

# Java网络程序设计

主讲老师：申雪萍



2021/6/17

Xueping Shen



北京航空航天大学  
COLLEGE OF SOFTWARE  
BEIHANG UNIVERSITY 软件学院

- **网络编程的基础知识**
- 基于TCP的网络程序设计
- 基于UDP的网络程序设计
- 访问Internet网络资源

# 计算机网络工作模式

- 客户机/服务器模式(Client/Server. C/S)
  - 一台或几台较大的计算机集中进行共享数据库的管理和存取，称为服务器。而其他的应用处理工作分散到网络中其它微机上去做，构成分布式的处理系统。
- 对等式网络(Peer to Peer)
  - 每一个工作站既可以起客户机的作用，也可以起服务器的作用。
- 浏览器/服务器模式 (Browser/Server)

# 网络通信协议与接口

- 网络通信协议：
  - 计算机网络中实现通信必须有一些约定：**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，称为通信协议。**
- 网络通信接口：
  - **为了使两个结点之间能进行对话，必须在他们之间建立通信工具(即接口)，使彼此之间能进行信息交换，接口包括两部分：**
    - 硬件装置:实现结点之间的信息传递。
    - 软件装置:规定双方进行通信的约定协议。

# 网络编程的基础知识 (1)

- 通信与协议

- 网络进程之间通信必须遵循预定的规则，这些规则叫做网络进程通信协议
- TCP/IP是一组在Internet网络上的不同计算机之间进行通信的协议的总称；从下往上可视为4层结构：物理层、网络层、传输层和应用层。
- TCP/IP由应用层的HTTP、FTP、SMTP和传输层的TCP（传输控制协议）、UDP（用户数据报协议）以及网络层的IP（Internet协议）等一系列协议组成。

# 网络编程的基础知识 (2)

- 数据的封装与拆封
  - **封装**: 发送方数据在网络模型的各层传送过程中加入头尾的过程;
  - **拆封**: 接受方收到数据后去除相应的头尾的过程。

TCP、UDP的数据帧格式简单图例:

协议类型	源IP	目标IP	源端口	目标端口	帧序号	帧数据
------	-----	------	-----	------	-----	-----

其中协议类型用于区分TCP, UDP

# 什么是网络编程？

- **网络进程**就是在网络结点计算机上运行的程序。
- **网络编程**一般是指利用不同层次的通信协议提供的接口实现网络进程通信的编程。
  - Java小应用程序是属于应用层的网络编程，主要用于交互式的网页设计。
- 传输层的网络进程通信机制是进行网络编程的基础。

# 网络编程的基础知识 (3)

- 在网络上，计算机通过IP地址或主机名进行标识，这样使得位于不同地理位置的计算机有可能互相访问和通信。
- 在网络上进行通信通常要遵循一定的规则，这些规则常常称为协议。最常用的传输层的网络通信协议是TCP和UDP。
- 每台计算机都可以存放一定数量的资源，并通过网络共享。
  - 网络资源定位器 (URL, Uniform Resource Locator) 可以指向网络上的各种资源。通过网络资源定位器可以获取网络上的资源。



# 网络编程的基础知识 (4)

- IP地址和Port(端口号)
  - ip是由32位的数字表示的,通常被分成四部分。例如:192.168.0.1
  - 本地回路的IP地址: 127.0.0.1 或localhost
  - Port(端口号):逻辑意义上的数据传输通道,或者说模拟通道。
    - 计算机与网络一般只有一个物理连接,网络数据通过这个连接流向计算机。但是计算机上一般有许多进程在同时运行,为确定网络数据流向的进程,TCP/IP协议引入了端口的概念。
  - 端口同一台网络计算机的一个特定进程关联,与进程建立的套接口绑定在一起。
  - 端口号的范围为0~65535之间,0~1023之间的端口数是用于一些知名的网络服务和应用(我们一般不使用)
    - 例如:Web服务使用端口80,FTP服务使用端口21等
  - 当一个互联网上的服务使用一个非默认的ip和port号码时,是这样表示的192.168.0.2:8080

# 网络编程的基础知识 (5)

- UDP与TCP
  - **TCP**, 传输控制协议(Transmission Control Protocol), **是面向连接的通信协议**。
    - 使用TCP协议进行数据传输时,两个进程之间会建立一个连接,数据以流的形式顺序传输。
  - **UDP**, 用户数据协议(User Datagram Protocol), **是无连接通信协议**。
    - 使用UDP协议进行数据传输时,两个进程之间不建立特定的连接,不对数据到达的顺序进行检查。
- 在互联网上进行数据传输,多用TCP和UDP协议,它们传输的都是一个**byte stream/字节型的数据流**

# 网络编程的基础知识 (5)

- 网络资源不仅可以包括网络上各种简单对象：
  - 例如网络上的路径和文件(Web页、文本文件、图形(像)文件、声音片断)等；
- 还可以是一些复杂的对象：
  - 如数据库或搜索引擎。
- URL:是WWW资源统一资源定位器的缩写。  
他规范了WWW资源网络定位地址的表示方法。

# URL (Uniform Resource Locator)


- 网络资源定位器通常有5个部分组成：
  - 协议、主机名、端口号、文件和引用。
- URL的基本表示格式是：
  - *Protocol://hostname:/resourcenam#anch or*
  - Protocol,使用的协议, 它可以是http、ftp、news、telnet等。
  - Hostname:主机名, 指定域名服务器(DNS)能访问到的WWW服务的计算机, 例 [www.sun.com](http://www.sun.com)
  - Port:是可选的, 表示所连接的端口号, 如缺省, 将连接到协议缺省的端口。
  - Resourcenam:资源名, 是主机上能访问到的目录或文件。
  - Anchor:标记, 是可选的。他指定文件中有特  
定标记的位置。

# URL (Uniform Resource Locator)

- 例如:
- <http://www.sina.com.cn:8080/index.html>
- <http://www.gamelan.com:80/Gamelan/network.html#BOTTOM>
- <http://www.cumt.edu.cn/library/library.htm>

# 如何理解Socket?


## socket

/ˈsɒkɪt/ 

n. 插座，灯座，牙槽；窝，孔

高中

### 单词变形

 **socket**  
复数 **sockets**

### 配图例句

He is putting the plug into the **socket**.

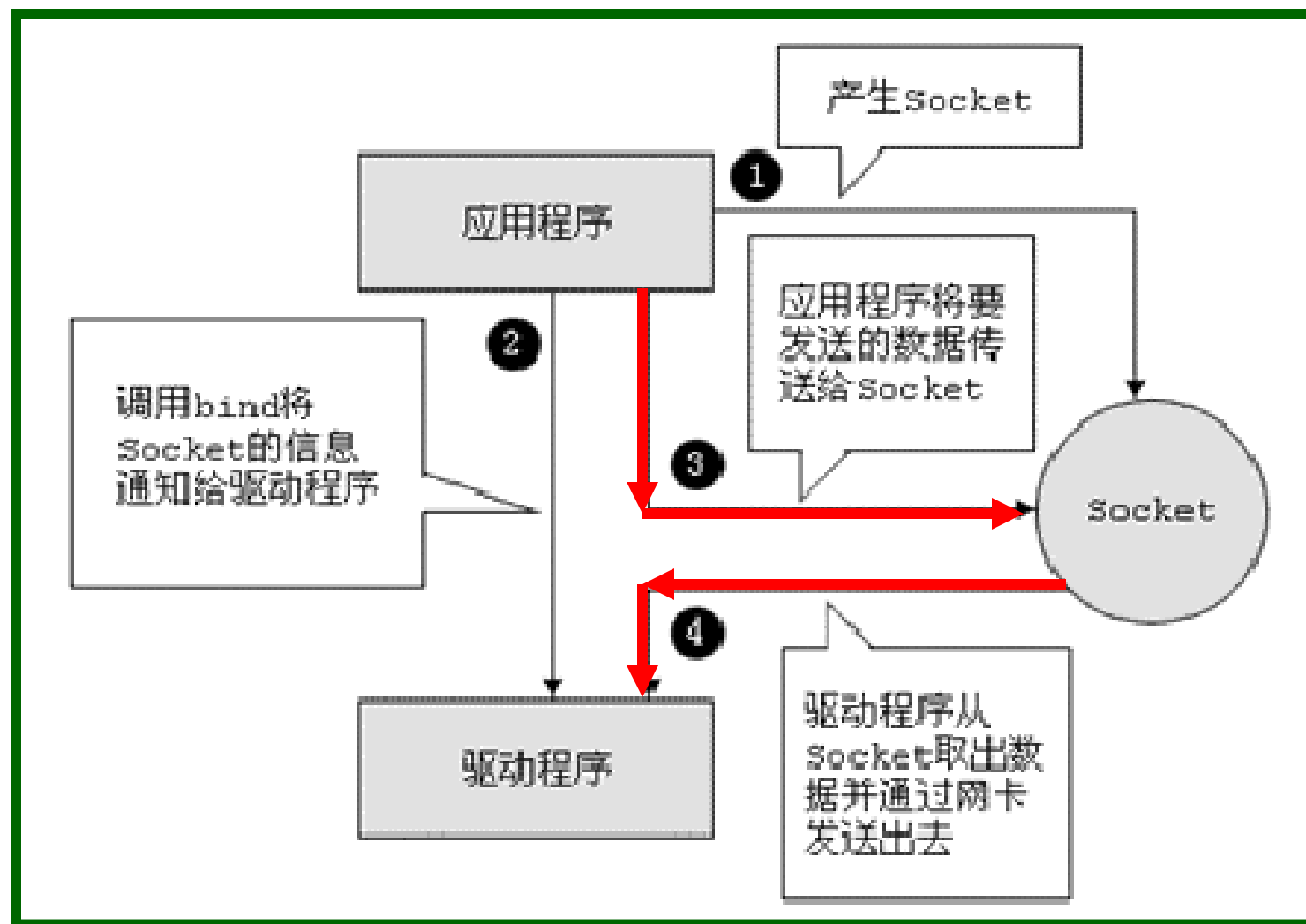
他正把插头插入插座。



# Socket (套接字)

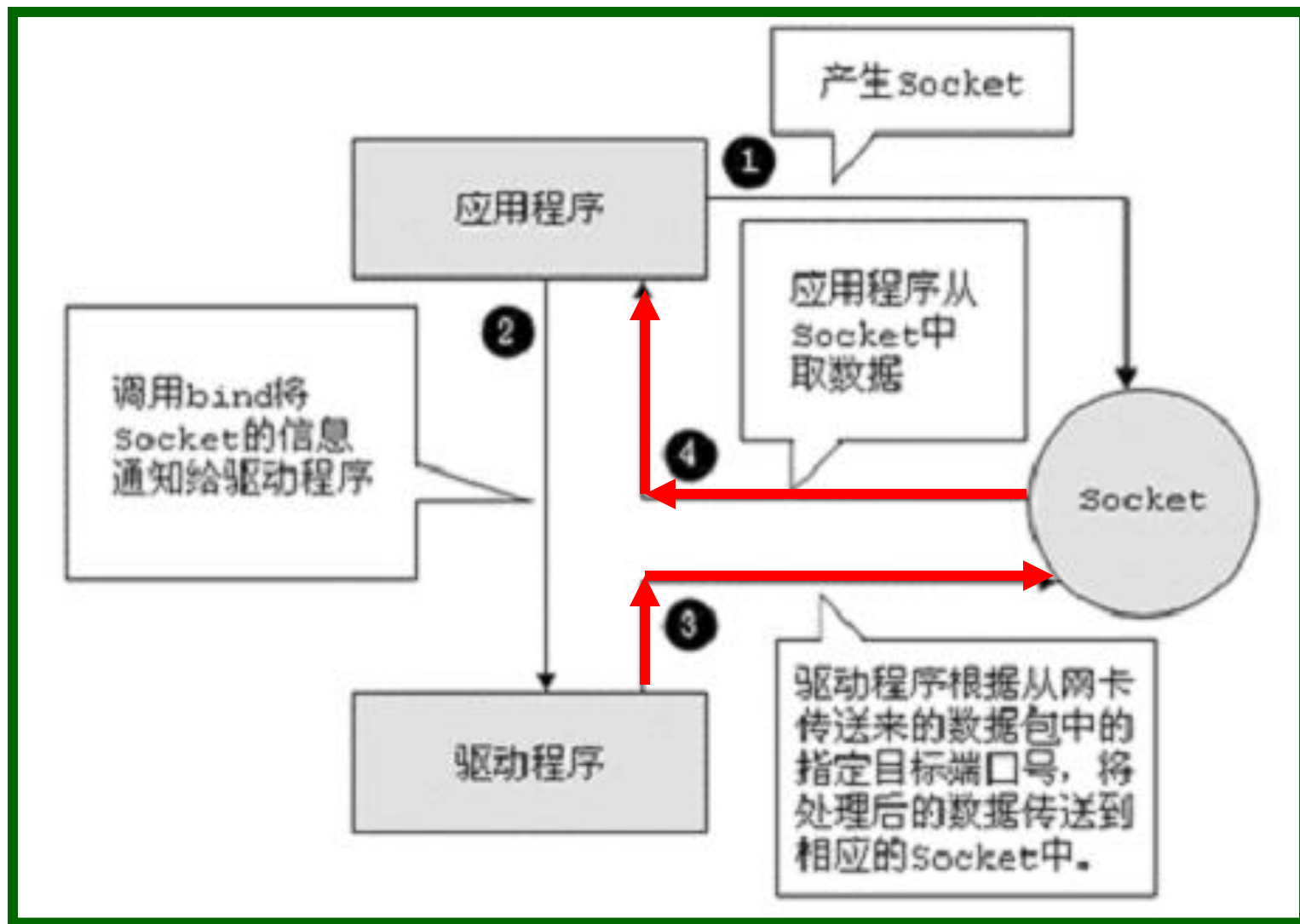
- **Socket是网络驱动层**提供给应用程序编程的接口和一种机制;
- **Socket在应用程序中创建，通过一种绑定机制与驱动程序建立关系，告诉自己所对应的IP和Port。**
- **可以把Socket比喻成一个港口码头，应用程序只要把货物放到港口码头上，就算完成了货物的运送。应用程序只需等待货物到达码头后，将货物取走；**

# Socket数据发送过程





# Socket数据接收的过程



# Java中网络编程类

- Java中网络编程类位于**java.net**包中。
- 很多应用程序需要可靠的、按顺序的数据传输，也有的应用程序不需要，因此在设计网络程序时应正确选择网络类。
  - URL、URLConnection、Socket和ServerSocket类是基于TCP协议的；
    - Socket类用于TCP通信的服务器和客户端。
    - ServerSocket类用于TCP通信的服务器端。
  - DatagramPacket、DatagramSocket和MulticastSocket类是基于UDP协议的。
    - DatagramSocket类用于UDP通信。

# 主要内容

- 网络编程的基础知识
- **基于TCP的网络程序设计**
- 基于UDP的网络程序设计
- 访问Internet网络资源



# TCP网络程序

- TCP网络程序的工作原理
- ServerSocket类
- Socket类
- 案例解析



# TCP网络程序

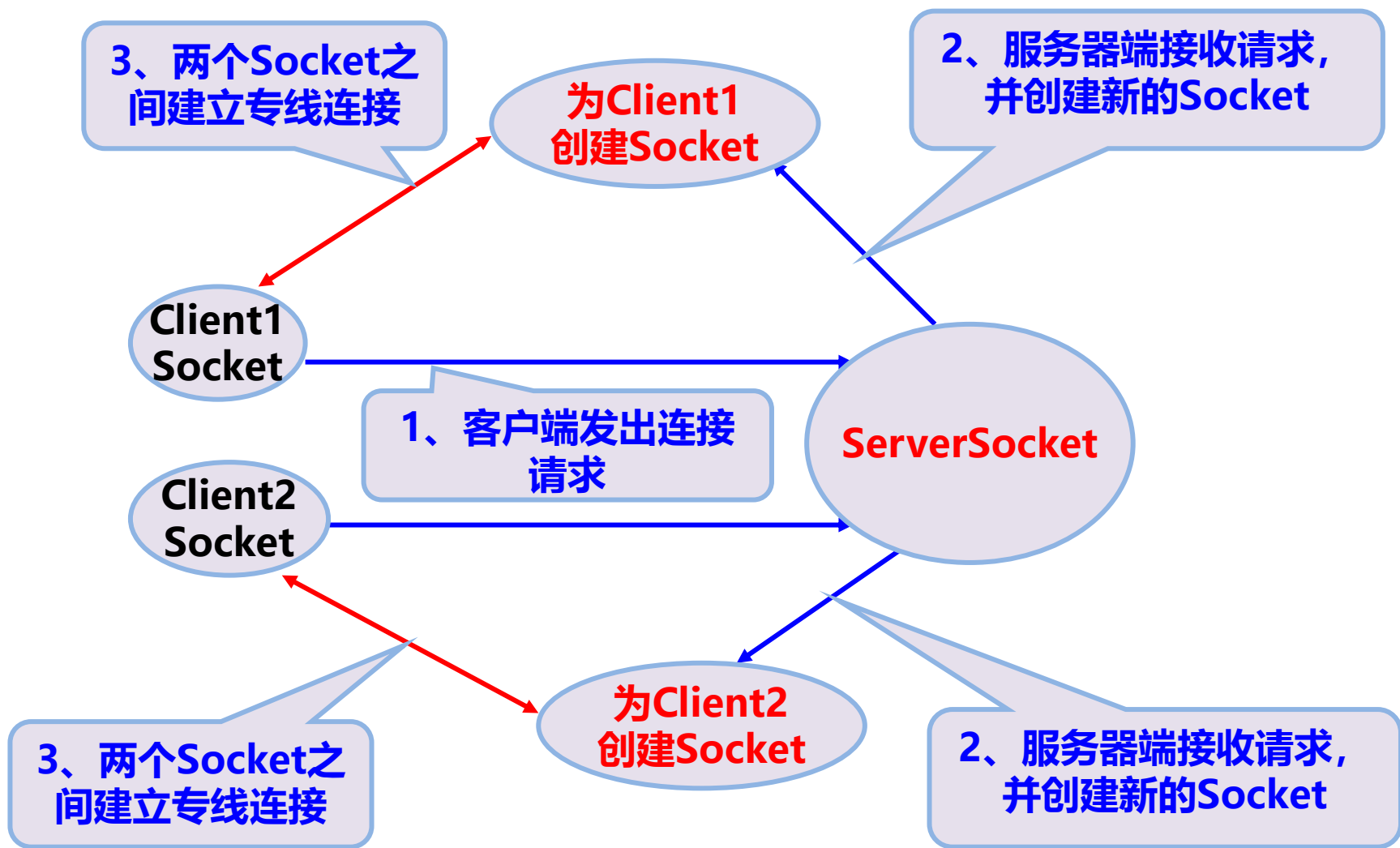
- 利用TCP协议进行通信的两个应用程序是有主从之分的，一个称为服务器程序，一个称为客户机程序，**两者的功能和编写方法不大一样。**

- TCP服务器程序类似114查号台，而TCP客户机程序类似普通电话。必须先有114查号台。普通电话才能拨打114，在114查号台那边是先有一个总机，总机专门用来接听拨打进来的电话，并不与外面的电话直接对话，而是将接进来的电话分配到一个空闲的座机上，然后由这个座机与外面的电话直接对话。
- 总机在没有空闲的座机时，可以让对方排队等候，但等候服务的电话达到一定数量时，总机就会彻底拒绝以后再拨打进来的电话。

# TCP网络程序

- Java中提供的ServerSocket类用于完成类似114查号台总机的功能,
- Socket类用于完成普通电话和114查号台端的座机功能。

# TCP网络程序的工作原理





# TCP客户端程序与TCP服务器端程序的交互过程

- 1) 服务器端创建一个ServerSocket，然后调用accept方法等待客户来连接；
- 2) 客户端程序创建一个Socket并请求与服务器建立连接；
- 3) 服务器端接收客户的连接请求，并创建一个新的Socket与该客户建立专线连接；
- 4) 建立了连接的两个Socket在一个单独的线程(由服务器程序创建)上对话；
- 5) 服务器开始等待新的连接请求，当新的连接请求到达时，重复步骤2-5的过程。

# InetAddress类

- InetAddress是用于表示计算机IP地址的一个类，而在日常应用中的计算机地址是用“192.168.0.1”、[www.sina.com.cn](http://www.sina.com.cn)等字符串格式来表示的。
- `getByName(String host)`方法：
  - 通过域名来构造类InetAddress的实例对象
- `getByAddress(byte[] addr)`方法
  - 通过4个字节的网络地址构造类InetAddress的实例对象

# J\_InetAddress.java

```
import java.net.InetAddress;
import java.net.UnknownHostException;
public class J_InetAddress
{
    public static void main(String[] args){
        String s="www.buaa.edu.cn";
        InetAddress ts=null;
        try{
            ts=InetAddress.getByName(s);
        }
        catch(UnknownHostException e){
            System.err.println(e);
        }
        if(ts!=null){
            System.out.println("The IP address of beihang is :"+ts.getHostAddress());
            System.out.println("The host name of beihang is :"+ts.getHostName());
        }
        else{
            System.out.println("can not access "+s);
        }
    }
}
```

The IP address of beihang is :106.39.41.16  
The host name of beihang is :www.buaa.edu.cn

## ServerSocket的常用方法

ServerSocket();	创建一个未绑定的ServerSocket
-----------------	----------------------

ServerSocket(int port);	绑定到指定的port上,连接队列默认50
-------------------------	----------------------

ServerSocket(int port, int backlog);	指定最大连接队列
--------------------------------------	----------

ServerSocket(int port, int backlog, InetAddress bindAddr);	
--	--



Socket accept();	void bind(SocketAddress endpoint);
------------------	------------------------------------

void bind(SocketAddress endpoint, int backlog);	
---	--

SocketAddress getLocalSocketAddress();	void close();
--	---------------

InetAddress getInetAddress();	int getLocalPort();
-------------------------------	---------------------

boolean isBound();	boolean isClosed();
--------------------	---------------------

void setSoTimeout(int timeout);	int getSoTimeout();
---------------------------------	---------------------

- 构造函数
  - public ServerSocket();
  - **public ServerSocket(int port);**
  - public ServerSocket(int port, int backlog);
  - public ServerSocket(int port, int backlog, InetAddress bindAddr);
- close()方法;
- accept()方法;

用第一个构造函数创建 `ServerSocket` 对象，没有与任何端口号绑定，不能被直接使用，还要继续调用 `bind` 方法，才能完成其他构造函数所完成的功能。↵

用第二个构造函数创建 `ServerSocket` 对象，我们就可以将这个 `ServerSocket` 绑定到一个指定的端口上，就象为我们的呼叫中心安排一个电话号码一样，如果在这里指定的端口号为 0，系统就会为我们分配一个还没有被其他网络程序所使用的端口号，作为服务器程序，端口号必须事先指定，其他客户才能根据这个号码进行连接，所以将端口号指定为 0 的情况并不常见。↵

用第三个构造函数创建 `ServerSocket` 对象，就是在第二个构造函数的基础上，我们根据 `backlog` 参数指定，在服务器忙时，可以与之保持连接请求的等待客户数量，对于第二个构造函数，没有指定这个参数，则使用默认的数量，大小为 50。↵

用第四个构造函数创建 `ServerSocket` 对象，我们除了指定第三个构造函数中的参数外，还可以指定相关的 IP 地址，这种情况适用于计算机上有多块网卡和多个 IP 的情况，我们可以明确规定我们的 `ServerSocket` 在哪块网卡或 IP 地址上等待客户的连接请求，在前面几个构造函数中，都没有指定网卡的 IP 地址，底层驱动程序会为我们选择其中一块网卡或一个 IP 地址，显然，对于我们一般只有一块网卡的情况，我们就不用专门指定 IP 地址了。

# Socket的常用构造方法

## Socket的常用构造方法

<b>Socket();</b>	<b>创建一个未连接的Socket</b>
------------------	-----------------------

<b>Socket(InetAddress addr, int port);</b>	<b>指定IP地址和port</b>
--	--------------------

<b>Socket(InetAddress addr, int port, InetAddress localaddr, int localPort);</b>
--

<b>Socket(String host, int port, InetAddress localaddr, int localPort);</b>
---

<b>Socket(String host, int port);</b>	<b>Socket(SocketImpl impl);</b>
---------------------------------------	---------------------------------

## Socket的常用方法(续)

**void bind(SocketAddress bindpoint); void close();**

**void connect (SocketAddress endpoint); int getPort();**

**void connect (SocketAddress endpoint, int timeout);**

**InetAddress getInetAddress(); int getLocalPort();**

**InetAddress getLocalAddress(); boolean isConnected();**

**SocketAddress getLocalSocketAddress();**

**InputStream getInputStream(); boolean isClosed();**

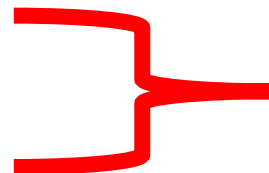
**OutputStream getOutputStream(); boolean isBound();**

**SocketAddress getRemoteSocketAddress();**



# Socket

- 构造函数
  - Socket();
  - Socket(String host,int port);
  - Socket(InetAddress addr, int port);
  - Socket(String host,int port, InetAddress localaddr,int localPort);
  - Socket(InetAddress addr, int port, InetAddress localaddr, int localPort);
- getInputStream和getOutputStream方法



```

public Socket() throws
    UnknownHostException, IOException
public Socket(InetAddress address, int port) throws IOException
public Socket(String host, int port, InetAddress localAddr, int localPort)
    throws IOException
public Socket(InetAddress address, int port, InetAddress localAddr,
    int localPort) throws IOException

```

用第一个构造函数创建 Socket 对象，不与任何服务器建立连接，不能被直接使用，需要调用 connect 方法，才能完成和其他构造函数一样的功能。如果我们想用同一个 Socket 对象，去轮循连接多个服务器，可以使用这个构造函数创建 socket 对象后，再不断调用 connect 方法去连接每个服务器。

用第二个和第三个构造函数创建 Socket 对象后，会根据参数去连接在特定地址和端口上运行的服务器程序，第二个构造函数接受字符串格式的地址，第三个构造函数接受 InetAddress 对象所包装的地址。

第四个和第五个构造函数在第二个和第三个构造函数的基础上，还指定了本地 Socket 所绑定的 IP 地址和端口号，由于客户端的端口号的选择并不重要，所以一般情况下，我们不会使用这两个构造函数，其中的原因，我们在前面已经讲了许多，这里就不再多说了。

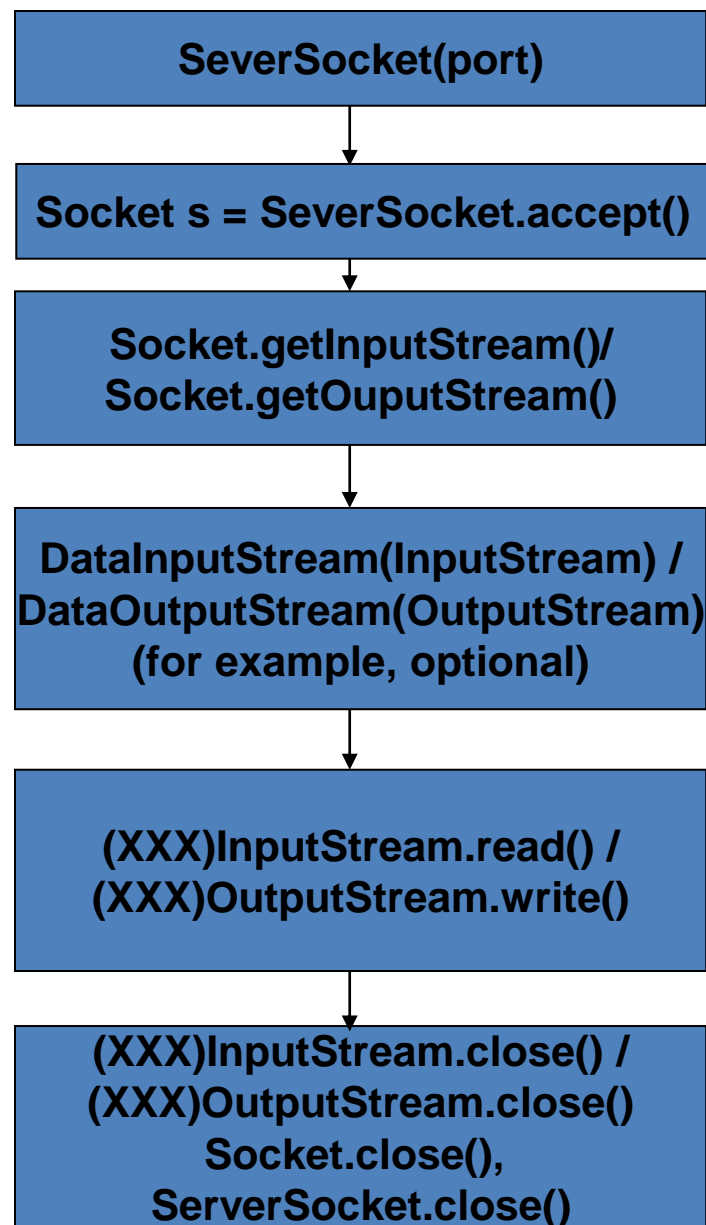
看完了上面几个 Socket 构造函数的介绍，我们了解到，对于通常情况的应用，选择第二个构造函数来创建客户端的 Socket 对象并与服务器建立连接，是非常简单和方便的。

服务器端程序调用 ServerSocket.accept 方法等待客户的连接请求，一旦 accept 接收了客户连接请求，该方法将返回一个与该客户建立了专线连接的 Socket 对象，不用程序去创建这个 Socket 对象。

当客户端和服务端端的两个 Socket 建立了专线连接后，连接的一端能向另一端连续写入字节，也能从另一端连续读入字节，也就是建立了专线连接的两个 Socket 是以 IO 流的方式进行数据交换的，Java 提供了 Socket.getInputStream 方法返回 Socket 的输入流对象，Socket.getOutputStream 方法返回 Socket 的输出流对象。只要连接的一端向该输出流对象写入了数据，连接的另一端就能从其输入流对象中读取到这些数据。

# Typical Steps to Write a Socket Server

1. Create a server socket
2. Accept a connection request from a client and return a new socket
3. Get input stream / output stream from the accepted socket
4. Encapsulate raw input stream / output stream to high level streams according to data type (optional)
5. Receive/send message from/to streams
6. Release resources



# 简单的Client/Server程序

- 客户端和服务端采用TCP协议进行通信，当客户端链接服务器端程序后，服务器程序向客户端程序发送客户端的IP地址和端口号。

# 简单的Client/Server程序

```
public class TestServer {  
    public static void main(String args[]) {  
        try {  
            ServerSocket s = new ServerSocket(8888);  
            while (true) {  
                Socket s1 = s.accept();  
                OutputStream os = s1.getOutputStream();  
                DataOutputStream dos = new DataOutputStream(os);  
                dos.writeUTF("客户端地址信息: " + s1.getInetAddress()  
                    + "\t客户端通信端口号: " + s1.getPort());  
                dos.writeUTF("再见!");  
                dos.close();  
                s1.close();  
            }  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

客户端可以  
运行多次

UTF-8是与操作系统和平台  
无关的编码方式，所以对于  
异构计算机之间的通信采用  
这种方式比较好。

# 简单的Client/Server程序

```
import java.io.*;
import java.net.Socket;
public class TestClient {
    public static void main(String args[]) {
        try {
            Socket s1 = new Socket("127.0.0.1", 8888);
            InputStream is = s1.getInputStream();
            DataInputStream dis = new DataInputStream(is);
            System.out.print("服务器端地址信息:"+s1.getInetAddress());
            System.out.println("服务器端通信端口号:"+s1.getPort());
            System.out.println("接受的信息是*****");
            System.out.println(dis.readUTF());
            System.out.println(dis.readUTF());
            dis.close();
            s1.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

# 简单的Client/Server程序

服务器端地址信息：**/127.0.0.1**服务器端通信端口号：**8888**

接受的信息是**\*\*\*\*\***

客户端地址信息：**/127.0.0.1** 客户端通信端口号：**52941**

再见！

# 简单的一对一聊天程序

- 程序只能一对一聊天，而且只能一方说一句，不能多说，
- 因为程序是单线程的。



```
import java.io.*;
import java.net.Socket;
import java.net.ServerSocket;
import java.net.SocketException;

public class TestServer {
    public static void main(String args[]) {
        try {
            ServerSocket s = new ServerSocket(8888);
            Socket s1 = s.accept();

            OutputStream os = s1.getOutputStream();
            DataOutputStream dos = new DataOutputStream(os);

            InputStream is = s1.getInputStream();
            DataInputStream dis = new DataInputStream(is);

            InputStreamReader isr = new InputStreamReader(System.in);
            BufferedReader br = new BufferedReader(isr);

            String info;
            while(true){
                info = dis.readUTF();
                System.out.println("对方说: " + info);
                if(info.equals("bye"))
                    break;
                info = br.readLine();
                dos.writeUTF(info);
                if(info.equals("bye"))
                    break;
            }
            dis.close();
            dos.close();
            s1.close();
            s.close();
        } catch (SocketException e) {
            System.out.println("网络连接异常, 程序退出!");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



```

import java.io.*;
import java.net.Socket;
import java.net.SocketException;

public class TestClient {
    public static void main(String args[]) {
        try {
            Socket s1 = new Socket("localhost",8888);

            InputStream is = s1.getInputStream();
            DataInputStream dis = new DataInputStream(is);

            OutputStream os = s1.getOutputStream();
            DataOutputStream dos = new DataOutputStream(os);

            InputStreamReader isr = new InputStreamReader(System.in);
            BufferedReader br = new BufferedReader(isr);

            String info;
            while(true){
                info = br.readLine();
                dos.writeUTF(info);
                if(info.equals("bye"))
                    break;
                info = dis.readUTF();
                System.out.println("对方说: " + info);
                if(info.equals("bye"))
                    break;
            }
            dis.close();
            dos.close();
            s1.close();
        }catch (SocketException e) {
            System.out.println("网络连接异常, 程序退出!");
        }catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

GA C:\WINDOWS\system32\cmd.

E:\>java TestServer

对方说: hello

how are you

对方说: i am fine

and you

对方说: me too,thank you!

GA C:\WINDOWS\system32\cmd.

E:\>java TestClient

hello

对方说: how are you

i am fine

对方说: and you

me too,thank you!

# 自由聊天程序

- 为了保证一次可以说多句，可以使用多线程，一个线程用来说，一个线程用来听。  
互不干扰！！！！

# 服务器主线程

```
public class TestServer {  
    public static void main(String args[]) {  
        try {  
            ServerSocket s = new ServerSocket(8888);  
            Socket s1 = s.accept();  
            OutputStream os = s1.getOutputStream();  
            DataOutputStream dos = new DataOutputStream(os);  
            InputStream is = s1.getInputStream();  
            DataInputStream dis = new DataInputStream(is);  
  
            new MyServerReader(dis).start();  
            new MyServerWriter(dos).start();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

# 读线程

```
class MyServerReader extends Thread{
    private DataInputStream dis;
    public MyServerReader(DataInputStream dis ){
        this.dis = dis;
    }
    public void run(){
        String info;
        try{
            while(true){
                info = dis.readUTF();
                System.out.println("对方说: " + info);
                if(info.equals("bye")){
                    System.out.println("对方下线, 程序退出!");
                    System.exit(0);
                }
            }
        }catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



# 写线程

```
class MyServerWriter extends Thread{
    private DataOutputStream dos;
    public MyServerWriter(DataOutputStream dos){
        this.dos = dos;
    }
    public void run(){
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        String info;
        try{
            while(true){
                info = br.readLine();
                dos.writeUTF(info);
                if(info.equals("bye")){
                    System.out.println("自己下线, 程序退出!");
                    System.exit(0);
                }
            }
        }catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

# 客户端主线程

```
public class TestClient {  
    public static void main(String args[]) {  
        try {  
            Socket s1 = new Socket("127.0.0.1",8888);  
            InputStream is = s1.getInputStream();  
            DataInputStream dis = new DataInputStream(is);  
            OutputStream os = s1.getOutputStream();  
            DataOutputStream dos = new DataOutputStream(os);  
  
            new MyClientReader(dis).start();  
            new MyClientWriter(dos).start();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```



# 客户端读线程

```
class MyClientReader extends Thread{
    private DataInputStream dis;
    public MyClientReader(DataInputStream dis ){
        this.dis = dis;
    }
    public void run(){
        String info;
        try{
            while(true){
                info = dis.readUTF();
                System.out.println("对方说: " + info);
                if(info.equals("bye")){
                    System.out.println("对方下线, 程序退出!");
                    System.exit(0);
                }
            }
        }catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

# 客户端写线程

```
class MyClientWriter extends Thread{
    private DataOutputStream dos;
    public MyClientWriter(DataOutputStream dos){
        this.dos = dos;
    }
    public void run(){
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        String info;
        try{
            while(true){
                info = br.readLine();
                dos.writeUTF(info);
                if(info.equals("bye")){
                    System.out.println("自己下线，程序退出!");
                    System.exit(0);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

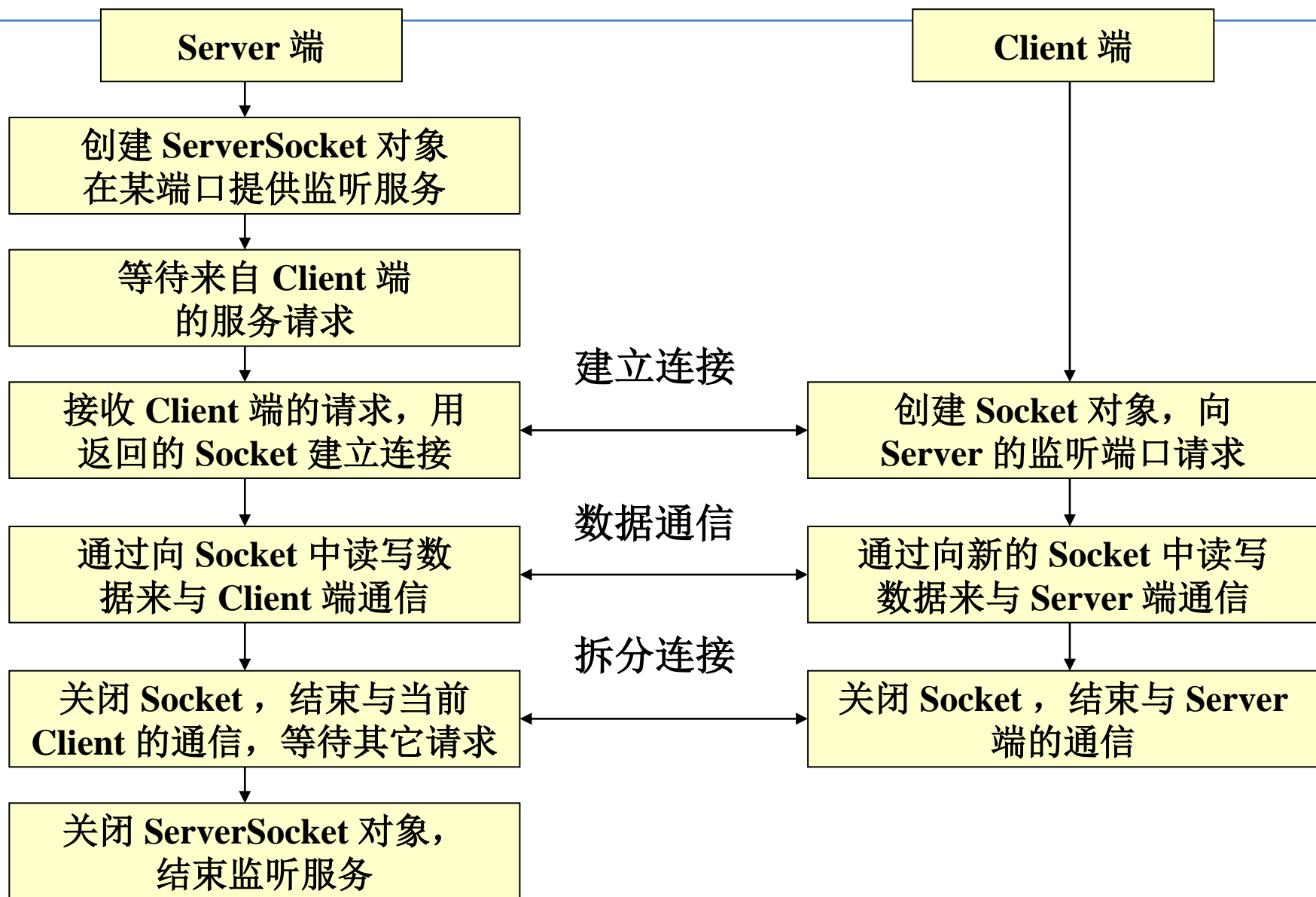


# 完善的TCP服务器程序模型（思考题）

- 编程实例：
  - 服务器程序能够同时与多个客户端对话，客户端每次向服务器发送一行字符文本，服务器就将这行字符文本中的所有字符反向排列返回送给客户端，当客户端向服务器发送的一行字符文本内容为“quit”时，服务器结束与客户端的会话。
- TCP服务器程序模型的缩写要点：
  - TCP服务器程序要想能接收多个客户端连接，需要循环调用ServerSocket.accept方法；
  - 服务器程序与每个客户端连接的会话过程不能互相影响，需要在独立的线程中进行；
  - 一个线程服务对象与一个服务器端Socket对象相关联，共同来完成一个客户端的会话。

# 在TCP网络连接上传递对象

- **ObjectInputStream和ObjectOutputStream**可以使用**ObjectInputStream和ObjectOutputStream**来包装底层网络字节流，TCP服务器和TCP客户端之间就可以传递对象类型的数据，实现从底层输入流中读取对象类型的数据和将对象类型的数据写入到底层输出流。
- RMI(remote method invocation)编程：是java进行分布式编程的基础。



# 思考题

- 创建一个服务器端，多个客户端。客户端每隔一秒向服务器端报告它的名字及当前时间。服务器端将客户端传送的内容在服务器端屏幕上打印出来。客户端可以运行在不同的机器上。
- 服务器端首先创建一个服务线程池，每当接受到一个客户请求，从线程池中调度一个空闲状态线程和客户通信。

# 主要内容

- 网络编程的基础知识
- 基于TCP的网络程序设计
- **基于UDP的网络程序设计**
- 访问Internet网络资源



# 基于UDP的网络程序设计

- UDP网络程序的工作原理
- DatagramSocket类
- DatagramPacket类
- InetAddress类
- 最简单的UDP程序
- 用UDP编写网络聊天程序



# UDP网络程序

- 用户数据报协议UDP (user datagram protocol) 是一个无连接的、发送独立数据包的协议，它不保证数据按顺序传送和正确到达。
- 数据报Socket又称为UDP套接字，它无需建立、拆除连接，而是直接将信息打包传向指定的目的地，使用简单，占用资源少，适合于断续、非实时通信。
- 利用UDP通信的两个程序是平等的，没有主次之分，两个程序的代码可以完全一样。

# DatagramSocket类

- 构造函数:
  - Public DatagramSocket();
  - Public DatagramSocket(int port);
  - Public DatagramSocket(int port, InetAddress laddr);
- Close()方法;
- Send(DatagramPacket p)方法
- Receive(DatagramPacket p)方法


## Constructor Summary

	<u><a href="#">DatagramSocket</a></u> () Constructs a datagram socket and binds it to any available port on the local host machine.
protected	<u><a href="#">DatagramSocket</a></u> ( <u><a href="#">DatagramSocketImpl</a></u> impl) Creates an unbound datagram socket with the specified <code>DatagramSocketImpl</code> .
	<u><a href="#">DatagramSocket</a></u> (int port) Constructs a datagram socket and binds it to the specified port on the local host machine.
	<u><a href="#">DatagramSocket</a></u> (int port, <u><a href="#">InetAddress</a></u> laddr) Creates a datagram socket, bound to the specified local address.
	<u><a href="#">DatagramSocket</a></u> ( <u><a href="#">SocketAddress</a></u> bindaddr) Creates a datagram socket, bound to the specified local socket address.

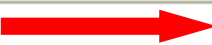
# DatagramPacket类

- 如果把DatagramSocket比作创建的港口码头，那么DatagramPacket就是发送和接收数据的集装箱。
- 构造函数
  - `Public DatagramPacket(byte[] buf,int length);`
  - `Public DatagramPacket(byte[] buf,int length,InetAddress address,int port);`
- `getInetAddress()`和`getPort()`方法;
- `Byte[] getData()`和`getLength()`方法;


## Constructor Summary

**DatagramPacket**(byte[] buf, int length) 


Constructs a DatagramPacket for receiving packets of length length.

**DatagramPacket**(byte[] buf, int length, InetAddress address, int port) 


Constructs a datagram packet for sending packets of length length to the specified port number on the specified host.

**DatagramPacket**(byte[] buf, int offset, int length) 


Constructs a DatagramPacket for receiving packets of length length, specifying an offset into the buffer.

**DatagramPacket**(byte[] buf, int offset, int length, InetAddress address, int port) 

Constructs a datagram packet for sending packets of length length with offset ioffsetto the specified port number on the specified host.

**DatagramPacket**(byte[] buf, int offset, int length, SocketAddress address) 

Constructs a datagram packet for sending packets of length length with offset ioffsetto the specified port number on the specified host.

**DatagramPacket**(byte[] buf, int length, SocketAddress address) 

Constructs a datagram packet for sending packets of length length to the specified port number on the specified host.

## Method Summary

<u>InetAddress</u>	<u><b>getAddress()</b></u> Returns the IP address of the machine to which this datagram is being sent or from which the datagram was received.
byte[]	<u><b>getData()</b></u> Returns the data buffer.
int	<u><b>getLength()</b></u> Returns the length of the data to be sent or the length of the data received.
int	<u><b>getOffset()</b></u> Returns the offset of the data to be sent or the offset of the data received.
int	<u><b>getPort()</b></u> Returns the port number on the remote host to which this datagram is being sent or from which the datagram was received.
<u>SocketAddress</u>	<u><b>getSocketAddress()</b></u> Gets the SocketAddress (usually IP address + port number) of the remote host that this packet is being sent to or is coming from.
void	<u><b>setAddress(InetAddress iaddr)</b></u> Sets the IP address of the machine to which this datagram is being sent.
void	<u><b>setData(byte[] buf)</b></u> Set the data buffer for this packet.
void	<u><b>setData(byte[] buf, int offset, int length)</b></u> Set the data buffer for this packet.
void	<u><b>setLength(int length)</b></u> Set the length for this packet.

# 最简单的UDP程序

- 字符串与字节数组之间的双向转换。
- UDP接收程序必须先启动运行，才能接收UDP发送程序发送的数据。
- 举例：
  - [UdpSend.java](#)    [UdpRecv.java](#)

```
public class UdpRecv {  
    public static void main(String[] args) throws Exception {  
        DatagramSocket ds = new DatagramSocket(3000);  
        byte[] buf = new byte[1024];  
        DatagramPacket dp = new DatagramPacket(buf, 1024);  
        ds.receive(dp);  
        String strRecv = new String(dp.getData(), 0, dp.getLength()) + " from "  
            + dp.getAddress().getHostAddress() + ":" + dp.getPort();  
        System.out.println(strRecv);  
        ds.close();  
    }  
}
```



```
public class UdpSend {  
    public static void main(String[] args) throws Exception {  
        DatagramSocket ds = new DatagramSocket();  
        String str = "hello world";  
        DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(),  
            InetAddress.getByName("127.0.0.1"), 3000);  
        ds.send(dp);  
        ds.close();  
    }  
}
```

hello world from 127.0.0.1:63584

# UDP网络程序的编写步骤

- 第一步：服务器端/客户端创建 DatagramSocket 实例对象；
- 第二步：编写 send() 和 receive() 方法
  - DatagramSocket.send(DatagramPacket p);
  - DatagramSocket.receive(DatagramPacket p);
- 第三步：DatagramSocket.close();

# 小结

- 理解 URL类是对统一资源定位符的抽象
- 理解IP、port、InetAddress类
- 理解Socket
- 掌握基于TCP的编程模式
- 掌握基于UDP的编程模式
- 基于UDP的通信和基于TCP的通信不同，基于UDP的信息传递更快，但不提供可靠性保证。