
·用于报文处理

·将byte数组、（目标地址）、目标端口等数据包装成报文或者将报文或者将报文拆卸成byte数组

·是UDP的发送实体、也是接收实体

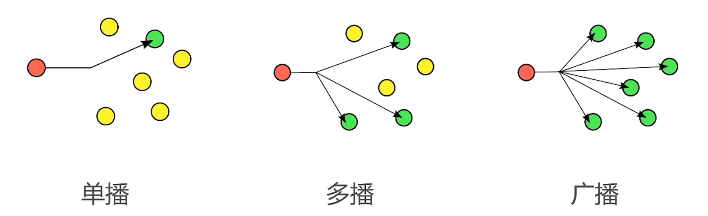




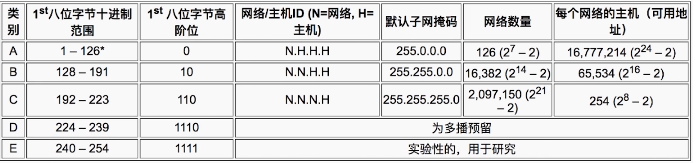




### 2.3 UDP单播、广播、多播



IP地址类别

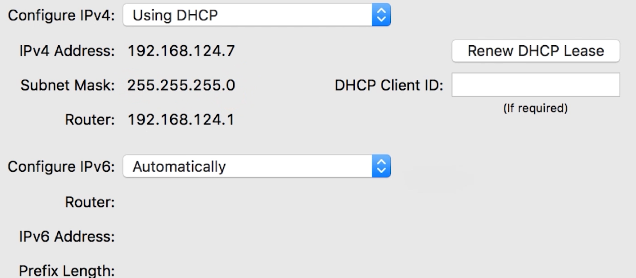


广播地址

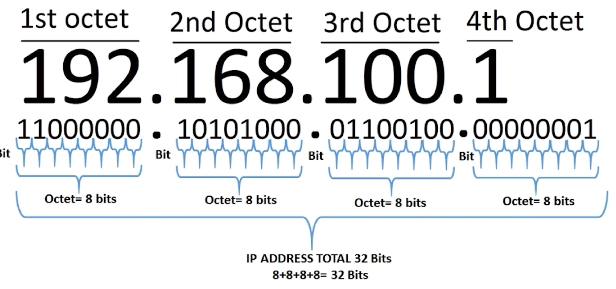
·255.255.255.255为受限广播地址

·C网广播地址一般为：XXX.XXX.XXX.255（192.168.1.255）

·D类IP地址为多播预留



IP地址构成

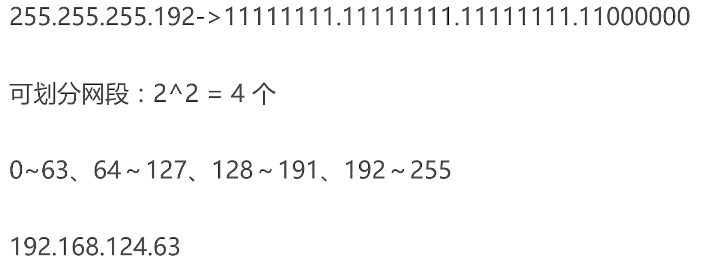


广播地址运算





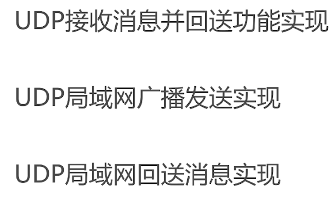
·通过子网掩码计算可划分网段



·掩码下的两个主机地址，计算能否直接通信



### 2.4案例实操-局域网搜索案例



|  |
| --- |
| **public class** UDPProvider {  **public static void** main(String[] args) **throws** IOException {  System.***out***.println(**"UDPProvider Started."**);   *// 作为接收者，指定一个端口用于数据接收* DatagramSocket ds = **new** DatagramSocket(20000);  *//构建接收实体* **final byte**[] buf = **new byte**[512];  DatagramPacket receivePack = **new** DatagramPacket(buf, buf.**length**);   *//接收* ds.receive(receivePack);   *//打印接收到的信息与发送者的信息  //发送者的IP地址* String ip = receivePack.getAddress().getHostAddress();  **int** port = receivePack.getPort();  **int** dataLen = receivePack.getLength();  String data = **new** String(receivePack.getData(),0,dataLen);  System.***out***.println(**"UDPProvider receive from ip: "** + ip + **"\tport: "** + port  + **"\tdata: "** + data);  *// 构建一份回送数据* String responseData = **"Receive data with len: "** + dataLen;  **byte**[] responseDataBytes = responseData.getBytes();  *// 直接根据发送者构建一份回送信息* DatagramPacket responsePacket = **new** DatagramPacket(responseDataBytes,  responseDataBytes.**length**,  receivePack.getAddress(),  receivePack.getPort());  ds.send(responsePacket);   *// 完成* System.***out***.println(**"UDPProvider Finished."**);  ds.close();  } } |

|  |
| --- |
| **public class** UDPSearcher {  **public static void** main(String[] args) **throws** IOException {  System.***out***.println(**"UDPProvider Started."**);   *// 作为搜索方，指定一个端口用于数据接收* DatagramSocket ds = **new** DatagramSocket();   *// 构建一份回送数据* String requestData = **"Hello World."**;  **byte**[] requestDataBytes = requestData.getBytes();  *// 直接根据发送者构建一份回送信息* DatagramPacket requestPacket = **new** DatagramPacket(requestDataBytes,  requestDataBytes.**length**);  requestPacket.setAddress(InetAddress.*getLocalHost*());  requestPacket.setPort(20000);  ds.send(requestPacket);   *//构建接收实体* **final byte**[] buf = **new byte**[512];  DatagramPacket receivePack = **new** DatagramPacket(buf, buf.**length**);   *//接收* ds.receive(receivePack);   *//打印接收到的信息与发送者的信息  //发送者的IP地址* String ip = receivePack.getAddress().getHostAddress();  **int** port = receivePack.getPort();  **int** dataLen = receivePack.getLength();  String data = **new** String(receivePack.getData(),0,dataLen);  System.***out***.println(**"UDPSearcher receive from ip: "** + ip + **"\tport: "** + port  + **"\tdata: "** + data);    *// 完成* System.***out***.println(**"UDPSearcher Finished."**);  ds.close();  } } |

|  |
| --- |
| > Task :UDPProvider.main()  UDPProvider Started.  UDPProvider receive from ip: 192.168.0.101 port: 62988 data: Hello World.  UDPProvider Finished. |
| > Task :UDPSearcher.main()  UDPProvider Started.  UDPSearcher receive from ip: 192.168.0.101 port: 20000 data: Receive data with len: 12  UDPSearcher Finished. |

### 2.5广播

|  |
| --- |
| **public class** MessageCreator {  **private static final** String ***SN\_HEADER*** = **"收到暗号，我是(SN): "**;  **private static final** String ***PORT\_HEADER*** = **"这是暗号，请回电端口(PORT): "**;   **public static** String buildWithPort(**int** port){  **return *PORT\_HEADER*** + port;  }   **public static int** parsePort(String data){  **if** (data.startsWith(***PORT\_HEADER***)){  **return** Integer.*parseInt*(data.substring(***PORT\_HEADER***.length()));  }  **return** -1;  }   **public static** String buildWithSn(String sn){  **return *SN\_HEADER*** +sn;  }  **public static** String parseSn(String data){  **if** (data.startsWith(***SN\_HEADER***)){  **return** data.substring(***SN\_HEADER***.length());  }  **return null**;  }  } |

|  |
| --- |
| */\*\*  \* UDP 提供者，用于提供服务  \*/* **public class** UDPProvider {   **public static void** main(String[] args) **throws** IOException {  *// 生成一份唯一标示* String sn = UUID.*randomUUID*().toString();  Provider provider = **new** Provider(sn);  provider.start();   *// 读取任意键盘信息后可以退出  //noinspection ResultOfMethodCallIgnored* System.***in***.read();  provider.exit();  }   **private static class** Provider **extends** Thread {  **private final** String **sn**;  **private boolean done** = **false**;  **private** DatagramSocket **ds** = **null**;   **public** Provider(String sn) {  **super**();  **this**.**sn** = sn;  }   @Override  **public void** run() {  **super**.run();   System.***out***.println(**"UDPProvider Started."**);   **try** {  *// 监听20000 端口* **ds** = **new** DatagramSocket(20000);   **while** (!**done**) {   *// 构建接收实体* **final byte**[] buf = **new byte**[512];  DatagramPacket receivePack = **new** DatagramPacket(buf, buf.**length**);   *// 接收* **ds**.receive(receivePack);   *// 打印接收到的信息与发送者的信息  // 发送者的IP地址* String ip = receivePack.getAddress().getHostAddress();  **int** port = receivePack.getPort();  **int** dataLen = receivePack.getLength();  String data = **new** String(receivePack.getData(), 0, dataLen);  System.***out***.println(**"UDPProvider receive form ip:"** + ip  + **"\tport:"** + port + **"\tdata:"** + data);   *// 解析端口号* **int** responsePort = MessageCreator.*parsePort*(data);  **if** (responsePort != -1) {  *// 构建一份回送数据* String responseData = MessageCreator.*buildWithSn*(**sn**);  **byte**[] responseDataBytes = responseData.getBytes();  *// 直接根据发送者构建一份回送信息* DatagramPacket responsePacket = **new** DatagramPacket(responseDataBytes,  responseDataBytes.**length**,  receivePack.getAddress(),  responsePort);   **ds**.send(responsePacket);  }   }   } **catch** (Exception ignored) {  } **finally** {  close();  }   *// 完成* System.***out***.println(**"UDPProvider Finished."**);  }    **private void** close() {  **if** (**ds** != **null**) {  **ds**.close();  **ds** = **null**;  }  }    */\*\*  \* 提供结束  \*/* **void** exit() {  **done** = **true**;  close();  }   }  } |

|  |
| --- |
| */\*\*  \* UDP 搜索者，用于搜索服务支持方  \*/* **public class** UDPSearcher {  **private static final int *LISTEN\_PORT*** = 30000;    **public static void** main(String[] args) **throws** IOException, InterruptedException {  System.***out***.println(**"UDPSearcher Started."**);   Listener listener = *listen*();  *sendBroadcast*();   *// 读取任意键盘信息后可以退出  //noinspection ResultOfMethodCallIgnored* System.***in***.read();   List<Device> devices = listener.getDevicesAndClose();   **for** (Device device : devices) {  System.***out***.println(**"Device:"** + device.toString());  }   *// 完成* System.***out***.println(**"UDPSearcher Finished."**);  }   **private static** Listener listen() **throws** InterruptedException {  System.***out***.println(**"UDPSearcher start listen."**);  CountDownLatch countDownLatch = **new** CountDownLatch(1);  Listener listener = **new** Listener(***LISTEN\_PORT***, countDownLatch);  listener.start();   countDownLatch.await();  **return** listener;  }   **private static void** sendBroadcast() **throws** IOException {  System.***out***.println(**"UDPSearcher sendBroadcast started."**);   *// 作为搜索方，让系统自动分配端口* DatagramSocket ds = **new** DatagramSocket();    *// 构建一份请求数据* String requestData = MessageCreator.*buildWithPort*(***LISTEN\_PORT***);  **byte**[] requestDataBytes = requestData.getBytes();  *// 直接构建packet* DatagramPacket requestPacket = **new** DatagramPacket(requestDataBytes,  requestDataBytes.**length**);  *// 20000端口, 广播地址* requestPacket.setAddress(InetAddress.*getByName*(**"255.255.255.255"**));  requestPacket.setPort(20000);   *// 发送* ds.send(requestPacket);  ds.close();   *// 完成* System.***out***.println(**"UDPSearcher sendBroadcast finished."**);  }   **private static class** Device {  **final int port**;  **final** String **ip**;  **final** String **sn**;   **private** Device(**int** port, String ip, String sn) {  **this**.**port** = port;  **this**.**ip** = ip;  **this**.**sn** = sn;  }   @Override  **public** String toString() {  **return "Device{"** +  **"port="** + **port** +  **", ip='"** + **ip** + **'\''** +  **", sn='"** + **sn** + **'\''** +  **'}'**;  }  }   **private static class** Listener **extends** Thread {  **private final int listenPort**;  **private final** CountDownLatch **countDownLatch**;  **private final** List<Device> **devices** = **new** ArrayList<>();  **private boolean done** = **false**;  **private** DatagramSocket **ds** = **null**;    **public** Listener(**int** listenPort, CountDownLatch countDownLatch) {  **super**();  **this**.**listenPort** = listenPort;  **this**.**countDownLatch** = countDownLatch;  }   @Override  **public void** run() {  **super**.run();   *// 通知已启动* **countDownLatch**.countDown();  **try** {  *// 监听回送端口* **ds** = **new** DatagramSocket(**listenPort**);    **while** (!**done**) {  *// 构建接收实体* **final byte**[] buf = **new byte**[512];  DatagramPacket receivePack = **new** DatagramPacket(buf, buf.**length**);   *// 接收* **ds**.receive(receivePack);   *// 打印接收到的信息与发送者的信息  // 发送者的IP地址* String ip = receivePack.getAddress().getHostAddress();  **int** port = receivePack.getPort();  **int** dataLen = receivePack.getLength();  String data = **new** String(receivePack.getData(), 0, dataLen);  System.***out***.println(**"UDPSearcher receive form ip:"** + ip  + **"\tport:"** + port + **"\tdata:"** + data);   String sn = MessageCreator.*parseSn*(data);  **if** (sn != **null**) {  Device device = **new** Device(port, ip, sn);  **devices**.add(device);  }  }  } **catch** (Exception ignored) {   } **finally** {  close();  }  System.***out***.println(**"UDPSearcher listener finished."**);   }   **private void** close() {  **if** (**ds** != **null**) {  **ds**.close();  **ds** = **null**;  }  }   List<Device> getDevicesAndClose() {  **done** = **true**;  close();  **return devices**;  }  } } |

|  |
| --- |
| > Task :UDPProvider.main()  UDPProvider Started.  UDPProvider receive form ip:192.168.0.101 port:59081 data:这是暗号，请回电端口(PORT): 30000  UDPProvider receive form ip:192.168.0.101 port:50008 data:这是暗号，请回电端口(PORT): 30000  exit  UDPProvider Finished. |
| > Task :UDPSearcher.main()  UDPSearcher Started.  UDPSearcher start listen.  UDPSearcher sendBroadcast started.  UDPSearcher sendBroadcast finished.  UDPSearcher receive form ip:192.168.0.101 port:20000 data:收到暗号，我是(SN): c73ab30c-4809-4157-adf0-2e32abc98455  exit  UDPSearcher listener finished.  Device:Device{port=20000, ip='192.168.0.101', sn='c73ab30c-4809-4157-adf0-2e32abc98455'}  UDPSearcher Finished. |

## 第3节Socket-TCP入门

### 3.1TCP是什么



·TCP机制

·三次握手建立、四次挥手关闭

·具有校验机制、可靠、数据传输稳定

·TCP可以做

·聊天消息传输、推送

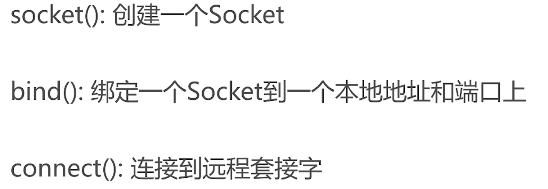
·单人语音、视频聊天等

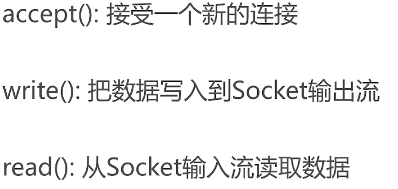
·几乎UDP能做的都能做，考虑复杂性、性能

·限制：无法进行广播，多播操作

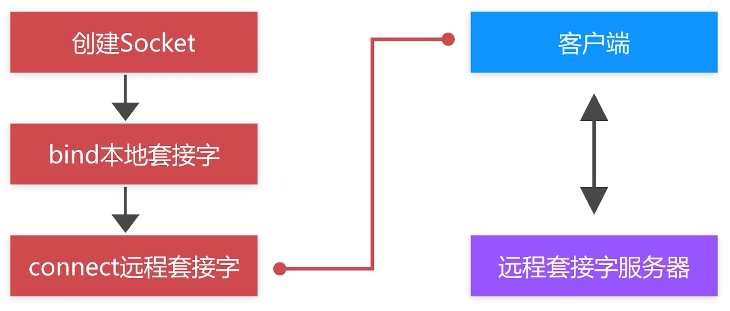
·搜索：UDP优越

### 3.2TCP核心API

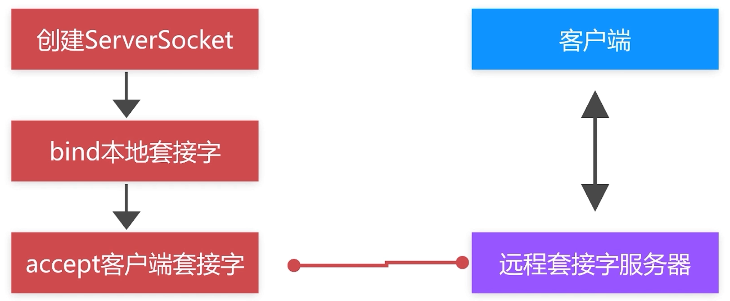




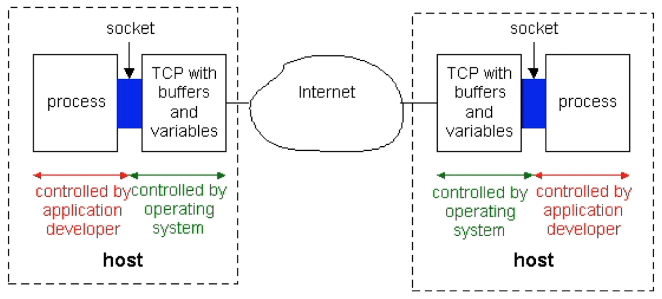
·客户端



·服务端

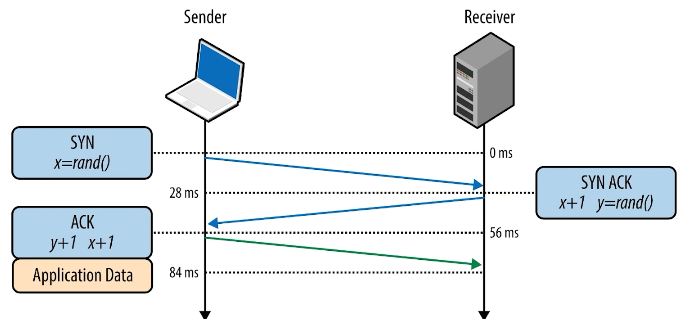


·Socket与进程关系

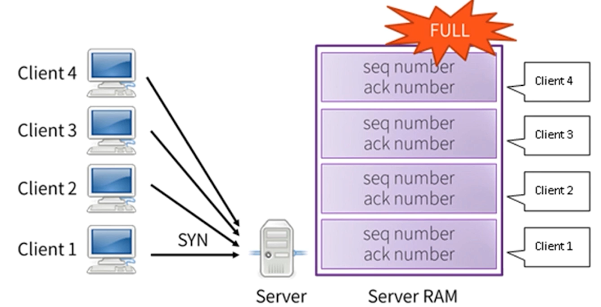


### 3.3TCP可靠性-三次握手、四次挥手

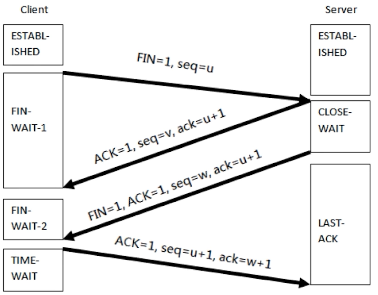
·连接可靠性



·三次握手--数据随机的必要性



·四次挥手

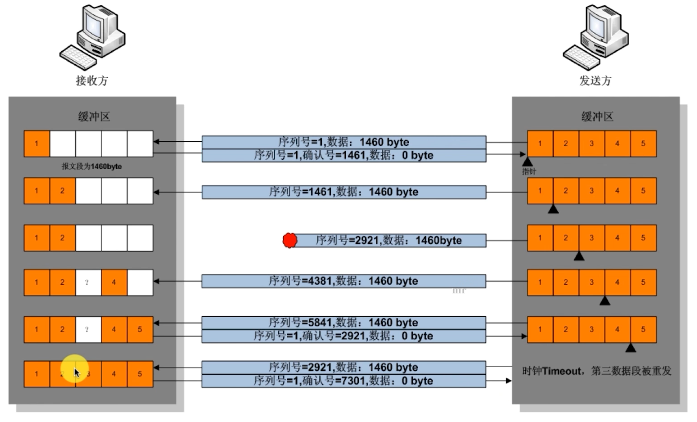


### 3.4TCP传输可靠性

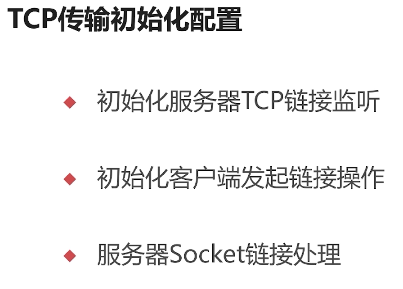
·排序、顺序发送、顺序组装

·丢弃、超时

·重发机制--定时器



### 3.5TCP传输初始化配置



|  |
| --- |
| **import** java.io.IOException; **import** java.io.InputStream; **import** java.io.OutputStream; **import** java.net.Inet4Address; **import** java.net.InetSocketAddress; **import** java.net.ServerSocket; **import** java.net.Socket; **import** java.nio.ByteBuffer;  **public class** Server {  **private static final int *PORT*** = 20000;   **public static void** main(String[] args) **throws** IOException {  ServerSocket server = *createServerSocket*();   *initServerSocket*(server);   *// 绑定到本地端口上* server.bind(**new** InetSocketAddress(Inet4Address.*getLocalHost*(), ***PORT***), 50);    System.***out***.println(**"服务器准备就绪～"**);  System.***out***.println(**"服务器信息："** + server.getInetAddress() + **" P:"** + server.getLocalPort());    *// 等待客户端连接* **for** (; ; ) {  *// 得到客户端* Socket client = server.accept();  *// 客户端构建异步线程* ClientHandler clientHandler = **new** ClientHandler(client);  *// 启动线程* clientHandler.start();  }   }   **private static** ServerSocket createServerSocket() **throws** IOException {  *// 创建基础的ServerSocket* ServerSocket serverSocket = **new** ServerSocket();   *// 绑定到本地端口20000上，并且设置当前可允许等待链接的队列为50个  //serverSocket = new ServerSocket(PORT);   // 等效于上面的方案，队列设置为50个  //serverSocket = new ServerSocket(PORT, 50);   // 与上面等同  // serverSocket = new ServerSocket(PORT, 50, Inet4Address.getLocalHost());* **return** serverSocket;  }   **private static void** initServerSocket(ServerSocket serverSocket) **throws** IOException {  *// 是否复用未完全关闭的地址端口* serverSocket.setReuseAddress(**true**);   *// 等效Socket#setReceiveBufferSize* serverSocket.setReceiveBufferSize(64 \* 1024 \* 1024);   *// 设置serverSocket#accept超时时间  // serverSocket.setSoTimeout(2000);   // 设置性能参数：短链接，延迟，带宽的相对重要性* serverSocket.setPerformancePreferences(1, 1, 1);  }   */\*\*  \* 客户端消息处理  \*/* **private static class** ClientHandler **extends** Thread {  **private** Socket **socket**;   ClientHandler(Socket socket) {  **this**.**socket** = socket;  }   @Override  **public void** run() {  **super**.run();  System.***out***.println(**"新客户端连接："** + **socket**.getInetAddress() +  **" P:"** + **socket**.getPort());   **try** {  *// 得到套接字流* OutputStream outputStream = **socket**.getOutputStream();  InputStream inputStream = **socket**.getInputStream();   **byte**[] buffer = **new byte**[256];  **int** readCount = inputStream.read(buffer);  ByteBuffer byteBuffer = ByteBuffer.*wrap*(buffer, 0, readCount);   *// byte* **byte** be = byteBuffer.get();   *// char* **char** c = byteBuffer.getChar();   *// int* **int** i = byteBuffer.getInt();   *// bool* **boolean** b = byteBuffer.get() == 1;   *// Long* **long** l = byteBuffer.getLong();   *// float* **float** f = byteBuffer.getFloat();   *// double* **double** d = byteBuffer.getDouble();   *// String* **int** pos = byteBuffer.position();  String str = **new** String(buffer, pos, readCount - pos - 1);   System.***out***.println(**"收到数量："** + readCount + **" 数据："** + be + **"\n"** + c + **"\n"** + i + **"\n"** + b + **"\n"** + l + **"\n"** + f + **"\n"** + d + **"\n"** + str + **"\n"**);   outputStream.write(buffer, 0, readCount);  outputStream.close();  inputStream.close();   } **catch** (Exception e) {  System.***out***.println(**"连接异常断开"**);  } **finally** {  *// 连接关闭* **try** {  **socket**.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }   System.***out***.println(**"客户端已退出："** + **socket**.getInetAddress() +  **" P:"** + **socket**.getPort());   }  } } |
| **import** java.io.IOException; **import** java.io.InputStream; **import** java.io.OutputStream; **import** java.net.Inet4Address; **import** java.net.InetSocketAddress; **import** java.net.Socket; **import** java.net.SocketException; **import** java.nio.ByteBuffer;  **public class** Client {  **private static final int *PORT*** = 20000;  **private static final int *LOCAL\_PORT*** = 20001;   **public static void** main(String[] args) **throws** IOException {  Socket socket = *createSocket*();   *initSocket*(socket);   *// 链接到本地20000端口，超时时间3秒，超过则抛出超时异常* socket.connect(**new** InetSocketAddress(Inet4Address.*getLocalHost*(), ***PORT***), 3000);   System.***out***.println(**"已发起服务器连接，并进入后续流程～"**);  System.***out***.println(**"客户端信息："** + socket.getLocalAddress() + **" P:"** + socket.getLocalPort());  System.***out***.println(**"服务器信息："** + socket.getInetAddress() + **" P:"** + socket.getPort());   **try** {  *// 发送接收数据  todo*(socket);  } **catch** (Exception e) {  System.***out***.println(**"异常关闭"**);  }   *// 释放资源* socket.close();  System.***out***.println(**"客户端已退出～"**);   }   **private static** Socket createSocket() **throws** IOException {  */\*  // 无代理模式，等效于空构造函数  Socket socket = new Socket(Proxy.NO\_PROXY);   // 新建一份具有HTTP代理的套接字，传输数据将通过www.baidu.com:8080端口转发  Proxy proxy = new Proxy(Proxy.Type.HTTP,  new InetSocketAddress(Inet4Address.getByName("www.baidu.com"), 8800));  socket = new Socket(proxy);   // 新建一个套接字，并且直接链接到本地20000的服务器上  socket = new Socket("localhost", PORT);   // 新建一个套接字，并且直接链接到本地20000的服务器上  socket = new Socket(Inet4Address.getLocalHost(), PORT);   // 新建一个套接字，并且直接链接到本地20000的服务器上，并且绑定到本地20001端口上  socket = new Socket("localhost", PORT, Inet4Address.getLocalHost(), LOCAL\_PORT);  socket = new Socket(Inet4Address.getLocalHost(), PORT, Inet4Address.getLocalHost(), LOCAL\_PORT);  \*/* Socket socket = **new** Socket();  *// 绑定到本地20001端口* socket.bind(**new** InetSocketAddress(Inet4Address.*getLocalHost*(), ***LOCAL\_PORT***));   **return** socket;  }   **private static void** initSocket(Socket socket) **throws** SocketException {  *// 设置读取超时时间为2秒* socket.setSoTimeout(2000);   *// 是否复用未完全关闭的Socket地址，对于指定bind操作后的套接字有效* socket.setReuseAddress(**true**);   *// 是否开启Nagle算法* socket.setTcpNoDelay(**true**);   *// 是否需要在长时无数据响应时发送确认数据（类似心跳包），时间大约为2小时* socket.setKeepAlive(**true**);   *// 对于close关闭操作行为进行怎样的处理；默认为false，0  // false、0：默认情况，关闭时立即返回，底层系统接管输出流，将缓冲区内的数据发送完成  // true、0：关闭时立即返回，缓冲区数据抛弃，直接发送RST结束命令到对方，并无需经过2MSL等待  // true、200：关闭时最长阻塞200毫秒，随后按第二情况处理* socket.setSoLinger(**true**, 20);   *// 是否让紧急数据内敛，默认false；紧急数据通过 socket.sendUrgentData(1);发送* socket.setOOBInline(**true**);   *// 设置接收发送缓冲器大小* socket.setReceiveBufferSize(64 \* 1024 \* 1024);  socket.setSendBufferSize(64 \* 1024 \* 1024);   *// 设置性能参数：短链接，延迟，带宽的相对重要性* socket.setPerformancePreferences(1, 1, 0);  }   **private static void** todo(Socket client) **throws** IOException {  *// 得到Socket输出流* OutputStream outputStream = client.getOutputStream();    *// 得到Socket输入流* InputStream inputStream = client.getInputStream();  **byte**[] buffer = **new byte**[256];  ByteBuffer byteBuffer = ByteBuffer.*wrap*(buffer);   *// byte* byteBuffer.put((**byte**) 126);   *// char* **char** c = **'a'**;  byteBuffer.putChar(c);   *// int* **int** i = 2323123;  byteBuffer.putInt(i);   *// bool* **boolean** b = **true**;  byteBuffer.put(b ? (**byte**) 1 : (**byte**) 0);   *// Long* **long** l = 298789739;  byteBuffer.putLong(l);    *// float* **float** f = 12.345f;  byteBuffer.putFloat(f);    *// double* **double** d = 13.31241248782973;  byteBuffer.putDouble(d);   *// String* String str = **"Hello你好！"**;  byteBuffer.put(str.getBytes());   *// 发送到服务器* outputStream.write(buffer, 0, byteBuffer.position() + 1);   *// 接收服务器返回* **int** read = inputStream.read(buffer);  System.***out***.println(**"收到数量："** + read);   *// 资源释放* outputStream.close();  inputStream.close();  } } |
| **public class** Tools {  **public static int** byteArrayToInt(**byte**[] b) {  **return** b[3] & 0xFF |  (b[2] & 0xFF) << 8 |  (b[1] & 0xFF) << 16 |  (b[0] & 0xFF) << 24;  }   **public static byte**[] intToByteArray(**int** a) {  **return new byte**[]{  (**byte**) ((a >> 24) & 0xFF),  (**byte**) ((a >> 16) & 0xFF),  (**byte**) ((a >> 8) & 0xFF),  (**byte**) (a & 0xFF)  };  } } |

|  |
| --- |
| 服务器准备就绪～  服务器信息：BYF-PC/192.168.0.103 P:20000  新客户端连接：/192.168.0.103 P:20001  收到数量：43 数据：126  a  2323123  true  298789739  12.345  13.31241248782973  Hello你好！  客户端已退出：/192.168.0.103 P:20001 |
| 已发起服务器连接，并进入后续流程～  客户端信息：/192.168.0.103 P:20001  服务器信息：BYF-PC/192.168.0.103 P:20000  收到数量：43  客户端已退出～ |

·TCP基础数据传输



## 第4节UDP辅助TCP实现点对点传输案例

### 4.1UDP搜索IP和端口

