网络编程

一、基本知识

• IPv4地址：是一个32位的整数，通常以4个255以内的数字表示（X.X.X.X），用于唯一标识网络中的硬件设备

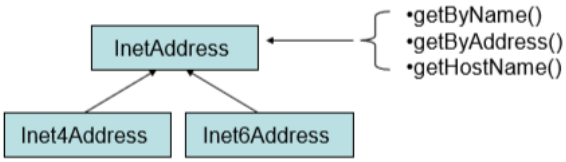
• 域名：为方便记忆和使用，用类似www.XXX.com 的字符串代替IP地址输入，靠DNS服务进行解析
localhost 127.0.0.1

• 端口号：是一个标记机器的逻辑通信信道数。端 口号是用一个16位的整数来表达的，其范围为0 ~ 65535，其中0～1023为系统所保留。 （和《计算机网络》连接在一起咯）

使用Java进行网络编程时，由jvm虚拟机实现了底层复杂的网络协议，Java程序只需要调用Java标准库提供的**接口**，就可以简单高效地编写网络程序。

补充：InetAddress地址类

> java.net.*



方法摘要：

byte[]	getAddress() 返回此 InetAddress 对象的原始 IP 地址。
static InetAddress[]	getAllByName(String host) 在给定主机名的情况下，根据系统上配置的名称服务返回其 IP 地址所组成的数组。
static InetAddress	getByAddress(byte[] addr) 在给定原始 IP 地址的情况下，返回 InetAddress 对象。
static InetAddress	getByAddress(String host, byte[] addr) 根据提供的主机名和 IP 地址创建 InetAddress。
static InetAddress	getByName(String host) 在给定主机名的情况下确定主机的 IP 地址。

例如：

```
InetAddress a = InetAddress.getByName(args[0]);
System.out.println(a);
System.out.println(a.getHostName());
```

```java
import java.net.*;

public class WhoAmI {
  public static void main(String[] args)
      throws Exception {
    if(args.length != 1) {
      System.err.println(
        "Usage: WhoAmI MachineName");
      System.exit(1);
```

```
        }
        InetAddress a = InetAddress.getByName(args[0]);//通过主机名获得IP地址
        System.out.println(a);
        System.out.println(a.getHostName());//获取地址对象的主机名
    }
}
```

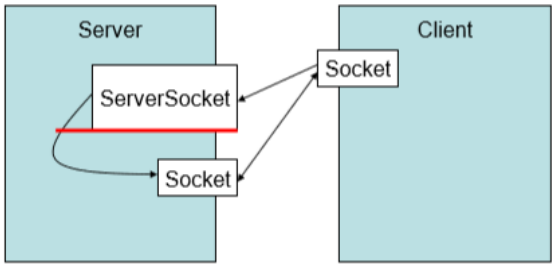
```
E:\学习PPT\大三上课件\java\网络例子>java WhoAmI localhost
localhost/127.0.0.1
localhost
```

二、TCP编程

（一）基本概念

TCP协议工作于传输层，是端到端的连接通信。TCP协议面向连接，具有可靠性和有序性，并且以字节流的方式发送数据，通常被称为流通信协议。(以下主要介绍服务器和客户端模式)

在Java中，基于TCP协议实现网络通信的类有两个：在客户端的Socket类与在服务器端的ServerSocket类



（二）ServerSocket类

> java.net.*

服务于服务器端。ServerSocket的作用是等待网络连接请求，并构造本地Socket与远程Socket通信。

ServerSocket只监听本地端口，通过accept()方法等待接入要求，在没有接入要求时，程序处于阻塞状态，不会往下执行。一旦有连入者，连接生成Socket.

构造方法：

```
ServerSocket(int port)
          创建绑定到特定端口的服务器套接字。
```

```
ServerSocket(int port, int backlog, InetAddress bindAddr)
          使用指定的端口、侦听 backlog 和要绑定到的本地 IP 地址创建服务器。
```

方法：

Socket	accept() 侦听并接受到此套接字的连接。
void	bind(SocketAddress endpoint) 将 ServerSocket 绑定到特定地址（IP 地址和端口号）。
void	bind(SocketAddress endpoint, int backlog) 将 ServerSocket 绑定到特定地址（IP 地址和端口号）。
void	close() 关闭此套接字。

- ServerSocket ss = new ServerSocket(port);
- Socket s = ss.accept();

（三）套接字Socket

> java.net.*

套接字Socket类服务于客户端。

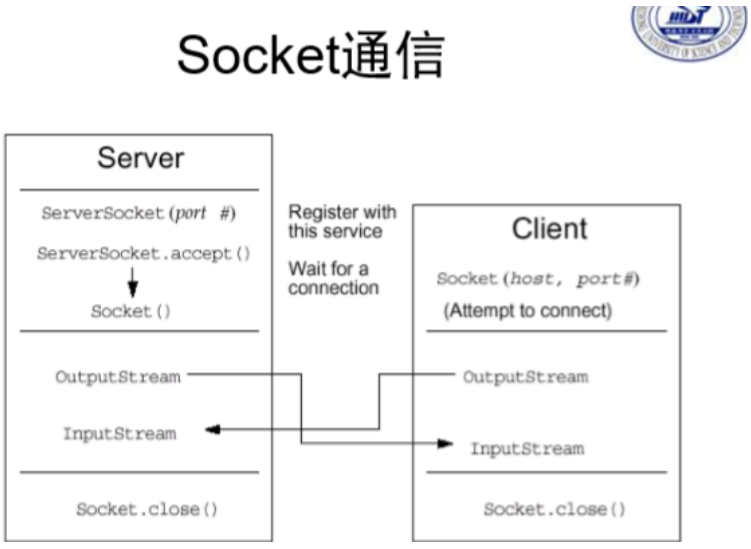
构造方法：

	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。

方法：

- Socket的输入/输出流管理
  - public InputStream getInputStream()
  - public OutputStream getOutputStream()
  - 这些方法都将抛出例外IOException，程序中需要捕获处理。
- 关闭Socket
  - public void close() throws IOException

注：Socket生成输入输出流（全双工通信），均需异常处理

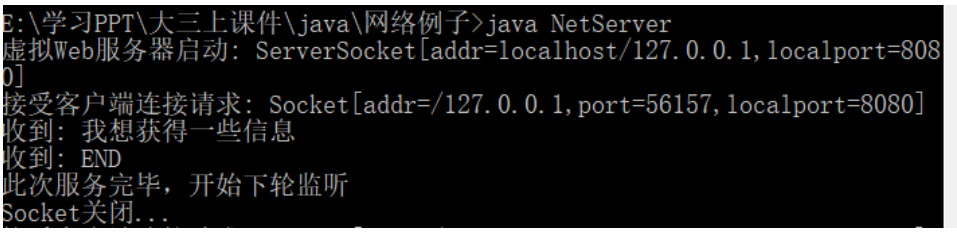


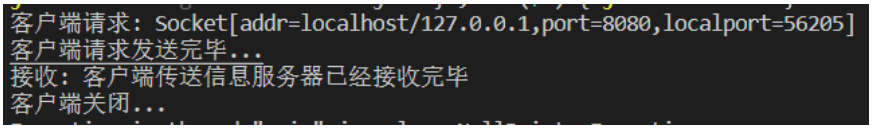
out.println()表示将信息输出到服务器

例如：

```java
/*服务器代码*/
import java.io.*;
import java.net.*;

public class NetServer {
    public static final int PORT = 8080;

    public static void main(String[] args) throws IOException {
        InetAddress addr = InetAddress.getByName("localhost");// 获得本地IP地址

        ServerSocket s = new ServerSocket(PORT, 10, addr);// 建立一个可连接的服务
        System.out.println("虚拟Web服务器启动: " + s);
        try {
            while (true) {//服务器一直在循环
                Socket socket = s.accept();// 等待连接
                try {
                    System.out.println("接受客户端连接请求: " + socket);// 只能为当前
客户端提供服务
                    BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

                    PrintWriter out = new PrintWriter(
                            new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream())), true);
                    String str = in.readLine();//读取
                    System.out.println("收到: " + str);
                    while (str != null && !str.equals("END")) {// 收到end就跳出循环
                        str = in.readLine();
                        System.out.println("收到: " + str);
                    }
                    out.println("客户端传送信息服务器已经接收完毕");//将信息发给客户端

                    System.out.println("此次服务完毕，开始下轮监听");

                } finally {
                    System.out.println("Socket关闭...");
                    socket.close();//关闭Socket套接字
                }
            } // end while
        } finally {
            s.close();// 关闭ServerSocket套接字
        }
    }
}

/*客户端代码*/
import java.io.*;
import java.net.*;

public class NetClient {
    public static final int PORT = 8080;

    public static void main(String[] args) throws IOException {

        InetAddress addr = InetAddress.getByName("localhost");

        Socket socket = new Socket(addr, PORT);
```

```
        try {
            System.out.println("客户端请求: " + socket);

            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

            PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream())),
                    true);
            out.println("我想获得一些信息");//将信息发给服务端
            out.println("END");
            String str;
            System.out.println("客户端请求发送完毕...");
            while ((str = in.readLine()).length() != 0) {
                System.out.println("接收: " + str);
            }
        } finally

        {
            System.out.println("客户端关闭...");
            socket.close();
        }
    }
}
```

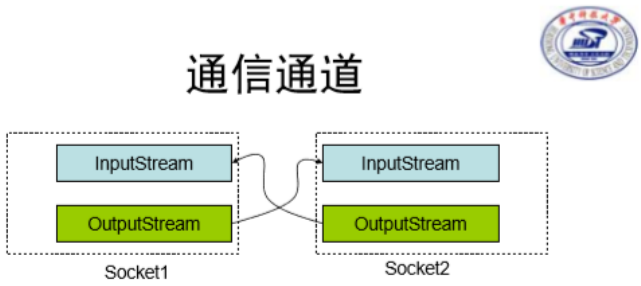





```
/*服务器 扩展板*/
            String str = in.readLine();//读套接字里面的信息
                byte[] input = new byte[20];
                int i = 0;
                System.out.println("收到" + (++i) + "条: " + str);
                while (str != null && !str.equals("END")) {
                    // System.in.read(input);
                    str = in.readLine();//如果没有这个读取，服务器端就一直无法跳出这个循
环，而且要现收后打印
                    System.out.println("收到" + (++i) + "条: " + str);
                }
/*客户端*/
    for (int i = 0; i < 20; i++) {
                out.println("我想获得一些信息");
            }
//说明是全双工通信，服务器接收与客户端发送的频率差不多，那边发的这边能准确接收到，通过Socket的
输入输出流
```

```
虚拟Web服务器启动: ServerSocket[addr=localhost/127.0.0.1,localport=8080]
接受客户端连接请求: Socket[addr=/127.0.0.1,port=56648,localport=8080]
收到1条: 我想获得一些信息
收到2条: 我想获得一些信息
收到3条: 我想获得一些信息
收到4条: 我想获得一些信息
收到5条: 我想获得一些信息
收到6条: 我想获得一些信息
收到7条: 我想获得一些信息
收到8条: 我想获得一些信息
收到9条: 我想获得一些信息
收到10条: 我想获得一些信息
收到11条: 我想获得一些信息
收到12条: 我想获得一些信息
收到13条: 我想获得一些信息
收到14条: 我想获得一些信息
收到15条: 我想获得一些信息
收到16条: 我想获得一些信息
收到17条: 我想获得一些信息
收到18条: 我想获得一些信息
收到19条: 我想获得一些信息
收到20条: 我想获得一些信息
收到21条: END
此次服务完毕，开始下轮监听
Socket关闭...
```

```
客户端请求: Socket[addr=localhost/127.0.0.1,port=8080,localport=56620]
客户端请求发送完毕...
客户端关闭...
```



- 由一对OutputStream-InputStream组成一个通信通道（上述是一个全双通通道）
- 通道中没有信息时，读取阻塞
- 通道中存储信息好像也感觉不到容量限制

三、UDP编程

（一）基本概念

UDP是一种无连接的传输协议。

– 首先需要将要传输的数据定义成一个**数据传输单元**

– 在数据报中指明数据所要达到的主机地址和端口号

– 将数据传输单元发送出去

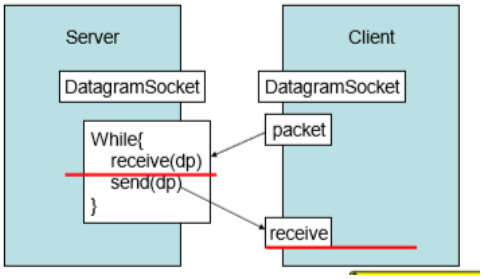
– 这与通过邮局发送邮件的情形非常相似。

• 这种传输方式是无序的，也不能确保绝对的安全可靠，但 它很**简单**也具有比较**高的效率** 。

在Java中，基于UDP协议实现网络通信的类有两个：

– 用于进行端到端通信的类 • **DatagramSocket** （数据报套接字）服务于服务器

– 用于表达通信数据的数据报类 • **DatagramPacket** （数据报文）服务于客户端



（二）DatagramSocket类

> java.net.*

服务于服务器端。DatagramSocket的作用是发送和接收数据报包的套接字。数据报套接字是包投递服务的发送或接收点。每个在数据报套接字上发送或接收的包都是单独编址和路由的。从一台机器发送到另一台机器的多个包可能选择不同的路由，也可能按不同的顺序到达。

用 UDP 广播发送。为了接收广播包，应该将 DatagramSocket 绑定到通配符地址。

构造方法：

	`DatagramSocket()` 构造数据报套接字并将其绑定到本地主机上任何可用的端口。
	`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。

方法：

void	`bind(SocketAddress addr)` 将此 DatagramSocket 绑定到特定的地址和端口。
void	`close()` 关闭此数据报套接字。
void	`connect(InetAddress address, int port)` 将套接字连接到此套接字的远程地址。
void	`connect(SocketAddress addr)` 将此套接字连接到远程套接字地址（IP 地址 + 端口号）。

（三）DatagramPacket类

> java.net.*

此类表示数据报包。

数据报包用来实现**无连接**包投递服务。每条报文仅根据该包中包含的信息从一台机器路由到另一台机器。从一台机器发送到另一台机器的多个包可能选择不同的路由，也可能按不同的顺序到达。不对包投递做出保证。

构造方法：

`DatagramPacket(byte[] buf, int length)` 构造 DatagramPacket，用来接收长度为 length 的数据包。
`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。
`DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)` 构造数据报包，用来将长度为 length 偏移量为 offset 的包发送到指定主机上的指定端口号。

方法：

InetAddress	getAddress() 返回某台机器的 IP 地址，此数据报将要发往该机器或者是从该机器接收到的。
byte[]	getData() 返回数据缓冲区。
int	getLength() 返回将要发送或接收到的数据的长度。
int	getOffset() 返回将要发送或接收到的数据的偏移量。
int	getPort() 返回某台远程主机的端口号，此数据报将要发往该主机或者是从该主机接收到的。
SocketAddress	getSocketAddress() 获取要将此包发送到的或发出此数据报的远程主机的 SocketAddress（通常为 IP 地址 + 端口号）。

还有对应的set部分

（四）通信流程

1.定义数据成员

DatagramSocket socket;

DatagramPacket packet;

InetAddress address; (用来存放接收方的地址)

int port; (用来存放接收方的端口号)

2.创建数据报文Socket对象

```
try {
      socket=new DatagramSocket(1111);
}catch(java.net.SocketException e) {.....}
```

socket 绑定到一个本地的可用端口,等待接收客户的请求.

3.分配并填写数据缓冲区(一个字节类型的数组)

byte[] Buf=new byte[256];

存放从客户端接收的请求信息.

4.创建一个DatagramPacket

packet=new DatagramPacket(buf, 256);

用来从socket接收数据,它只有两个参数

5.服务器阻塞

socket.receive(packet);

在客户的请求报道来之前一直等待

6.从到来的包中得到地址和端口号

InetAddress address=packet.getAddress();

int port=packet.getPort();

7.将数据送入缓冲区

或来自文件,或键盘输入

8.建立报文包,用来从socket上发送信息

packet=new DatagramPacket (buf,buf.length, address, port);

9.发送数据包　10.关闭socket

socket.send(packet);　socket.close();

```
/*服务端*/
import java.io.*;
import java.net.*;
import java.util.*;

public class UDPServer
{
    public static final int PORT =8081;
    public static void main(String[] args) throws IOException
    {
        //建立数据报Socket
        InetAddress addr = InetAddress.getByName("localhost");
        DatagramSocket ds =new DatagramSocket(PORT);
        System.out.println("UDP服务器启动: "+ds);
        //建立接收数据报
        byte[] buf = new byte[1000];
        DatagramPacket inDataPacket = new DatagramPacket(buf, buf.length);
        try{
            while(true) {
                //等待数据报的到来
                ds.receive(inDataPacket);
                //显示接收的数据报
                String str = new String(inDataPacket.getData(), 0,
inDataPacket.getLength());
                String rcvd = str +", from address: " +
inDataPacket.getAddress() +", port: " + inDataPacket.getPort();
                System.out.println(rcvd);
                //回应数据报
                String echoString = "Echoed: " + rcvd;
                DatagramPacket outDataPacket = new
DatagramPacket(echoString.getBytes(),echoString.length(),

inDataPacket.getAddress(), inDataPacket.getPort());
                ds.send(outDataPacket);
            }
        }catch(SocketException e){
            System.err.println("Can't open socket");
            System.exit(1);
        }catch(IOException e){
            System.err.println("Communication error");
            e.printStackTrace();
        }finally{
            ds.close();
        }
    }
}

/*客户端*/
```

```
import java.io.*;
import java.net.*;

public class UDPClient
{
    public static final int PORT =8081;
    public static void main(String[] args) throws IOException
    {
        //
        InetAddress addr = InetAddress.getByName("localhost");
        DatagramSocket datagramSocket = new DatagramSocket();
        //
        byte[] msg = new byte[100];
        DatagramPacket inDataPacket = new DatagramPacket(msg, msg.length);
        //
        DatagramPacket outDataPacket;
        String strSend = "udp request";
        outDataPacket = new DatagramPacket(strSend.getBytes(), strSend.length(),
addr, PORT);
        datagramSocket.send(outDataPacket);
        //
        datagramSocket.receive(inDataPacket);
        String receivedMsg = new String (inDataPacket.getData(), 0,
inDataPacket.getLength());
        System.out.println(receivedMsg);
        datagramSocket.close();

    }
}
```

```
E:\学习PPT\大三上课件\java\网络例子>java UDPServer
UDP服务器启动: java.net.DatagramSocket@34c45dca
udp request, from address: /127.0.0.1, port: 54914
udp request, from address: /127.0.0.1, port: 54915
```

```
E:\学习PPT\大三上课件\java\网络例子>java UDPClient
Echoed: udp request, from address: /127.0.0.1, port: 54915
```

服务器/客户机无本质差异，我们一般让服 务器while循环

```
import java.net.*;

public class WhoAmI {
  public static void main(String[] args) throws Exception {
    if (args.length != 1) {
      System.err.println("Usage: WhoAmI MachineName");
      System.exit(1);//非正常退出，异常中止
    }
    InetAddress a = InetAddress.getByName(args[0]);
    System.out.println(a);
    System.out.println(a.getHostName());
  }
}
```

```
E:\学习PPT\大三上课件\java\网络例子>java WhoAmI localhost
localhost/127.0.0.1
localhost
```