

JAVA网络编程

本章内容

1. 网络编程基础知识
2. java的基本网络支持
3. 基于TCP协议的网络编程
4. 一个具体编程实例

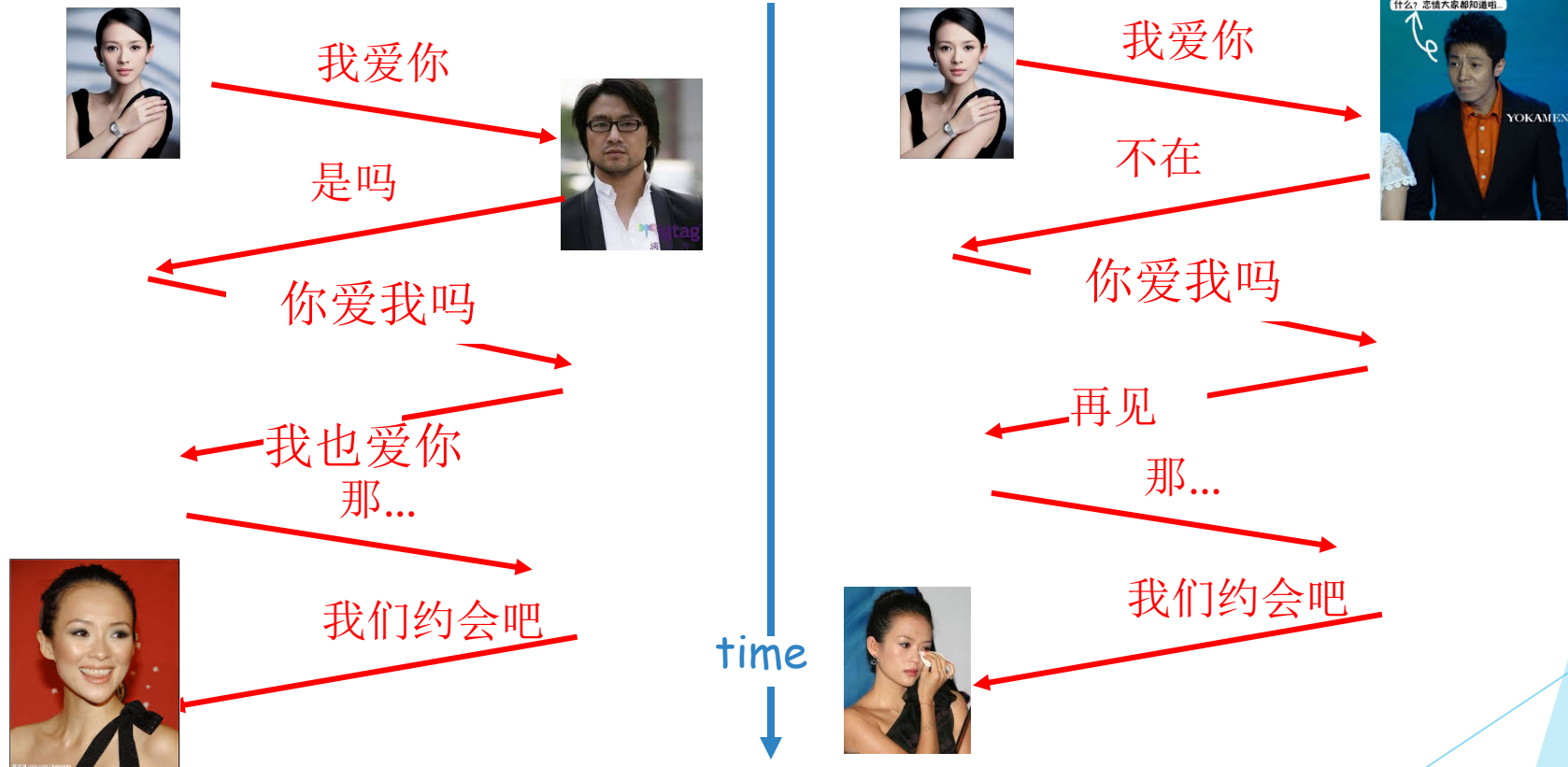
网络基础知识

- ▶ 通信协议：信息交换的双方约定
- ▶ 协议模型：OSI模型和TCP/IP模型



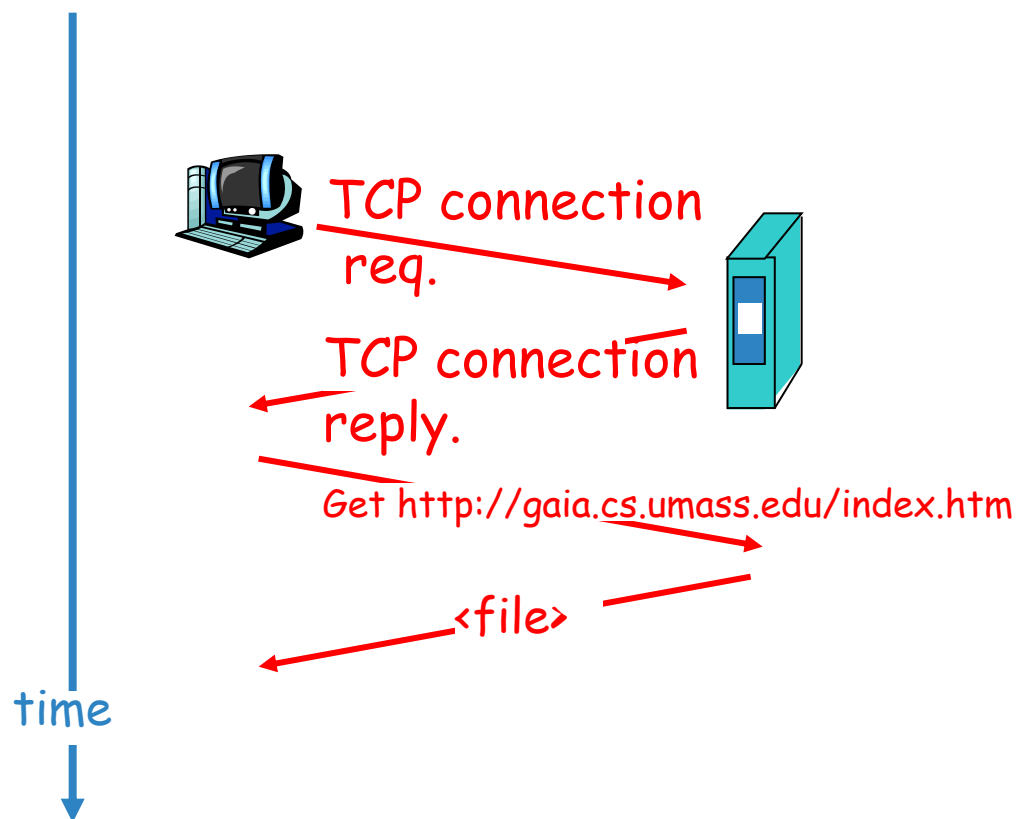
协议定义通信中消息的内容、顺序

a human protocol :



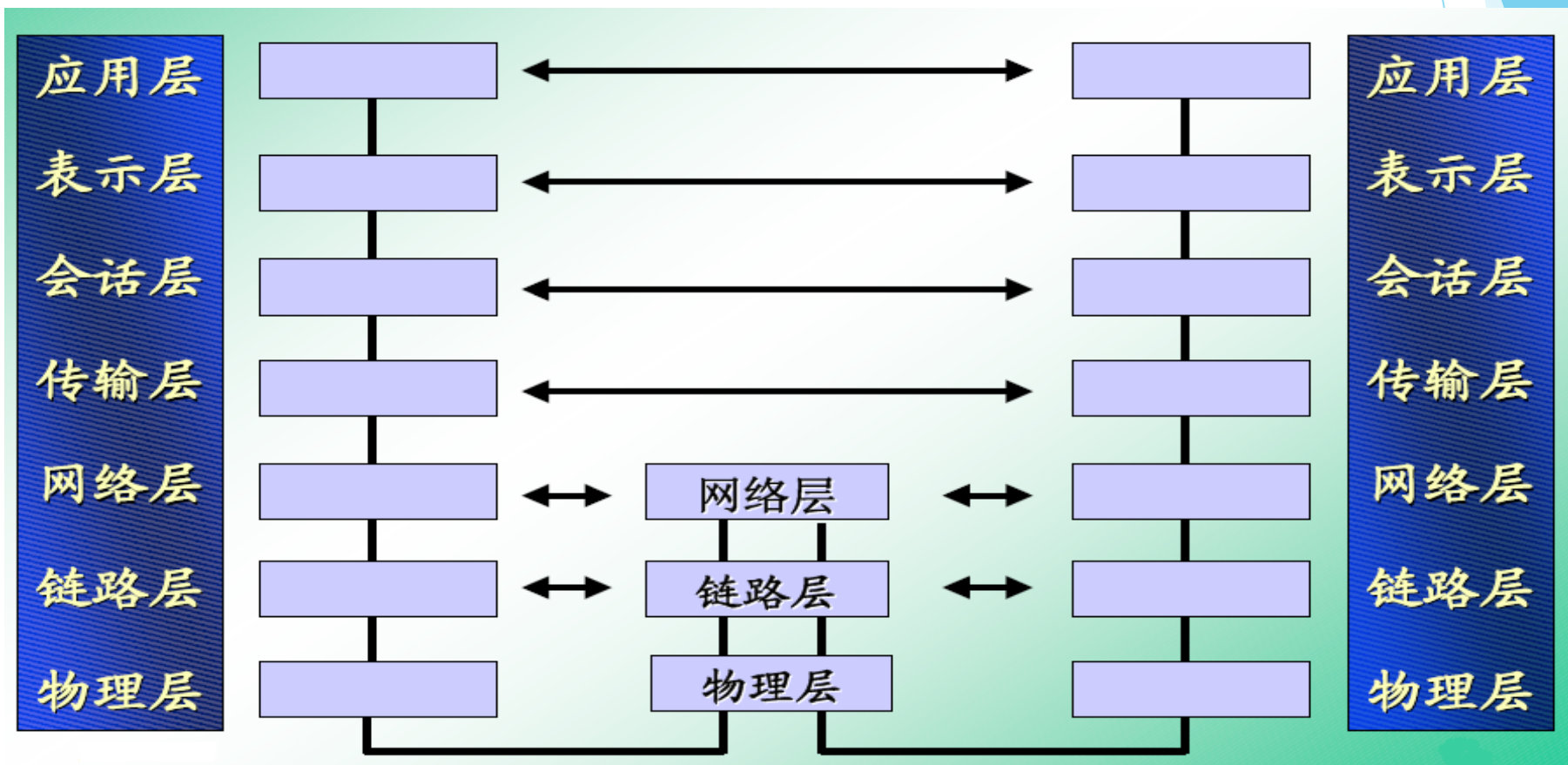
协议定义通信中消息的内容、顺序

a network protocol :



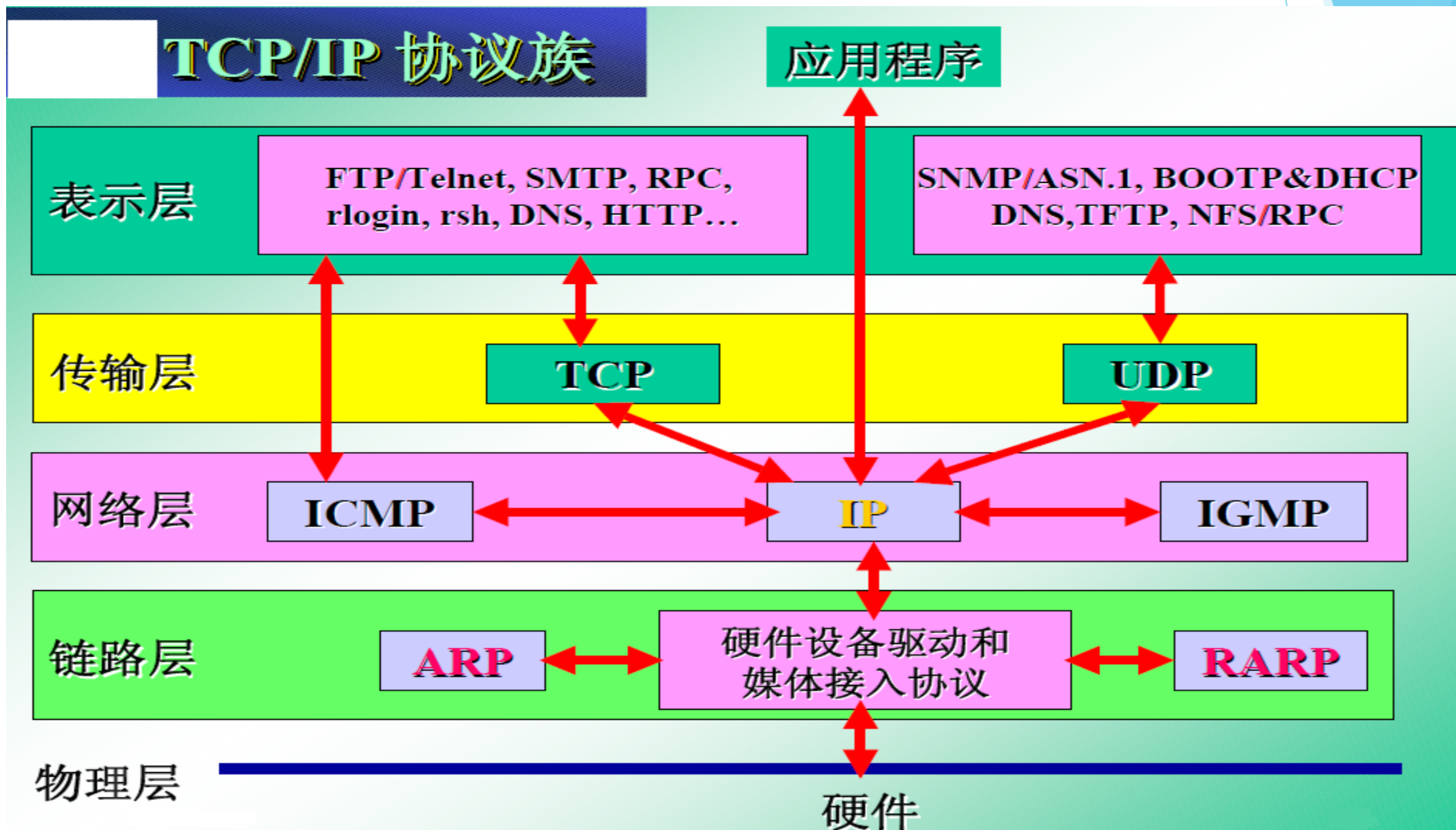
网络基础知识

▶ 协议模型：OSI模型



网络基础知识

- ▶ 协议模型：TCP/IP模型和TCP/IP协议



IP地址

- ▶ IP网络中每台主机都有唯一的IP地址，用于标识网络中的每台主机。
- ▶ IP地址是一个32位的二进制数序列。
- ▶ IP地址由两部分组成：IP网址和IP主机地址
- ▶ 网络掩码用来区分哪部分是网址，哪部分是主机地址，把主机地址与网络掩码进行二进制与的操作，即可得到IP网址。
- ▶ 域名(主机名.单位名.网络名.顶层域名)与IP，DNS服务器存放域名与IP地址的映射信息

TCP协议及端口

- ▶ 通常一台主机上总是有很多个进程需要网络资源进行网络通讯。**网络通讯的对象**准确的讲不是主机，而应该是**主机中运行的进程**。
- ▶ 只有通过**主机名或IP地址和端口号的组合**才能唯一的确定网络通讯中的对象：**进程**。
- ▶ 面向连接服务和无连接服务的通信协议端口，是一种抽象的软件结构，包括一些数据结构和**I/O（基本输入输出）缓冲区**

TCP协议及端口

- ▶ 端口号(port number): 网络通信时同一机器上的不同进程的标识。
 - ▶ 如:80, 21, 23, 25, 其中0~1023为系统保留的端口号, 如21分配给FTP服务, 80分配给http服务等。1024到65535的端口号供用户自定义的服务使用。
- ▶ 服务类型(service): 网络的各种服务。
 - ▶ http, telnet, ftp, smtp
 - ▶ 这里说明的是: TCP与UDP都用端口号来标志进程, 但在一个主机上其取值是独立的, 允许两者相同端口号。

TCP/UDP

▶ TCP-打电话

- ▶ 通过端口来区分程序间的若干通信
- ▶ 数据的传送是按字节流的方式顺序传播

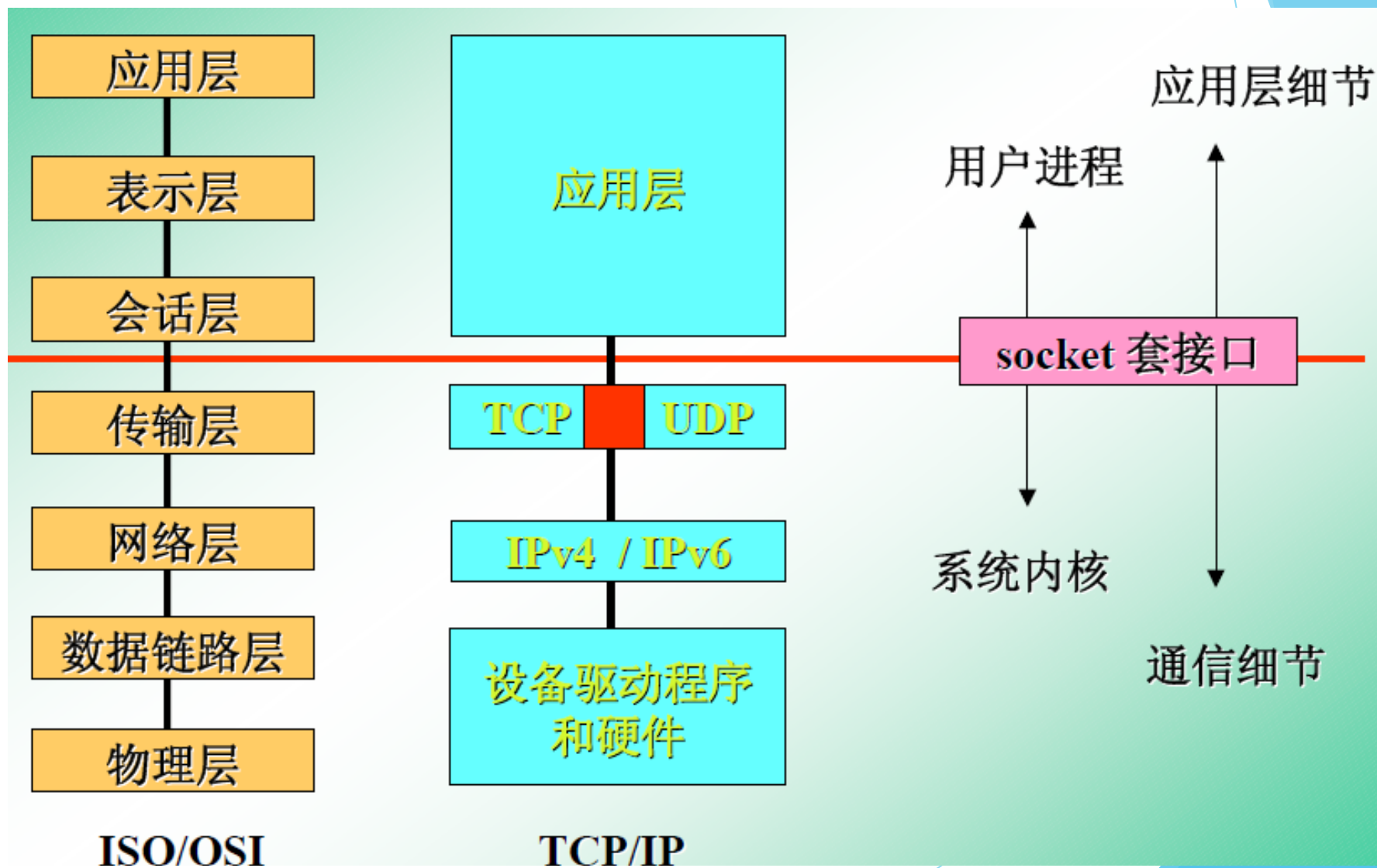
▶ UDP-写信

- ▶ 同样通过端口来区分程序间的若干通信
- ▶ 数据的传送是按数据报的方式传播，包到达的先后顺序不固定。

重要概念

- ▶ 应用层与传输层间的通信由用户编程实现，通过socket对象调用进行
- ▶ TCP/UDP 由端口号标识进程
- ▶ 用IP地址和端口号对可以标识通信的一方，这就是套接字socket

应用层与传输层间的通信连接



本章内容

1. 网络编程基础知识
2. java的基本网络支持
3. 基于TCP协议的网络编程
4. 一个具体编程实例
5. 大作业：珠海浮生记网络游戏

java的基本网络支持

- ▶ java.net.InetAddress类
- ▶ IP地址是IP使用的32位（IPv4）或者128位（IPv6）位无符号数字，它是传输层协议TCP，UDP的基础。InetAddress是Java对IP地址的封装，在java.net中有许多类都使用到了InetAddress，包括ServerSocket，Socket，DatagramSocket等等。

java的基本网络支持

- ▶ InetAddress描述了32位或64位IP地址，要完成这个功能，InetAddress类主要依靠两个支持类**Inet4Address**和**Inet6Address**，这三个类是继承关系，InetAddress是父类，Inet4Address和Inet6Address是子类。
- ▶ InetAddress类提供了将主机名解析为IP地址（或反之）的方法

java的基本网络支持

```
1. public class InetAddressTest{
2.     public static void main(String[] args)throws Exception    {
3.         //根据主机名来获取对应的InetAddress实例
4.         InetAddress ip = InetAddress.getByName("www.oneedu.cn");
5.         //判断是否可达
6.         System.out.println("oneedu是否可达: " + ip.isReachable(2000));
7.         //获取该InetAddress实例的IP字符串
8.         System.out.println(ip.getHostAddress());
9.         //根据原始IP地址来获取对应的InetAddress实例
10.        InetAddress local = InetAddress.getByAddress(new byte[]
11.        {127,0,0,1});
12.        System.out.println("本机是否可达: " + local.isReachable(5000));
13.        //获取该InetAddress实例对应的全限定域名
14.        System.out.println(local.getCanonicalHostName());
15.    }
16. }
```

本章内容

1. 网络编程基础知识
2. java的基本网络支持
3. 基于TCP协议的网络编程
4. 一个具体编程实例
5. 大作业：珠海浮生记网络游戏

基于TCP/IP协议的网络编程

▶ 1. 网络模型

▶ 客户机/服务器

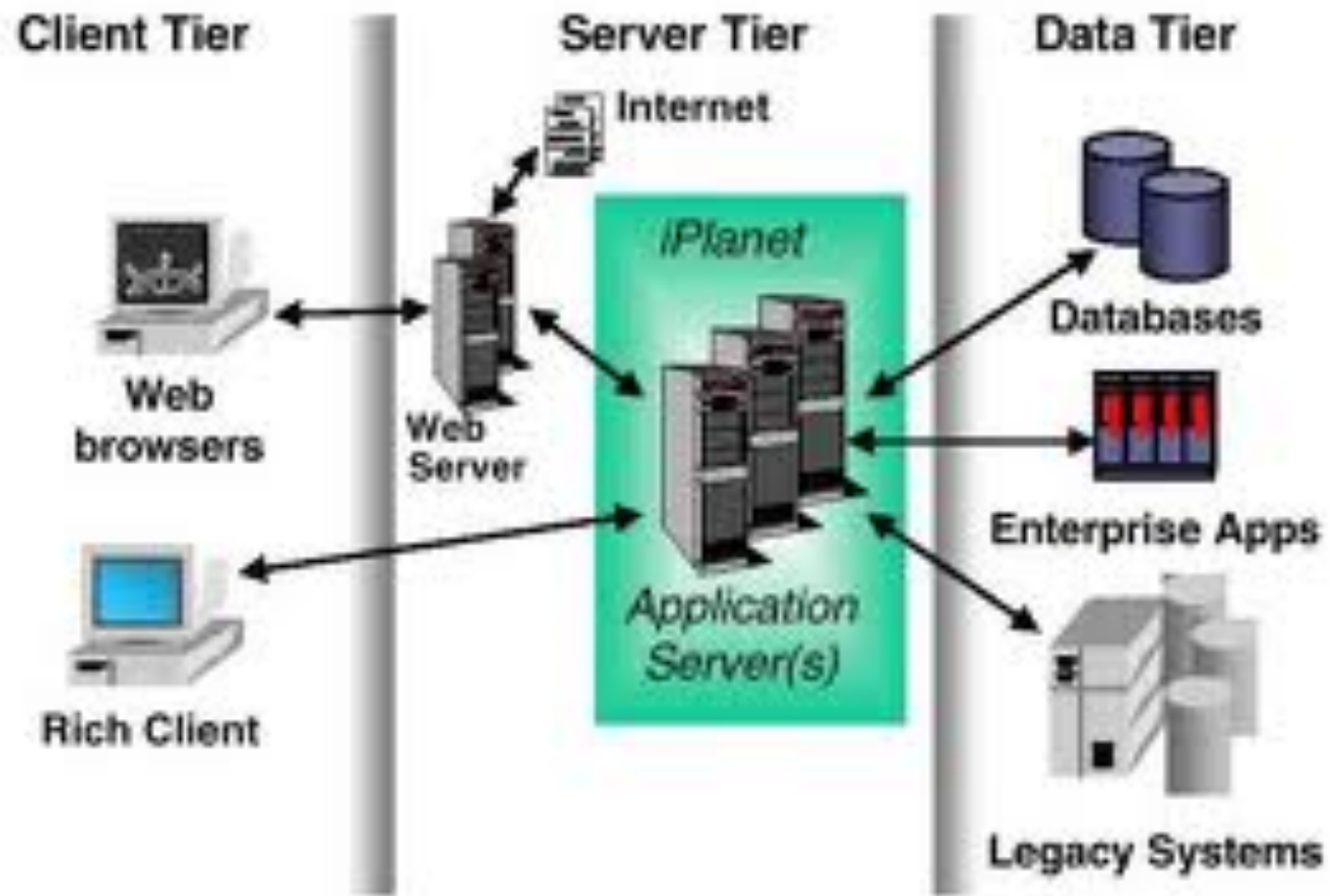
▶ 胖客户——C/S

▶ 瘦客户——B/S

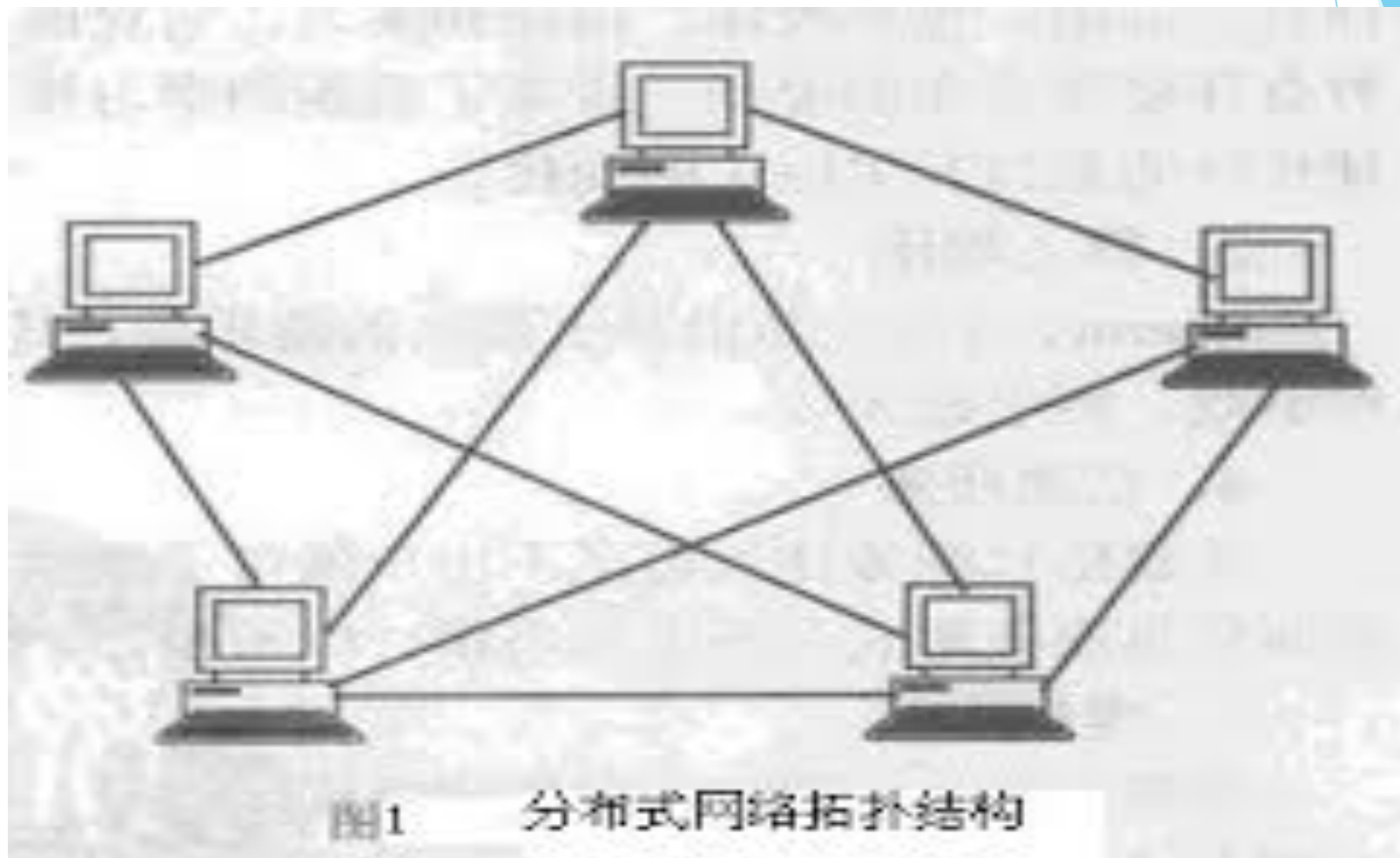
▶ 从两层到多层 (3-Tier、n-Tier)

▶ 分布式体系结构

基于TCP/IP协议的网络编程



基于TCP/IP协议的网络编程

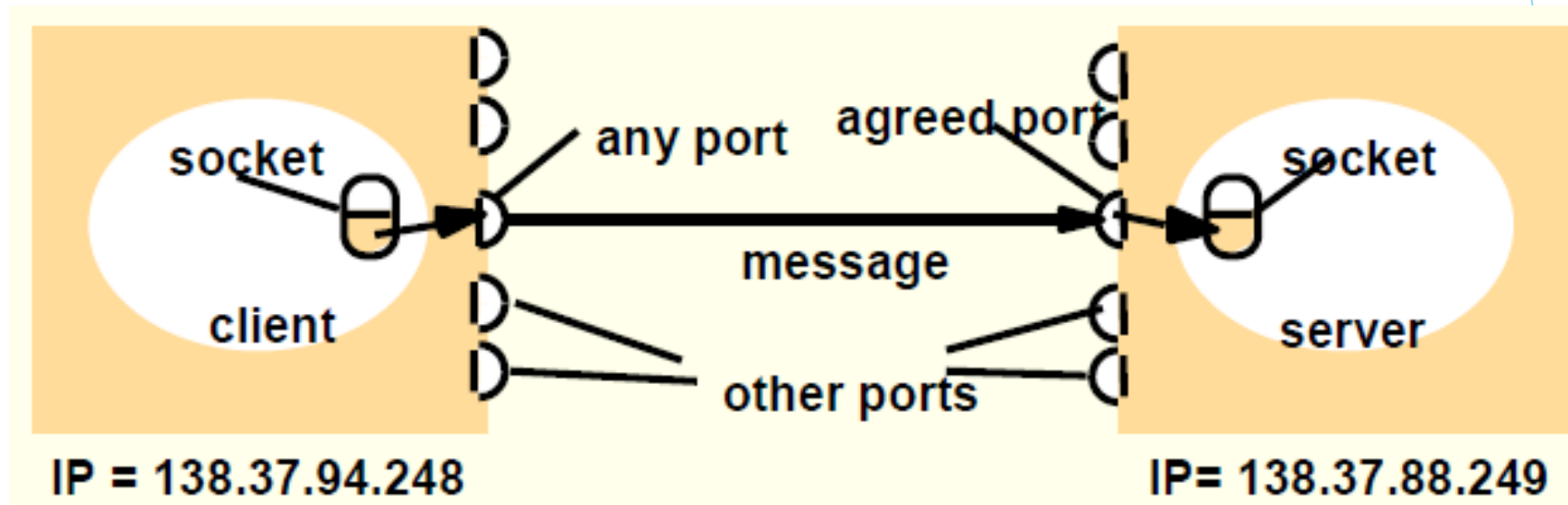


基于Socket的Java网络编程

▶ Socket通讯

- ▶ 网络上的两个程序通过一个双向的通讯连接实现数据的交换，这个双向链路的一端称为一个Socket。Socket通常用来实现客户方和服务方的连接。Socket是TCP/IP协议的一个十分流行的编程界面，一个Socket由一个IP地址和一个端口号唯一确定。
- ▶ 在Java环境下，Socket编程主要是指基于TCP/IP协议的网络编程。
- ▶ 传输层向应用层提供了套接字Socket接口，Socket封装了传输层细节，应用层的程序通过Socket来建立与远程主机的连接，以及进行数据传输

两个操作系统之间进程通信图示



基于Socket的Java网络编程

1、基本 类介绍

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**类和在服务器端的**ServerSocket**类。

- 在服务器端通过指定一个用来等待的连接的端口号创建一个 **ServerSocket**实例。
- 在客户端通过规定一个主机和端口号创建一个 **socket**实例，连到服务器上。
- **ServerSocket**类的**accept**方法使服务器处于阻塞状态，等待用户请求。

基于Socket的Java网络编程

2、基本编程步骤介绍

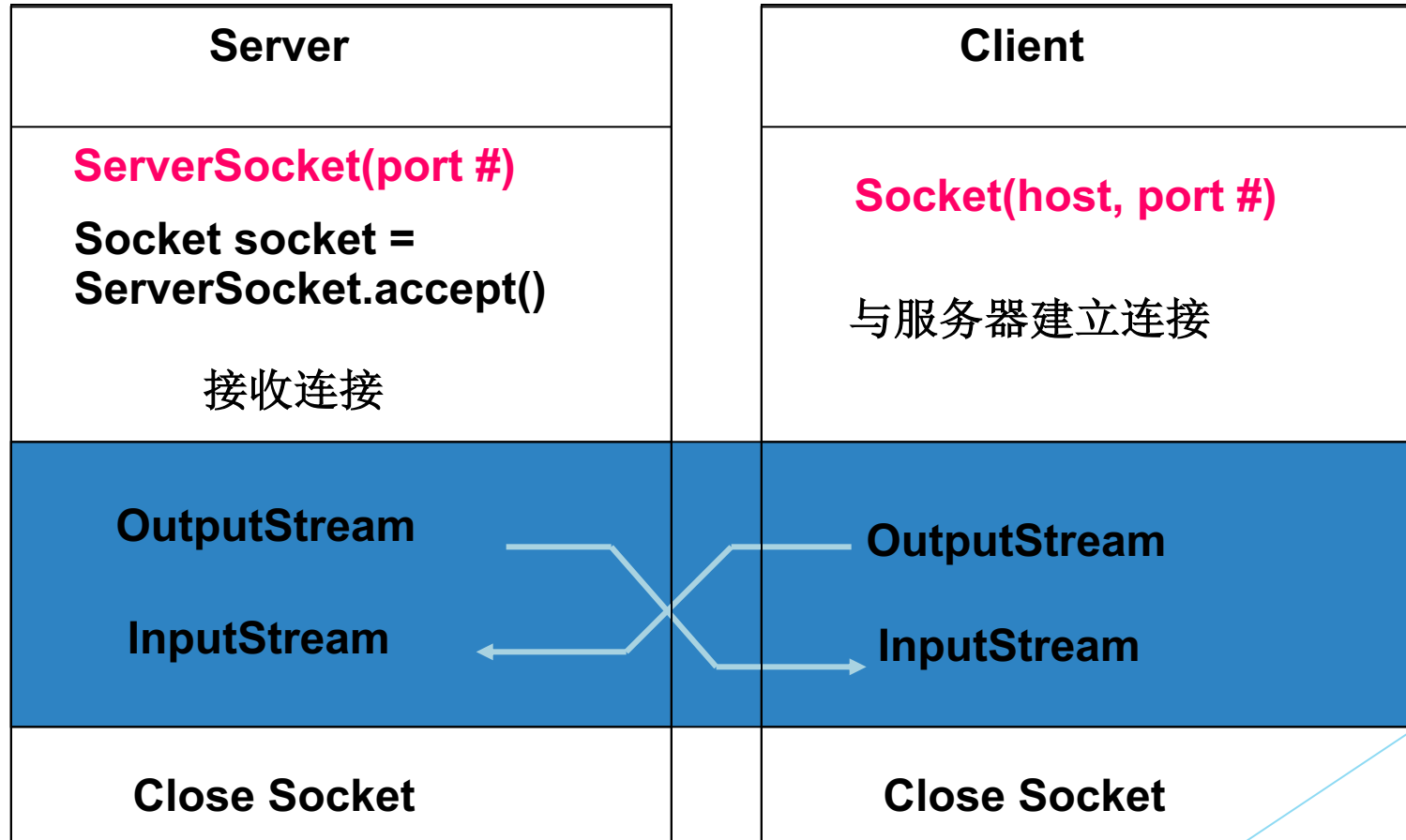
无论一个**Socket**通信程序的功能多么齐全、程序多么复杂，其基本结构都是一样的，都包括以下四个基本步骤：

- 1、在客户方和服务器方创建**Socket/ServerSocket**。
- 2、打开连接到**Socket**的输入/输出流。
- 3、利用输入/输出流，按照一定的协议对**Socket**进行读/写操作。
- 4、关闭输入/输出流和**Socket**。

通常，程序员的主要工作是针对所要完成的功能在第**3**步进行编程，第**1**、**2**、**4**步对所有的通信程序来说几乎都是一样的。

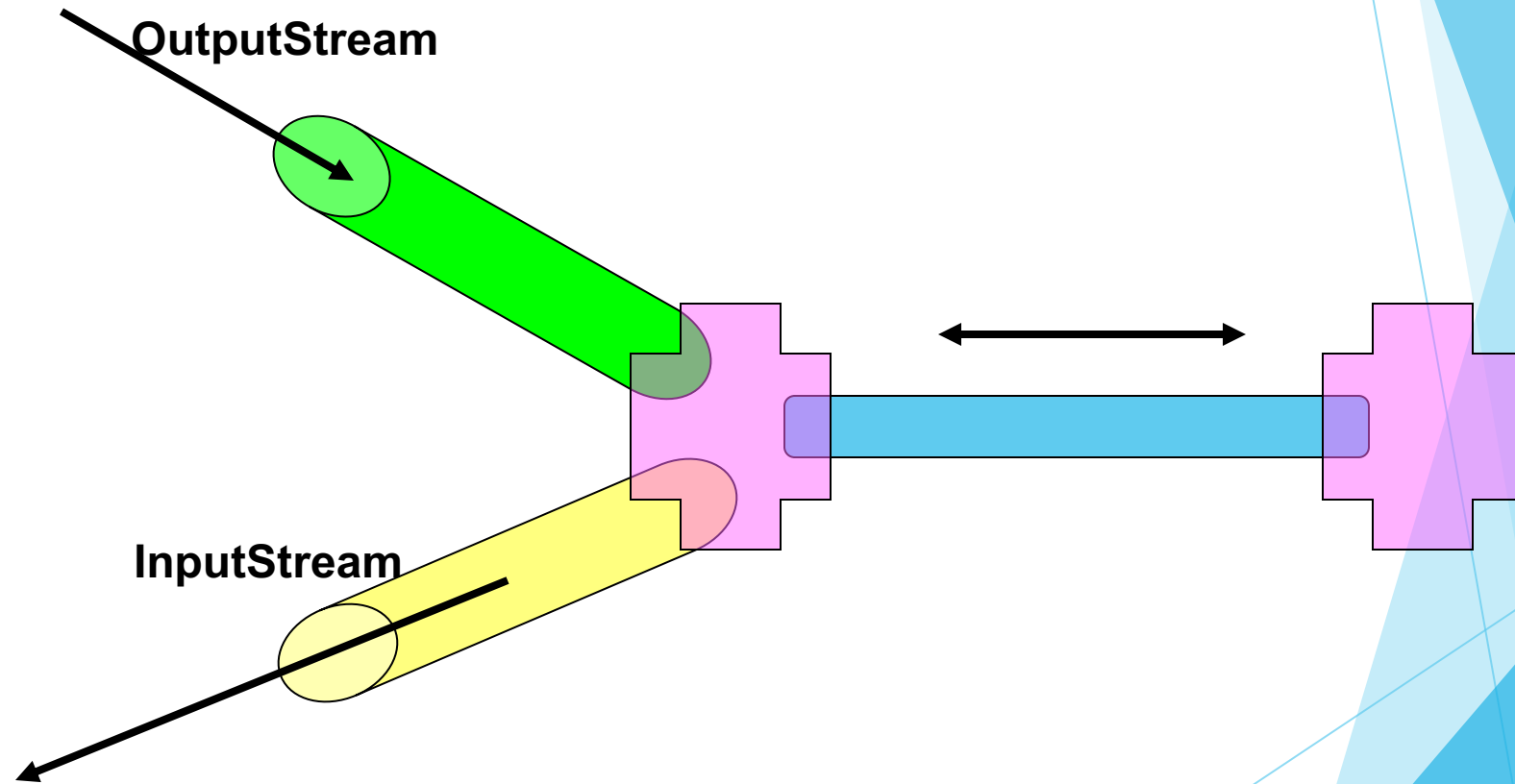
基于Socket的Java网络编程

3、客户程序和服务程序的基本通信机制



基于Socket的Java网络编程

4、Socket与I/O流



- ▶ 一个socket可以持有两个流—输入流与输出流

Socket通讯的一般过程

- ▶ Server端Listen(监听)某个端口是否有连接请求, Client端向Server端发出Connect(连接)请求, Server端向Client端发回Accept(接受)消息。一个连接就建立起来了。Server端和Client端都可以通过Send, Write等方法与对方通信。
- ▶ 对于一个功能齐全的Socket, 都要包含以下基本结构, 其工作过程包含以下四个基本的步骤:
 - (1) 创建Socket;
 - (2) 打开连接到Socket的输入/出流;
 - (3) 按照一定的协议对Socket进行读/写操作;
 - (4) 关闭Socket.

1、创建Socket

- ▶ java在包java.net中提供了两个类Socket和ServerSocket，分别用来表示双向连接的客户端和服务端。这是两个封装得非常好的类，使用很方便：
 - ▶ `Socket client = new Socket("127.0.0.1", 80);`
 - ▶ `ServerSocket server = new ServerSocket(80);`
- ▶ 每一个端口提供一种特定的服务，只有给出正确的端口，才能获得相应的服务。
- ▶ 在创建socket时如果发生错误，将产生IOException

1.1 客户端的Socket

▶ 典型的创建客户端Socket的过程

```
1. try{
2.     Socket socket=new Socket("127.0.0.1", 5000);
3.     //127.0.0.1是TCP/IP协议中默认的本机地址
4.     }catch(IOException e){
5.     System.out.println("Error:"+e);
6.     }
```

1.2服务器端的ServerSocket

▶ 典型的创建Server端ServerSocket的过程。

1. ServerSocket server=null;
2. try {
3. server=new ServerSocket(5000);
4. //创建一个ServerSocket在端口5000监听客户请求
5. }catch(IOException e){}
6. Socket socket=null;
7. try {
8. socket=server.accept();
9. //accept()是一个阻塞的方法，一旦有客户请求，它就会返回一个Socket对象用于同客户进行交互。该Socket对象绑定了客户程序的IP地址或端口号。
10. }catch(IOException e){}}

2、打开输入/输出流

- ▶ 输入输出流是网络编程的实质性部分

类Socket提供了方法getInputStream ()和getOutputStream()来得到对应的输入/输出流以进行读/写操作，这两个方法分别返回InputStream和OutputSteam类对象。为了便于读/写数据，我们可以在返回的输入/输出流对象上建立过滤流，如DataInputStream、DataOutputStream或PrintStream类对象，对于文本方式流对象，可以采用InputStreamReader和OutputStreamWriter、PrintWirter等处理

- ▶ `PrintWriter out=new PrintWriter(socket.getOutputStream(),true);`
- ▶ `BufferedReader in=new ButfferedReader(new InputSteramReader(Socket.getInputStream()));`

3、关闭Socket

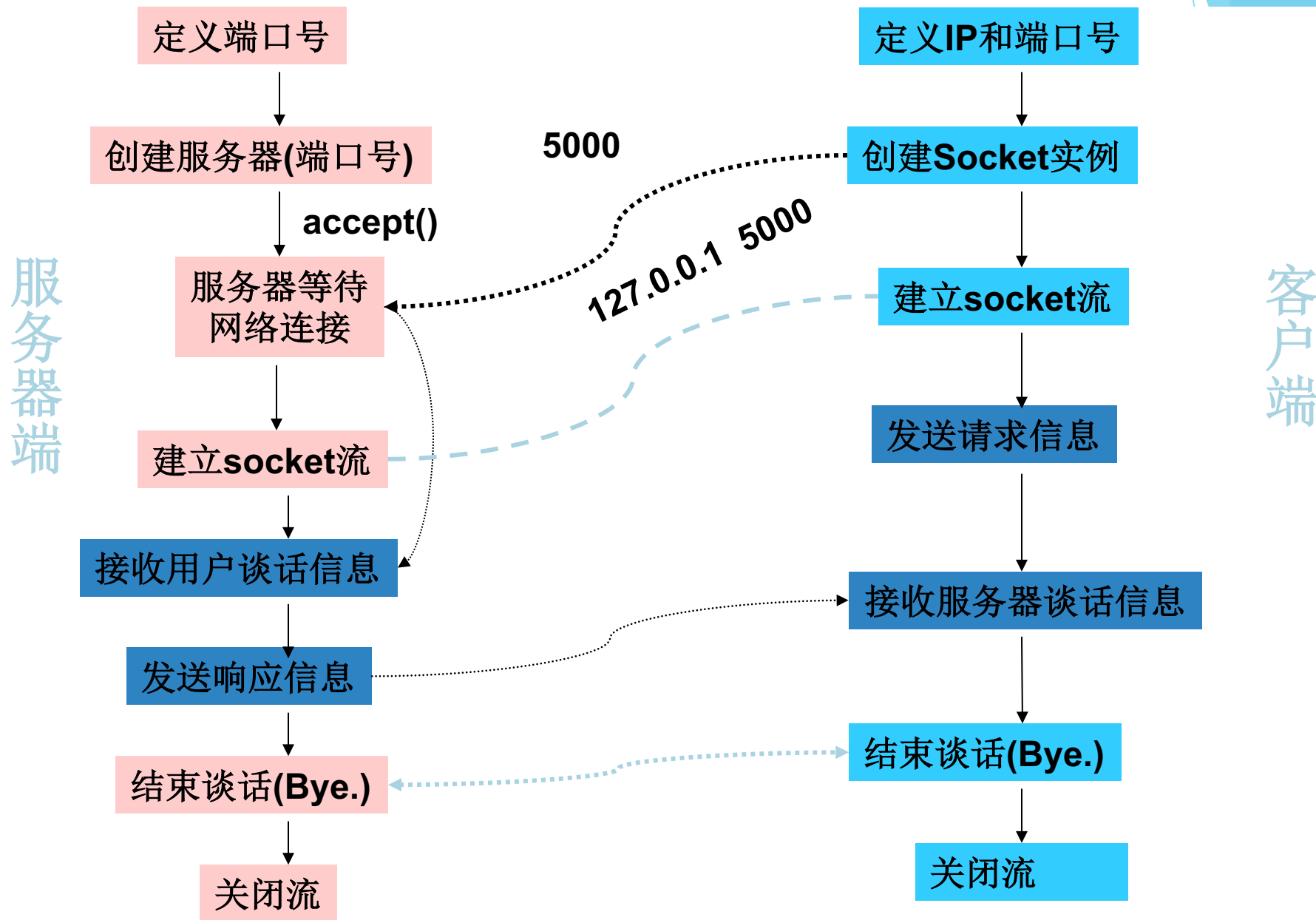
- ▶ 每一个Socket存在时，都将占用一定的资源，在Socket对象使用完毕时，要其关闭。关闭Socket可以调用Socket的Close（）方法。在关闭Socket之前，应将Socket相关的所有的输入/输出流全部关闭，以释放所有的资源。而且要注意关闭的顺序，与Socket相关的所有的输入/输出该首先关闭，然后再关闭Socket。

```
os.close();
```

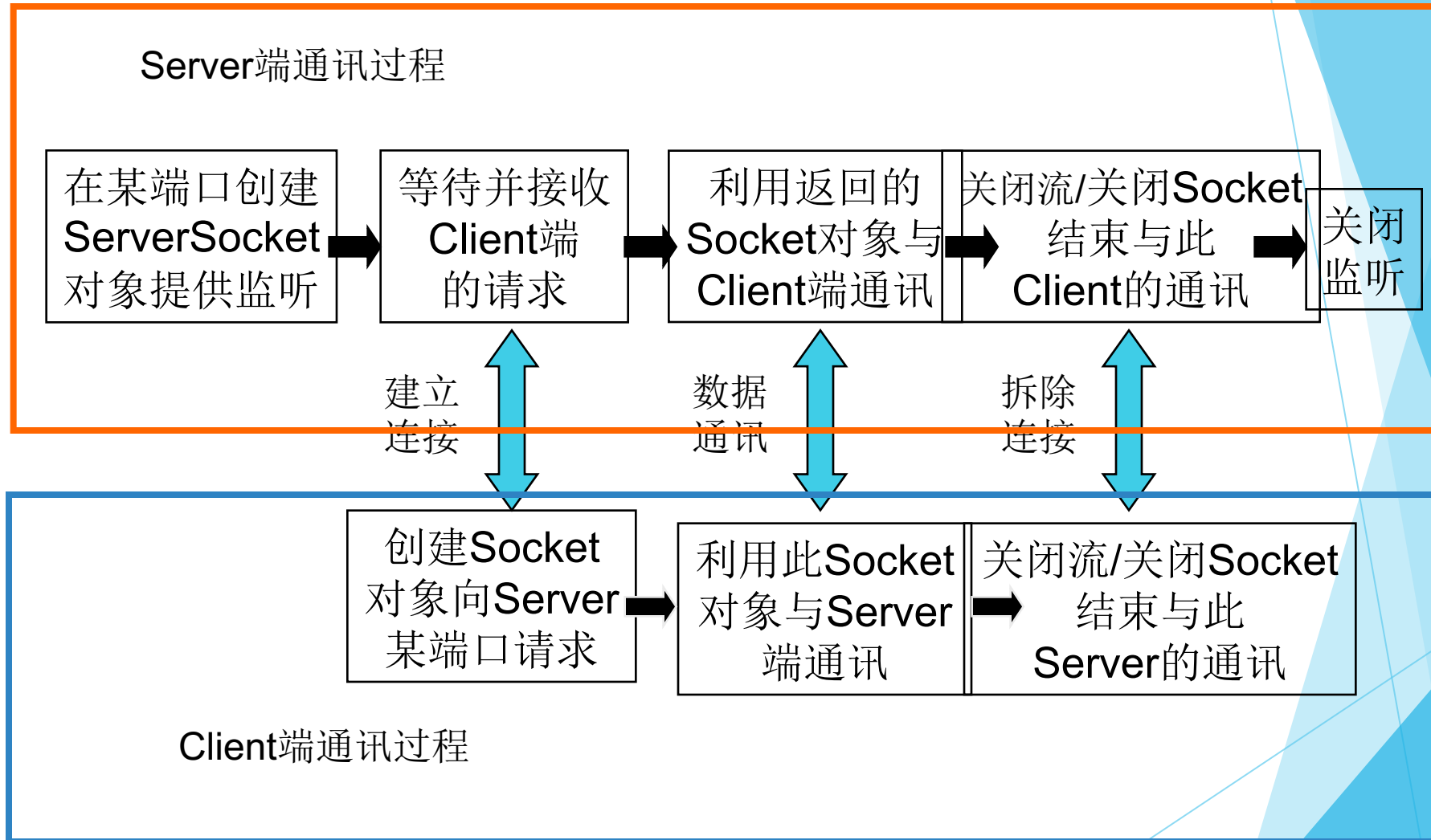
```
is.close();
```

```
socket.close();
```

socket通信过程图示1



socket通信过程图示2



本章内容

1. 网络编程基础知识
2. java的基本网络支持
3. 基于TCP协议的网络编程
4. 一个具体编程实例
5. 大作业：珠海浮生记网络游戏

简单的Client/Server程序设计

- ▶ 下面是用Socket实现的客户和服务端交互的典型的C/S结构的演示程序，通过仔细阅读该程序，会对前面所讨论的各个概念有更深刻的认识。程序的意义请参考注释。
- ▶ 客户端程序
- ▶ 服务器端程序

简单的服务器端程序

```
1. import java.io.*;
2. import java.net.*;
3. public class EchoServer {
4.     private int port=5000;
5.     private ServerSocket serverSocket;

6.     public EchoServer() throws IOException {
7.         serverSocket = new ServerSocket(port);
8.         System.out.println("服务器启动");
9.     }
```

简单的服务器端程序

```
1. public String echo(String msg) {
2.     return "echo:" + msg;
3. }
4. // 获得serverSocket的输出流
5. private PrintWriter getWriter(Socket socket)throws
   IOException{
6.     OutputStream socketOut = socket.getOutputStream();
7.     return new PrintWriter(socketOut,true);
8. }
9. // 获得serverSocket的输入流
10. private BufferedReader getReader(Socket socket)throws
   IOException{
11.     InputStream socketIn = socket.getInputStream();
12.     return new BufferedReader(new
   InputStreamReader(socketIn));
13. }
```

简单的服务器端程序

1. `public void service() { Socket socket=null;`
2. `while (true) {`
3. `try {`
4. `socket = serverSocket.accept();` //等待客户连接
5. `System.out.println("New connection accepted "`
`+socket.getInetAddress() + ":" socket.getPort());`
6. `BufferedReader br =getReader(socket);`//读取端口
的输入信息
7. `PrintWriter pw = getWriter(socket);`//得到输出对象
8. `String msg = null;`

简单的服务器端程序

```
1. while ((msg = br.readLine()) != null) {
2.     System.out.println(msg);
3.     pw.println(echo(msg)); // 向客户端输出响应信息
4.     if (msg.equals("bye")) // 如果客户发送的消息为
        “bye”，就结束通信
5.         break;     }
6.     }catch (IOException e) {}
7.     finally {     try{
8.         if(socket!=null)socket.close(); // 断开连接
9.     }catch (IOException e) { }
10.    } } }
11. public static void main(String args[])throws IOException
    { // 启动服务
12.     new EchoServer().service();
13. }
```

简单的客户端程序

```
1. import java.util.*;
2. public class EchoClient {
3.     private String host="localhost";
4.     private int port=5000;
5.     private Socket socket;
6.
7.     public EchoClient()throws IOException{
8.         socket=new Socket(host,port);
9.     }
10.    public static void main(String args[])throws IOException{
11.        new EchoClient().talk();
12.    }
```

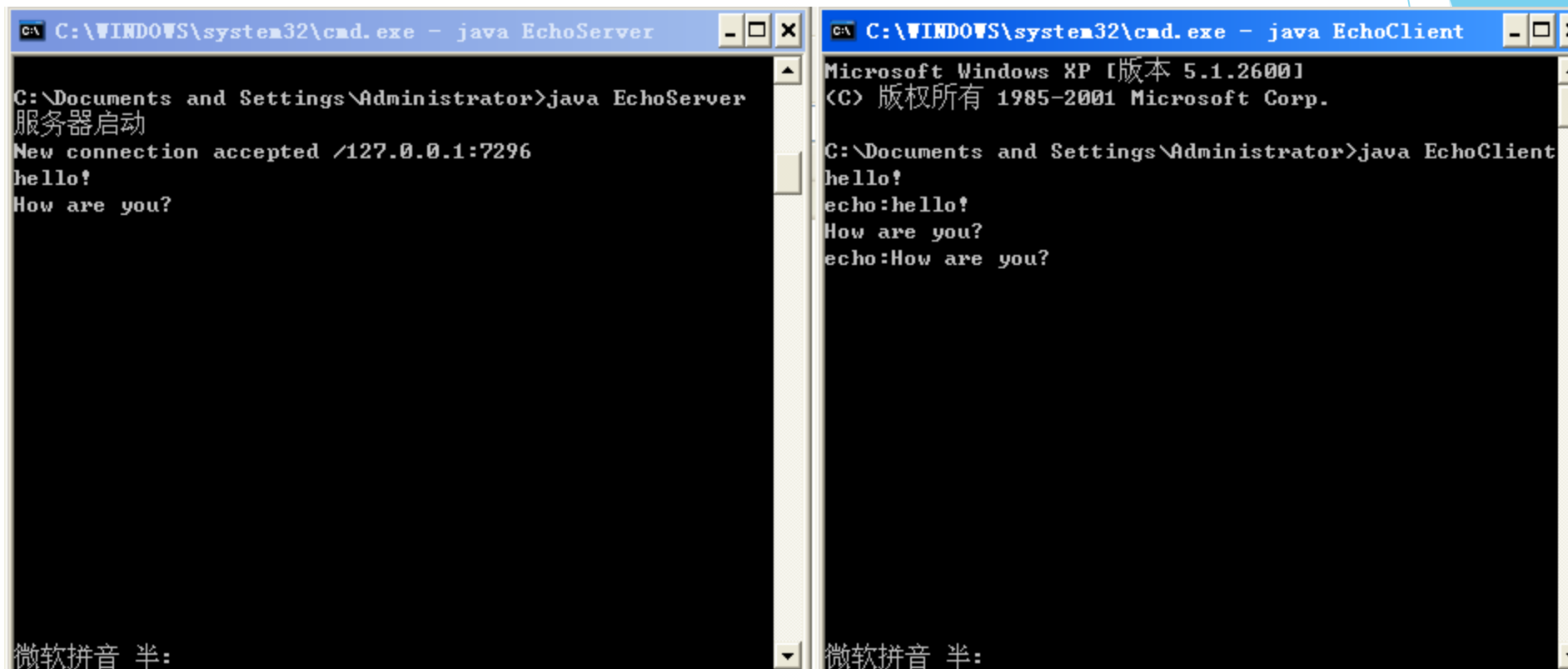
简单的客户端程序

1. `private PrintWriter getWriter(Socket socket) throws IOException{`
2. `OutputStream socketOut = socket.getOutputStream();`
3. `return new PrintWriter(socketOut,true);`
4. `}`
5. `private BufferedReader getReader(Socket socket) throws IOException{`
6. `InputStream socketIn = socket.getInputStream();`
7. `return new BufferedReader(new InputStreamReader(socketIn));`
8. `}`

简单的客户端程序

```
1. public void talk()throws IOException {
2.     try{
3.         BufferedReader br=getReader(socket);
4.         PrintWriter pw=getWriter(socket);
5.         BufferedReader localReader=new BufferedReader(new
InputStreamReader(System.in));
6.         String msg=null;
7.         while((msg=localReader.readLine())!=null){
8.             pw.println(msg);
System.out.println(br.readLine());
9.             if(msg.equals("bye"))
10.                break;
11.        } }catch(IOException e){
12.    }finally{
13.        try{socket.close();}catch(IOException e){ } }
```

执行效果



The image displays two side-by-side Windows command prompt windows. The left window, titled "C:\WINDOWS\system32\cmd.exe - java EchoServer", shows the execution of the Java EchoServer program. The output includes "服务器启动" (Server started), "New connection accepted /127.0.0.1:7296", and the prompts "hello!" and "How are you?". The right window, titled "C:\WINDOWS\system32\cmd.exe - java EchoClient", shows the execution of the Java EchoClient program. The output includes the Windows XP version information, the copyright notice, and the prompts "hello!" and "How are you?". Both windows show the text "微软拼音 半:" at the bottom, indicating the input method.

```
C:\WINDOWS\system32\cmd.exe - java EchoServer
C:\Documents and Settings\Administrator>java EchoServer
服务器启动
New connection accepted /127.0.0.1:7296
hello!
How are you?
微软拼音 半:
```

```
C:\WINDOWS\system32\cmd.exe - java EchoClient
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.
C:\Documents and Settings\Administrator>java EchoClient
hello!
echo:hello!
How are you?
echo:How are you?
微软拼音 半:
```

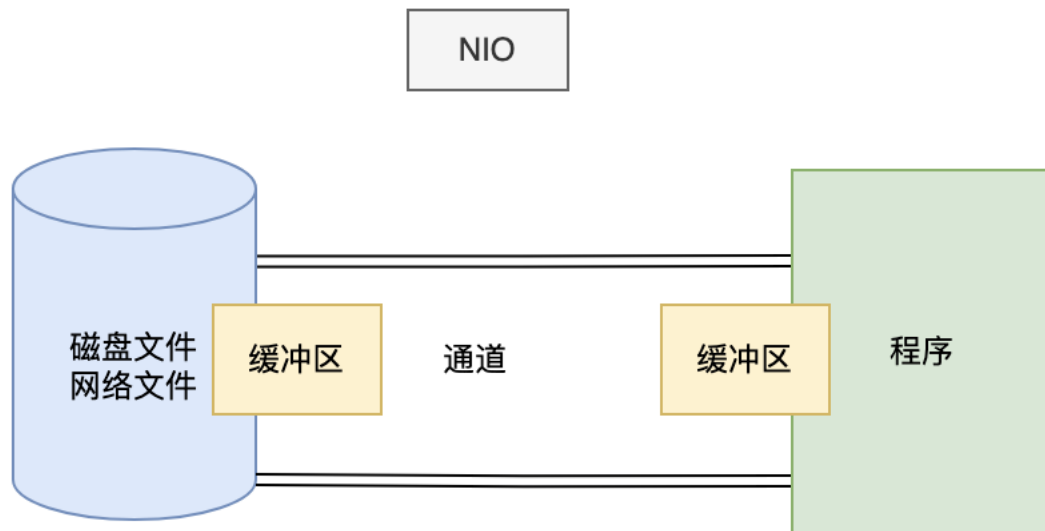
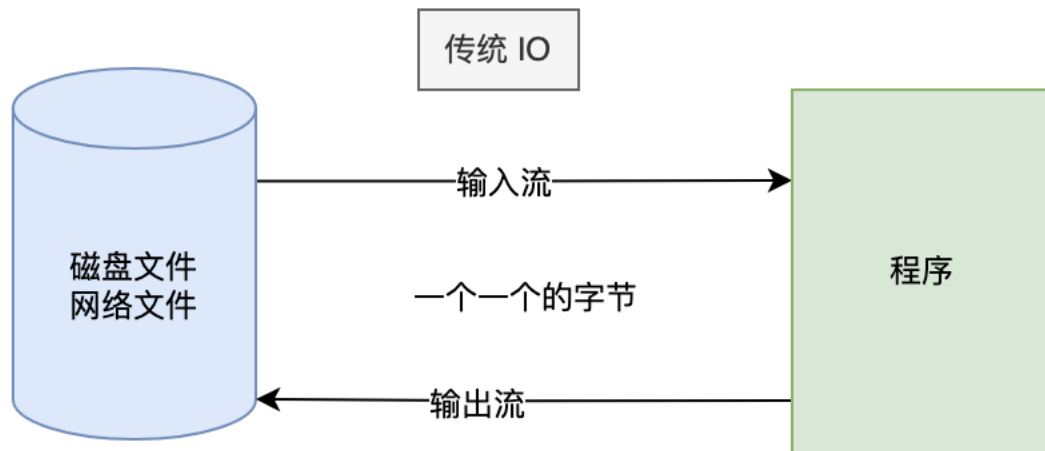
加入多线程支持

- ▶ 实际应用中的客户端则可能需要和服务器端保持长时间通信，即服务器需要不断地读取客户端数据，并向客户端写入数据；客户端也需要不断地读取服务器数据并向服务器写入数据。
- ▶ 使用传统BufferedReader的readLine()方法读取数据时，当该方法成功返回之前，线程被阻塞，程序无法继续执行。考虑到这个原因，因此服务器应该每个Socket单独启动一条线程，每条线程负责与一个客户端进行通信。
- ▶ 客户端读取服务器数据的线程同样会被阻塞，所以系统应该单独启动一条线程，该线程专门负责读取服务器数据。

Java的NIO为非阻塞式的Socket通信提供了如下几个特殊类：

- ▶ Selector: 它是SelectableChannel对象的多路复用器，所有希望采用非阻塞方式进行通信的Channel都应该注册到Selector对象。可通过调用此类的静态open()方法来创建Selector实例，该方法将使用系统默认的Selector来返回新的Selector。
- ▶ SelectableChannel: 它代表可以支持非阻塞IO操作的Channel对象，可以将其注册到Selector上，这种注册的关系由SelectionKey实例表示。Selector对象提供了一个select()方法，该方法允许应用程序同时监控多个IO_Channel。
- ▶ SelectionKey: 该对象代表SelectableChannel和Selector之间的注册关系。
ServerSocketChannel: 支持非阻塞操作，对应于ServerSocket这个类，提供了TCP协议IO接口，只支持OP_ACCEPT操作。该类也提供了accept()方法，功能相当于ServerSocket提供的accept()方法。
- ▶ SocketChannel: 支持非阻塞操作，对应于Socket这个类，提供了TCP协议IO接口，支持OP_CONNECT, OP_READ和OP_WRITE操作。这个类还实现了ByteChannel接口、ScatteringByteChannel接口和GatheringByteChannel接口，所以可以直接通过SocketChannel来读写ByteBuffer对象。

NIO的非阻塞通信



NIO的非阻塞通信

- ▶ 举例 <https://www.linuxidc.com/Linux/2014-11/109779.htm>

UDP

- ▶ **UDP**，全称User Datagram Protocol(用户数据报协议)，是Internet 协议集支持一个无连接的传输协议。**UDP** 为应用程序提供了一种无需建立连接就可以发送封装的 IP 数据包的方法。
- ▶ **UDP**主要用于不要求分组顺序到达的传输中，分组传输顺序的检查与排序由应用层完成，提供面向报文的简单不可靠信息传送服务。**UDP** 协议基本上是IP协议与上层协议的接口，适用端口分别运行在同一台设备上的多个应用程序。

UDP的特点(与TCP相比)

- ▶ 正是UDP提供不可靠服务，具有了TCP所没有的优势。无连接使得它具有资源消耗小，处理速度快的优点，所以音频、视频和普通数据在传送时经常使用UDP，偶尔丢失一两个数据包，也不会对接收结果产生太大影响。
- ▶ UDP有别于TCP，有自己独立的套接字(IP+Port)，它们的端口号不冲突。和TCP编程相比，UDP在使用前不需要进行连接，没有流的概念。
- ▶ 如果说TCP协议通信与电话通信类似，那么UDP通信就与邮件通信类似：不需要实时连接，只需要目的地址；
- ▶ UDP通信前不需要建立连接，只要知道地址（ip地址和端口号）就可以给对方发送信息；
- ▶ 基于用户数据报文（包）读写；
- ▶ UDP通信一般用于线路质量好的环境，如局域网内，如果是互联网，往往应用于对数据完整性不是过于苛刻的场合，例如语音传送等。
- ▶ 以上是对UDP的基本认识，与以前学习的理论相比，接下来的实践更加有趣，实践出真知。

发送报文

- ▶ `DatagramSocket()`: 创建一个`DatagramSocket`实例并将该对象绑定到本机默认IP地址、本机所有可用端口中随机选择的某个端口。
- ▶ `DatagramSocket(intport)`: 创建一个`DatagramSocket`实例，并将该对象绑定到本机默认IP地址、指定端口。
- ▶ `DatagramSocket(intport, InetAddressladdr)`: 创建一个`DatagramSocket`实例，并将该对象绑定到指定IP地址、指定端口。

简单的UDP聊天程序

- ▶ 实例, <https://www.jb51.net/article/45092.htm>