# IP地址类InetAddress

## 1.1定义

Class InetAddress

java.lang.Object

java.net.InetAddress

All Implemented Interfaces:

Serializable

直接子类:

Inet4Address, Inet6Address

常用的方法:

|  |  |
| --- | --- |
| public static InetAddress[] getAllByName(String host)  throws UnknownHostException |  |
| public static InetAddress getByName(String host)  throws UnknownHostException | 给定字符串形式的ip地址或者主机名来创建一个ip地址对象 |
| public static InetAddress getLocalHost()  throws UnknownHostException | 返回本机的IP地址对象 |
| public String getHostAddress() | 返回本机的ip地址字符串的表现形式 |
| public String getHostName() | 返回计算机的名字(有可能还是ip地址) |

范例:

|  |
| --- |
| public class test {  public static void main(String args[]) throws Exception {  InetAddress addresss[] = InetAddress.**getAllByName**("www.baidu.com");  System.out.println("地址共有:" + addresss.length);  InetAddress address = InetAddress.**getLocalHost**();  System.out.println("本机ip地址:" + address.**getHostAddress**());  System.out.println("本机名称:" + address.**getHostName**());  }  } |

# 2.udp通讯

udp协议的服务类

Class DatagramSocket

All Implemented Interfaces:

Closeable, AutoCloseable

直接子类: MulticastSocket

|  |  |
| --- | --- |
| public DatagramSocket()  throws SocketException | 无参构造函数 |
| public DatagramSocket(int port)  throws SocketException | 构造函数 |
| public void send(DatagramPacket p)  throws IOException | 发送一个数据包 |
| public void close() | 关闭资源,释放端口号 |
| public void receive(DatagramPacket p)  throws IOException | 接受发送端发来的数据包 |

udp数据包类

Class DatagramPacket

public final class DatagramPacket

extends Object

|  |  |
| --- | --- |
| public DatagramPacket(  byte[] buf,int length,InetAddress address,int port) | 有参构造函数  buf:当前数据的字节数组表现形式  length:字节数组的长度  address:发送的ip地址  port:发送的端口号 |
| public int getLength() | 返回接受到的数据字节数组的长度 |
| public InetAddress getAddress() | 获取发送端的ip地址对象 |

**特点:**

1. 不需要建立连接,不可靠,速度快
2. 每个数据包限制在64K中
3. udp协议是不分客户端和服务端的，只分发送端和接受端

过程:

**接受端:**

1. 建立udp的服务,并且要监听一个端口
2. 准备一个空的数据包
3. 调用udp服务,接受数据
4. 关闭资源

范例:

|  |
| --- |
| public class Receive {  public static void main(String args[]) throws Exception {  //1.建立udp的服务,并且要监听一个端口  DatagramSocket socket = new DatagramSocket(6666);  //2.准备一个空的数据包  byte buf[] = new byte[1024];  DatagramPacket packet = new DatagramPacket(buf, buf.length);  //3.调用udp服务,接受数据  socket.receive(packet); //receive是阻塞型方法  System.out.print("接受端接受到了数据:" + new String(buf,0,packet.getLength()));  //4.关闭资源  socket.close();  }  } |

**发送端:**

1. 建立udp的服务
2. 准备数据,把数据封装到数据包中
3. 调用udp服务发送数据
4. 关闭资源

|  |
| --- |
| public class Sender{  public static void main(String args[]) throws Exception{  //1.建立udp的服务  DatagramSocket socket = new DatagramSocket();  //2.准备数据,把数据封装到数据包中  String data = "这个是我的第一个udp例子";  byte buf[] = data.getBytes();  DatagramPacket packet = new DatagramPacket(buf,buf.length, InetAddress.getLocalHost(),6666);  //3.调用udp服务发送数据  socket.send(packet);  //4.关闭资源  socket.close();  }  } |

注意:先运行接受端,再运行发送端

# 3.tcp通讯

tcp特点:

1. 面向连接,在数据传输前一定要建立连接,tcp的客户端一旦建立,马上就要与服务端建立连接
2. 在连接中可以传输大量数据,tcp是基于IO流传输数据的,udp是基于数据包传输数据的
3. 通过三次握手机制连接机制,可靠,但是比udp慢
4. tcp协议是分客户端和服务端的

客户端类:

Class Socket

All Implemented Interfaces:

Closeable, AutoCloseable

直接子类:SSLSocket

|  |  |
| --- | --- |
| public Socket(InetAddress address,int port)  throws IOException | 构造函数 |
| public OutputStream getOutputStream()  throws IOException | 返回此套接字的输出流 |
| public void close()  throws IOException | 关闭资源 |
| public InputStream getInputStream() throws IOException | 返回此套接字的输入流 |

服务端类:

Class ServerSockett

All Implemented Interfaces:

Closeable, AutoCloseable

直接子类: SSLServerSocket

|  |  |
| --- | --- |
| public ServerSocket(int port)  throws IOException | 构造函数 |
| public Socket accept()  throws IOException | 接受客户端的连接,是一个阻塞型的方法,没有客户端连接的时候会一直等待下去 |

**过程:**

**客户端:**

1. 建立tcp的客户端服务
2. 准备数据,获取对应的流通道
3. 把数据写出
4. 关闭资源

**服务端:**

1. 建立tcp服务端
2. .接受客户端的连接,accept阻塞型的方法
3. 获取socket对应的流通道
4. 通过输入流通道读取数据
5. 关闭资源

范例:客户端:

|  |
| --- |
| public class Client {  public static void main(String args[]) throws Exception {  //1.建立tcp的客户端服务  Socket socket = new Socket(InetAddress.getLocalHost(), 6666);  //2.准备数据,获取对应的流通道  String data = "你好,hello,world";  OutputStream out = socket.getOutputStream();  //3.把数据写出  out.write(data.getBytes());  //接受服务端发来的数据  InputStream in = socket.getInputStream();  byte buf[] = new byte[1024];  int length = in.read(buf);  System.out.println("服务端发来的数据:"+new String(buf,0,length));  //4.关闭资源  socket.close();  //out.close();关闭socket就可以了,out会自动关闭的  }  } |

范例:服务端

|  |
| --- |
| public class Server{  public static void main(String args[])throws Exception{  //1.建立tcp服务端  ServerSocket serversocket = new ServerSocket(6666);  //2.接受客户端的连接,accept阻塞型的方法  Socket socket = serversocket.accept();  //3.获取socket对应的流通道  InputStream in = socket.getInputStream();  //4.通过输入流通道读取数据  byte buf[] = new byte[1024];  int length = in.read(buf);  System.out.println("服务端接受到的数据:" + new String(buf,0,length));  //服务端给客户端会送数据  //获取socket输出流  OutputStream out = socket.getOutputStream();  out.write("客户端你幸苦了".getBytes());  //5.关闭资源  serversocket.close();  }  } |

# 4.多线程tcp通讯

服务端:

|  |
| --- |
| //需求:编写一个服务端可以与多个客户端进行连接,客户端一旦连接,马上  //发送一张图片过去  public class ImageServer extends Thread{  public Socket socket;  public ImageServer(Socket socket){  this.socket = socket;  }  @Override  public void run(){  try {  //3.获取socket输出流  OutputStream out = this.socket.getOutputStream();  //4.获取图片的输入流,读取图片的数据,把图片数据写入给客户端  FileInputStream fileInputStream = new FileInputStream("C:\\Users\\lw\\Pictures\\archlinux\\1.png");  byte buf[] = new byte[1024];  int length = 0;  while ((length = fileInputStream.read(buf)) != -1) {  out.write(buf, 0, length);  }  fileInputStream.close();  this.socket.close();  }catch (Exception e){  e.printStackTrace();  }  }  public static void main(String args[]) throws Exception{  //1.建立tcp服务端  ServerSocket serverSocket = new ServerSocket(9090);  while(true){  Socket socket = serverSocket.accept(); //不断接受用户请求连接  //如果产生了一个socket就意味着有一个用户与服务端连接了  //那么马上开启一个线程为其服务  new ImageServer(socket).start();  }  }  } |

客户端:

|  |
| --- |
| //客户端  public class Client{  public static void main(String args[]) throws Exception{  //1.准备一个客户端tcp服务  Socket socket = new Socket(InetAddress.getLocalHost(),9090);  byte buf[] = new byte[1024];  OutputStream fileoutputStream = new FileOutputStream("F:\\1.png");  InputStream in = socket.getInputStream();  while(in.read(buf)!=-1){  fileoutputStream.write(buf);  }  }  } |