

Java语言程序设计

配套教材由清华大学出版社出版发行

第9章 网络编程



中國農業大學

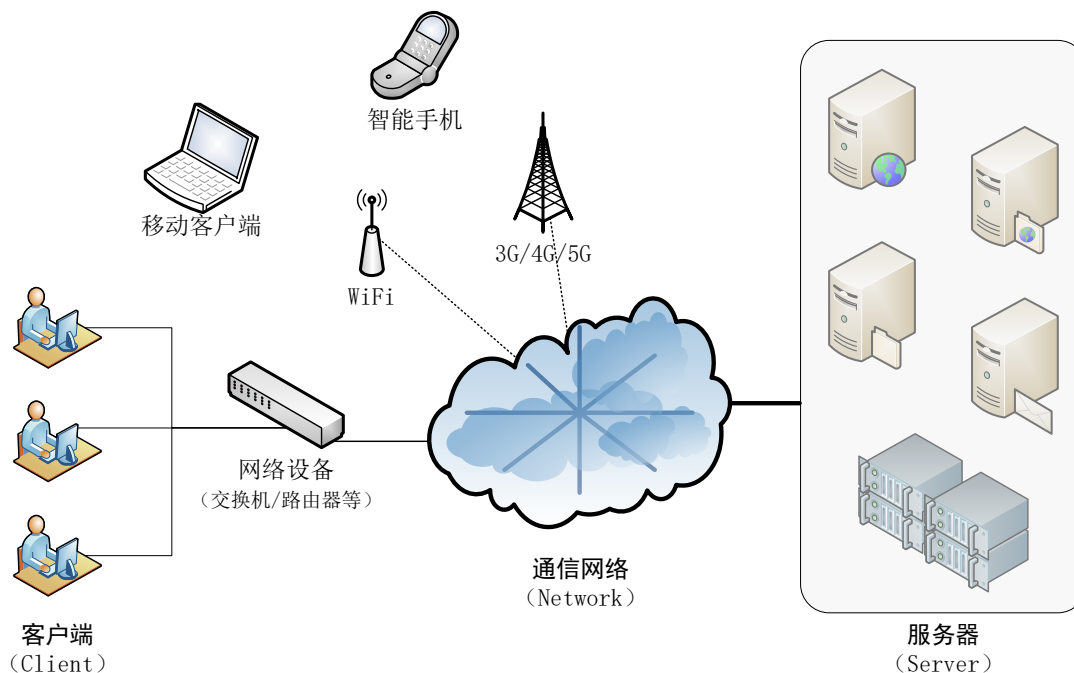
阚道宏

第9章 网络编程

- 计算机网络的全景图

一 三大要素

- 通信网络
- 服务器
- 客户端



中國農業大學

阚道宏

第9章 网络编程

- 计算机网络的三大要素

- 通信网络（network）

- 服务器

- 服务（service）

- 服务器（server）：硬件 + 软件

- 更强的计算能力和存储能力

- 性能更强、安全性更高的操作系统，例如Linux、Unix或Windows Server

- 服务器与普通计算机的最大区别：应用程序

- » WWW（World Wide Web，简称Web）服务（即网站服务）

- » email电子邮件服务（即收发电子邮件）

- » FTP（File Transfer Protocol）文件传输服务（即文件上传与下载）

- » 网络服务是通过服务器应用程序来提供的，不同网络服务需要不同的服务器应用程序



中國農業大學

閻道宏

第9章 网络编程

- 计算机网络由三大要素组成
 - 通信网络（network）
 - 服务器
 - 客户端（client）
 - 客户端应用程序
 - 使用WWW网站服务需要Web客户端程序，例如IE浏览器
 - 使用email电子邮件服务需要Mail客户端程序，例如Outlook
 - 使用FTP文件传输服务需要FTP客户端程序
 - 计算机：浏览器（browser）
 - 智能手机App：Application（应用）的昵称
 - C/S架构网络服务：客户端应用程序 + 服务器应用程序



第9章 网络编程

- 本章内容
 - [9.1 计算机网络的基本原理](#)
 - [9.2 网络服务与网络资源](#)
 - [9.3 程序之间的网络通信](#)
 - [9.4 基于UDP协议的网络通信](#)



9.1 计算机网络的基本原理

- 计算机网络

- TCP/IP协议

- <http://www.cau.edu.cn>
- 浏览器获取服务器信息
 - 客户端应用程序与服务器应用程序的通信
- 计算机网络中的协议（protocol）
 - **RFC**（Request For Comments）
 - 传输控制协议（Transmission Control Protocol，简称**TCP**）
 - 网间互连协议（Internet Protocol，简称**IP**）
 - **TCP/IP**协议



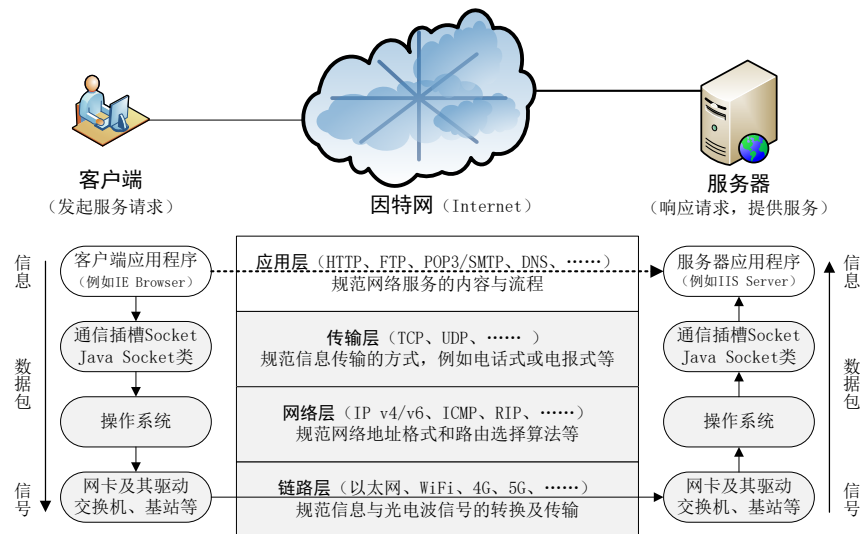
中國農業大學

閻道宏

9.1 计算机网络的基本原理

- TCP/IP网络四层模型

- 应用层
- 传输层
- 网络层
- 链路层



9.1 计算机网络的基本原理

- TCP/IP网络四层模型
 - 应用层（application layer）
 - 与Web服务相关的应用层协议

- HTTP

GET /index.html HTTP/1.1

Accept: */*

Accept-Language: zh-CN

Host: localhost:8000

Connection: Keep-Alive

HTTP/1.1 200 OK

Content-Type: text/html; charset=UTF-8

<html>

<head> <title>Web服务HTML测试页面</title> </head>

<body> <p>Hello World!</p> </body>

<body> <p>你好，世界！</p> </body>

</html>

- HTML

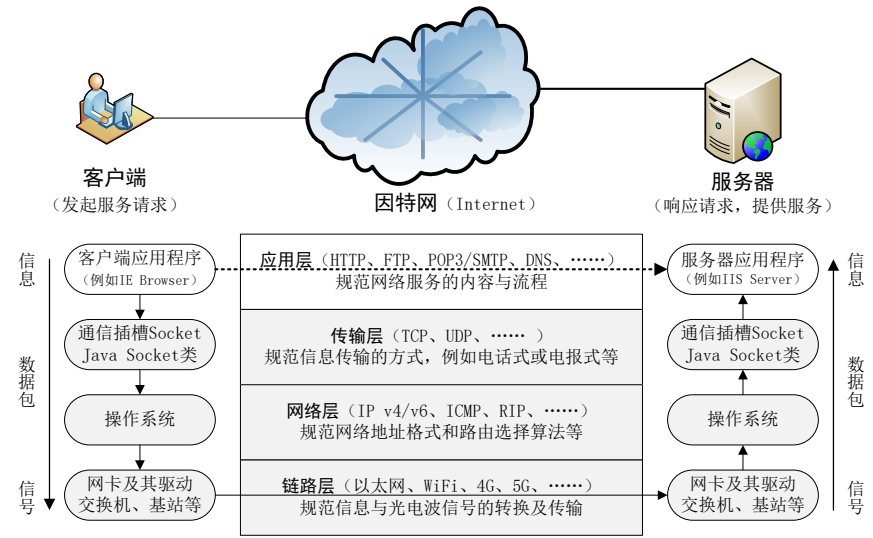


中國農業大學

閻道宏

9.1 计算机网络的基本原理

- TCP/IP网络四层模型
 - 应用层（application layer）
 - DNS域名系统
 - 主机（host）
 - 主机名（host name）
 - 服务器的域名（domain name）
 - www.baidu.com
 - www.oracle.com
 - www.cau.edu.cn
 - www.icourse163.org
 - localhost



- 域名便于人的记忆，但计算机网络内部使用的则是数值形式的网络地址，称作IP地址
- DNS（Domain Name System）域名系统是一个关于域名的应用层协议，其中规范了域名的格式、域名与IP地址之间如何映射等

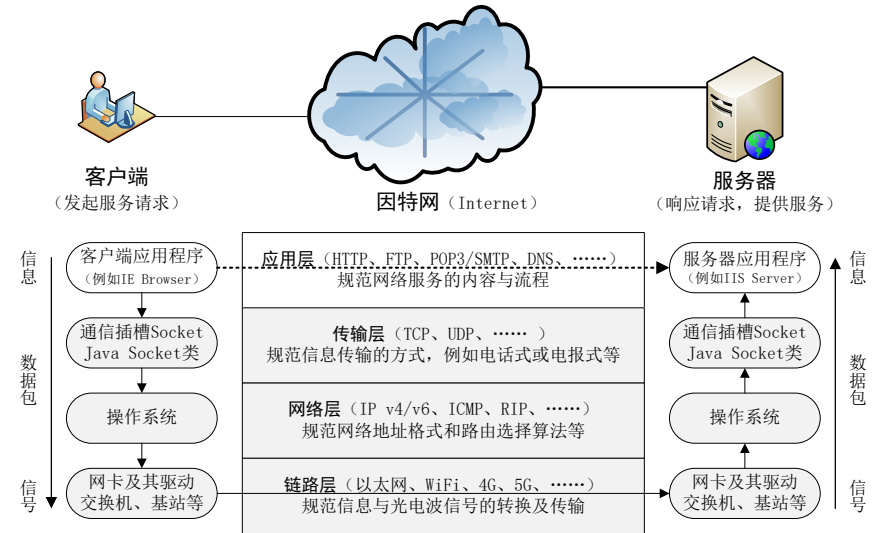


中国农业大学

阚道宏

9.1 计算机网络的基本原理

- TCP/IP网络四层模型
 - 应用层 (application layer)



- 其他应用层协议
 - **FTP** (File Transfer Protocol) 文件传输协议
 - **SMTP** (Simple Mail Transfer Protocol) 简单邮件传输协议
 - **POP3** (Post Office Protocol 3) 邮局协议 (第3版)
- 程序员与应用层协议的关系
 - 编写Web服务等**通用**网络服务程序, **程序员**首先应当了解TCP/IP网络中相关的应用层协议, 然后按照协议要求编写程序
 - 编写自己**专有**的网络服务程序, **程序员**可以制定自己的应用层协议

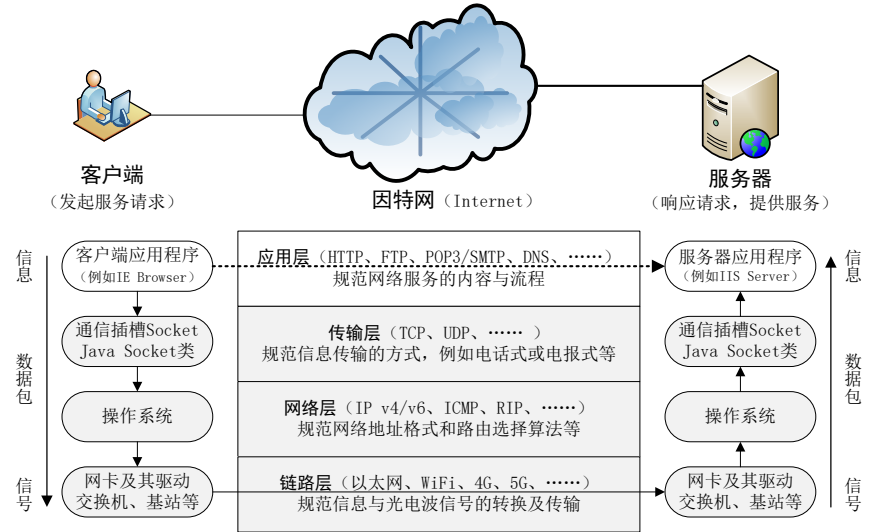


中國農業大學

阚道宏

9.1 计算机网络的基本原理

- TCP/IP网络四层模型
 - **传输层**（transport layer）
 - 有连接通信TCP
 - 无连接通信UDP



— TCP和UDP

- **TCP**: 通信双方先建立**连接** (connection), 然后再进行双向数据传输。
传输控制协议 (Transmission Control Protocol, 简称TCP)
- **UDP**: 直接将数据单向传输给对方, 不需要事先建立连接, 事后也不需要对方回复。
用户数据报协议 (User Datagram Protocol, 简称UDP)
- TCP可靠, UDP不是百分之百可靠
- **发送方** (sender)、**接收方** (receiver)
- 网络通信的**本质**是网络应用程序之间的数据传输

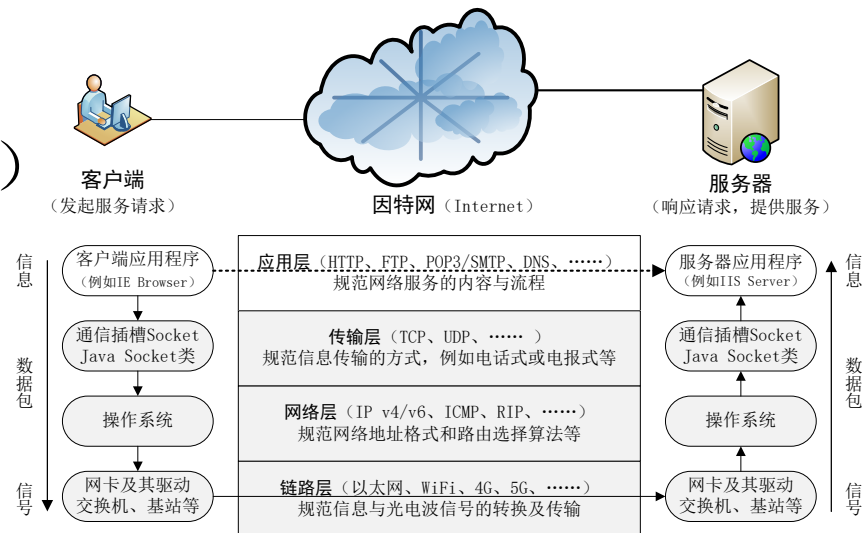


中國農業大學

閻道宏

9.1 计算机网络的基本原理

- TCP/IP网络四层模型
 - 传输层 (transport layer)



– 传输层与应用层的关系

- 应用层：信息
- 传输层：数据，数据包 (**packet**)



中國農業大學

閻道宏

9.1 计算机网络的基本原理

- TCP/IP网络四层模型

- 传输层 (transport layer)

- 端口 (port)

- 多个网络应用程序如何**共用**一条物理链路?

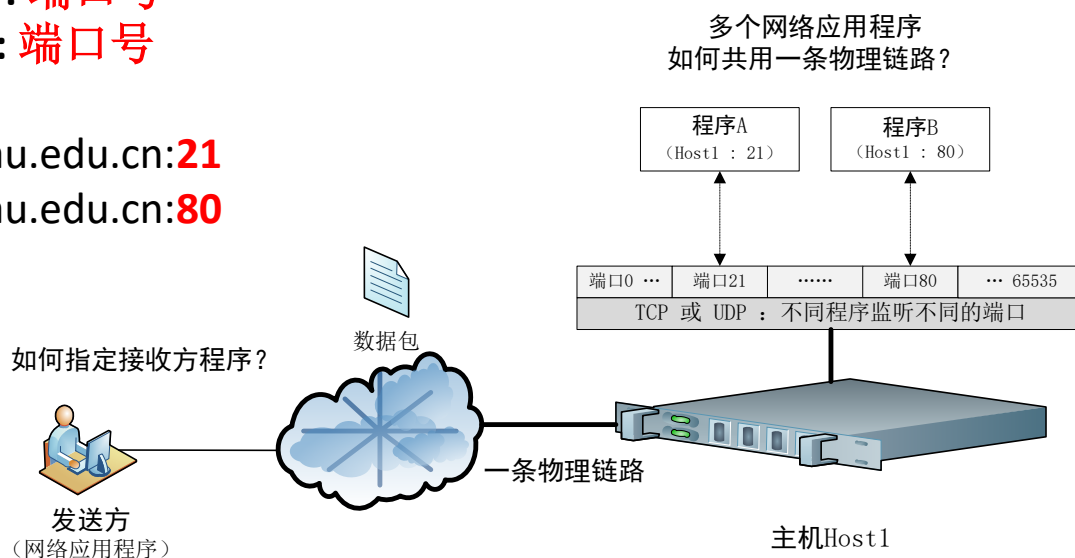
- 发送方如何**指定**接收方程序?

主机名: **端口号**

IP地址: **端口号**

www.cau.edu.cn:**21**

www.cau.edu.cn:**80**

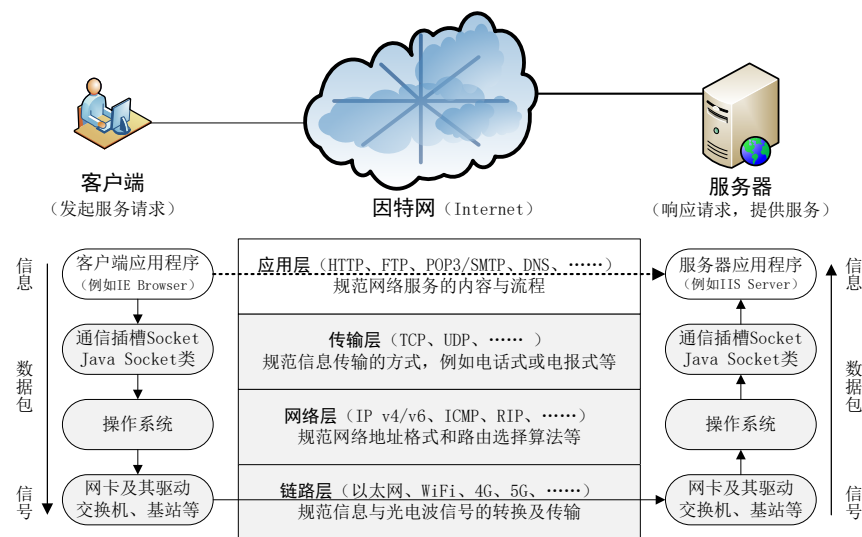


中國農業大學

閻道宏

9.1 计算机网络的基本原理

- TCP/IP网络四层模型
 - 传输层 (transport layer)



– 程序员与应用层、传输层的关系

- 应用层协议用于规范网络服务的内容及服务流程，而传输层协议用于规范发送方与接收方之间的通信方式
- 编写网络应用程序，**程序员**需按照应用层协议拟定通信内容，再根据传输层协议选择通信方式（例如TCP或UDP），将通信内容从**发送方**传输到**接收方**
- **接收方**程序也可能需要对接收到的通信内容进行**回复**。编写接收方程序的程序员也需要按照应用层协议的规定拟定回复内容，再通过传输层将回复内容传输给发送方



9.1 计算机网络的基本原理

- TCP/IP网络四层模型

- **网络层**（network layer）

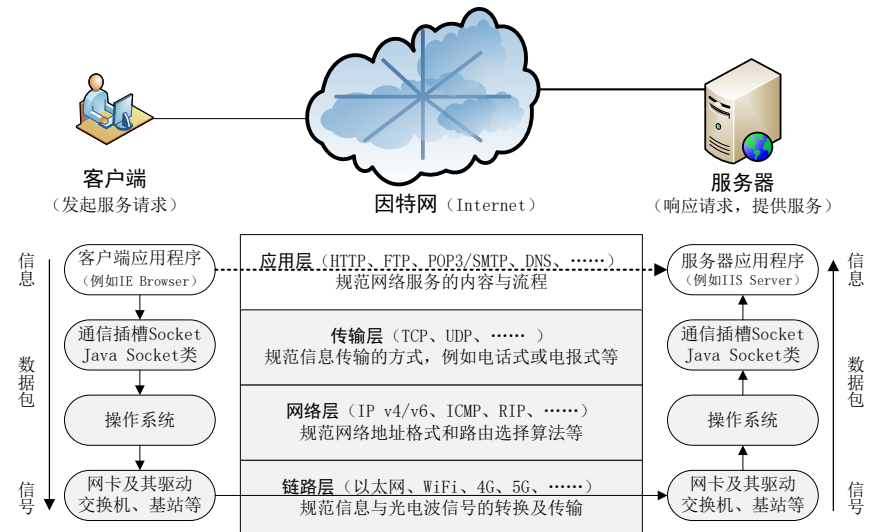
- IP、ICMP等

- **链路层**（link layer）

- 以太网、WiFi、4G、5G等

- **网络层：IP地址**

- **IPv4**地址：192.168.1.175
 - **IPv6**地址：ABCD:EF01:2345:6789:ABCD:EF01:2345:6789
 - **静态**或**动态**分配IP地址
 - 域名服务器（Domain Name Server，简称**DNS**）
 - **Java API**提供了一个因特网地址类**InetAddress**



9.1 计算机网络的基本原理

- 因特网地址类 **InetAddress**

java.net.InetAddress类说明文档			
public class InetAddress extends Object implements Serializable			
	修饰符	类成员（节选）	功能说明
1	static	InetAddress getLocalHost()	获取本机的因特网地址对象
2	static	InetAddress getByName (String host)	通过主机名创建因特网地址对象
3	static	InetAddress getByAddress (byte[] addr)	通过IP地址创建因特网地址对象
4		byte[] getAddress()	获取IP地址
5		String getHostAddress()	获取字符串形式的IP地址
6		String getHostName()	获取主机名
7		boolean isReachable (int timeout)	检查因特网地址是否可以连通
.....			



9.1 计算机网络的基本原理

- 因特网地址类 **InetAddress**

例9-1 一个因特网地址类InetAddress的Java演示程序（JInetAddressTest.java）

```
1 import java.net.*;      // 导入java.net网络包中的类
2 public class JInetAddressTest { // 测试类：测试因特网地址类
3     public static void main(String[] args) { // 主方法
4         try { // 处理可能出现的勾选异常UnknownHostException
5             InetAddress local = InetAddress.getLocalHost(); // 获取本机因特网地址对象
6             System.out.println("通过getLocalHost()获得本机因特网地址对象: " + local);
7             System.out.println("getHostName(): " + local.getHostName());
8             System.out.println("getHostAddress(): " + local.getHostAddress());
9             System.out.println();
10            // 下面演示域名与主机名、IP地址之间的关系
11            String cauWeb = "www.cau.edu.cn"; // 中国农业大学网站的主机名
12            InetAddress cau = InetAddress.getByName(cauWeb); // 根据主机名创建对象
13            System.out.println("根据主机名创建因特网地址对象: " + cau);
14            System.out.println("getHostName(): " + cau.getHostName()); // 主机名
15            System.out.println("getHostAddress(): " + cau.getHostAddress()); // IP地址
16        }
17        catch(UnknownHostException e) { e.printStackTrace(); } // 捕捉并处理勾选异常
18    } }
```

Problems @ Javadoc Declaration Console

<terminated> JInetAddressTest [Java Application] C:\Java\jre1.8.0

通过getLocalHost()获得本机因特网地址对象: Kan/192.168.1.7
getHostName(): Kan
getHostAddress(): 192.168.1.7

根据主机名创建因特网地址对象: www.cau.edu.cn/114.251.217.179
getHostName(): www.cau.edu.cn
getHostAddress(): 114.251.217.179



9.1 计算机网络的基本原理

- TCP/IP协议总结
 - TCP/IP协议是一组关于**网络服务**及**网络通信**的规范和标准
 - TCP/IP网络模型将TCP/IP协议按照功能划分成**四层**，它们分别是**应用层**、**传输层**、**网络层**和**链路层**
 - 编写**网络应用程序**主要涉及**应用层**和**传输层**协议，通常不会直接用到**网络层**或**链路层**协议。**程序员**应当了解**应用层**和**传输层**协议的主要内容和基本工作原理，特别是**传输层**的**TCP**协议和**UDP**协议
 - Java API提供了一组与**网络编程**相关的**类**，例如**网络层**的**因特网地址类**、**传输层**的**套接字类**、**应用层**的**统一资源定位符**



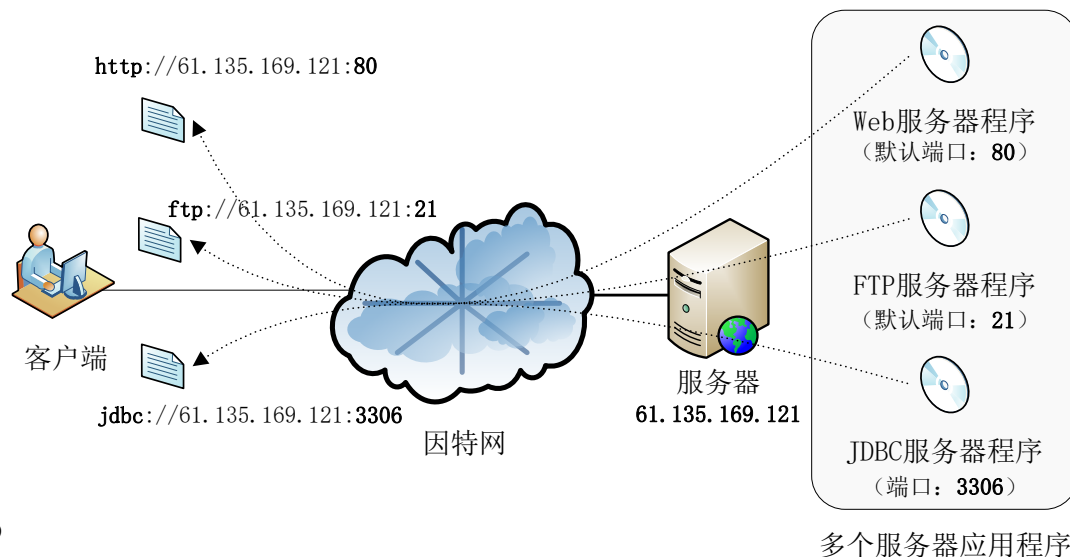
9.2 网络服务与网络资源

- 网络服务
 - Web服务、FTP文件传输服务、email电子邮件服务、JDBC数据库服务、Web Service服务等
 - 可以在一台**服务器**主机上**安装**多个服务器应用程序，这样就能用一台主机**同时**提供多个网络服务
 - **服务器**可以通过**不同端口**向外界提供多个网络服务



9.2 网络服务与网络资源

- 网络服务
 - 网络服务的**端口**



- 端口号: 0~65535
- **公认**端口: 0~1023
 - TCP **80**: Web服务
 - TCP **21**和**20**: FTP服务
 - TCP **25**和**110**: email服务
 -
- **程序员**: 1024~49151

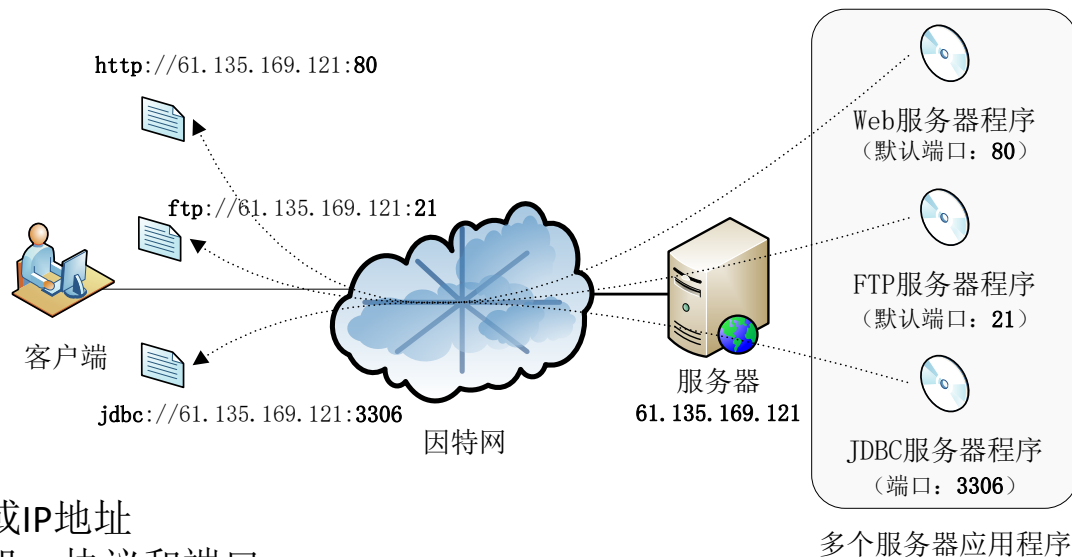


中國農業大學

阚道宏

9.2 网络服务与网络资源

- 网络服务
 - 网络服务的地址



- **主机**地址: 主机名或IP地址
- **网络服务**地址: 主机、协议和端口
protocol: **//host** [: **port**]
- 网络服务是由服务器应用程序提供的。一个**网络服务**对应一个**服务器应用程序**, 网络服务的地址实际上也就是提供该服务的服务器应用程序地址
- Web服务地址: **http://61.135.169.121:80**
- FTP服务地址: **ftp://61.135.169.121:21**
- JDBC服务地址: **jdbc://61.135.169.121:3306**



中國農業大學

閻道宏

9.2 网络服务与网络资源

- 统一资源定位符**URL**
 - 网络服务提供的**资源**（resource）
 - 数据文件（例如HTML网页文件）
 - 信息查询程序（例如搜索引擎或动态网页程序）
 - 主机地址：主机名或IP地址
 - 网络服务地址：protocol: //host [: port]
 - **网络资源**地址：统一资源定位符（Uniform Resource Locator，简称**URL**）



9.2 网络服务与网络资源

- 统一资源定位符**URL**
 - 表示**数据文件**的URL：网络服务地址 + 文件路径
 - **http://www.cau.edu.cn:80/index.html**
网络服务 “http://www.cau.edu.cn:80” 下的网页文件
“/index.html”
 - **http://www.cau.edu.cn/images/1182/culture.jpg**
网络服务 “http://www.cau.edu.cn:80” 下的图像文件
“/images/1182/culture.jpg”
 - 统一资源定位符**URL**可以认为是**数据文件的网络文件名**



9.2 网络服务与网络资源

- 统一资源定位符**URL**
 - 表示**信息查询程序**的URL：网络服务地址 + 程序路径 + 查询条件
 - <http://www.cau.edu.cn/search?id=1234&name=Messi>
网络服务“<http://www.cau.edu.cn:80>”下的搜索引擎程序“/search”，搜索条件是“学号id等于1234，并且姓名name等于Messi”
 - <http://www.cau.edu.cn/find.jsp?id=1234&name=Messi>
网络服务“<http://www.cau.edu.cn:80>”下的JSP动态网页程序“/find.jsp”，查询条件是“学号id等于1234，并且姓名name等于Messi”
 - 统一资源定位符**URL**可认为是信息查询程序在网络上的**程序文件名**



9.2 网络服务与网络资源

- 统一资源定位符URL
 - 统一资源定位符URL的语法
protocol: **//host** [**: port**] **path** [**? query**]
 - 表示本地文件的URL
盘符:\目录名\子目录名\.....\文件名.扩展名
 - 加前缀: **file:///**
 - 例如: D:\image\1.png
file:///D:\image\1.png、**file:///D:/image/1.png**



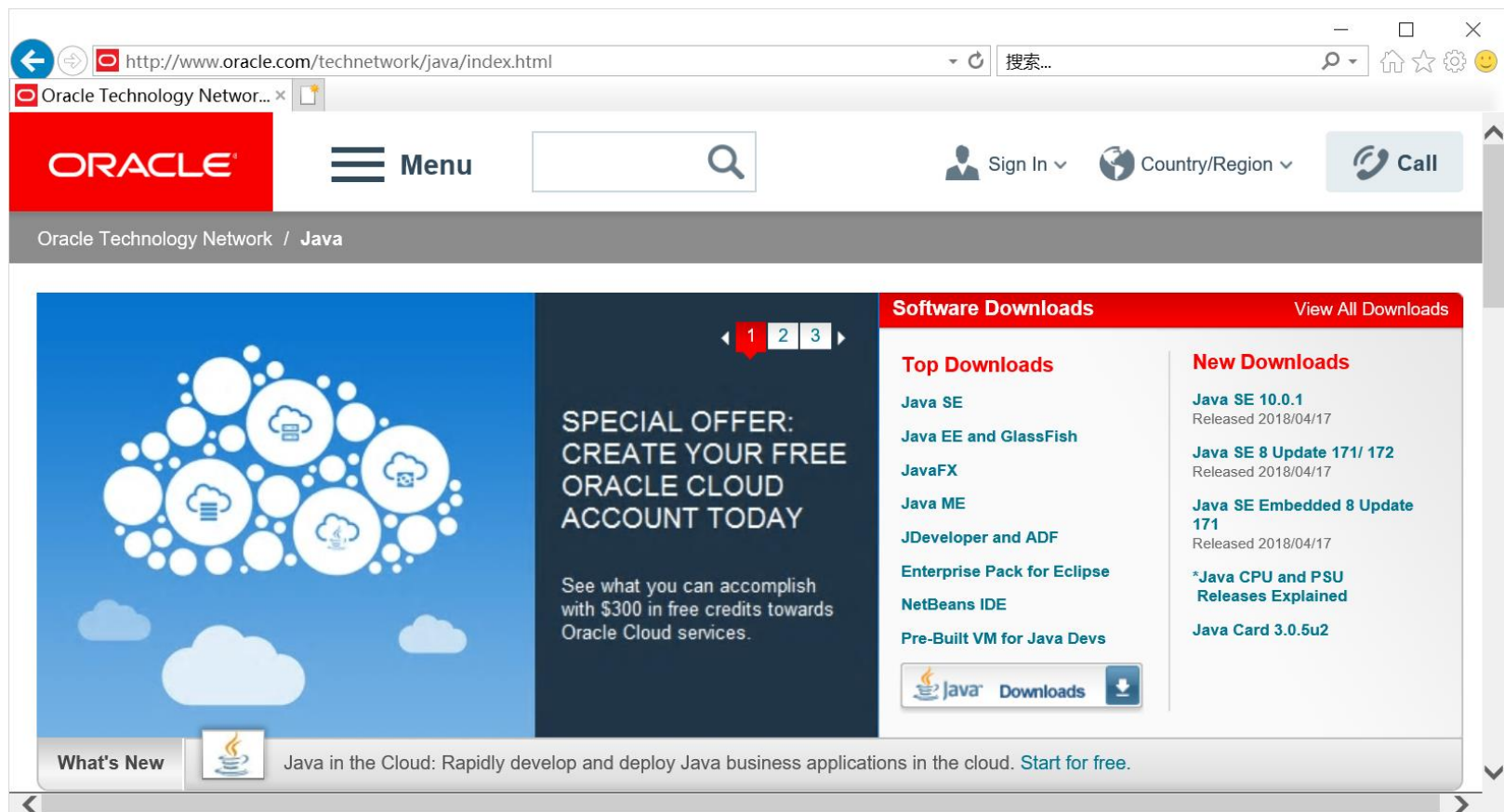
9.2 网络服务与网络资源

- 访问网络资源
 - 以**Web服务**为例讲解如何**编写**客户端应用程序来访问网络资源
 - Web服务主要用于信息查询，其所提供的网络资源就是一组**静态网页**文件，或是由信息查询程序自动生成的**动态网页**
 - Web服务使用的应用层协议是**HTTP**，默认端口为TCP **80**



9.2 网络服务与网络资源

- 访问网络资源



中國農業大學

閻道宏

9.2 网络服务与网络资源

- 访问网络资源

- 程序员可以不用浏览器，而是自己编写一个客户端应用程序来访问Web服务里的网络文件

- 编程过程

- 使用Java API中的统一资源定位符类URL来存放网络文件的文件名
 - 创建URL的输入流对象
 - 像读取本地硬盘文件一样来读取网络文件里的数据



9.2 网络服务与网络资源

- 访问网络资源
 - 统一资源定位符类**URL**

java.net. URL 类说明文档			
public final class URL extends Object implements Serializable			
	修饰符	类成员（节选）	功能说明
1		URL (String spec)	构造方法，创建 URL 对象
2		URL (String protocol, String host, String file)	构造方法，给定协议、主机名和文件名创建 URL 对象
3		URL (String protocol, String host, int port, String file)	构造方法，给定协议、主机名、端口号和文件名创建 URL 对象
4		String getProtocol ()	获取所使用的服务协议
5		String getHost ()	获取主机名
6		int getPort ()	获取端口号
7		int getDefaultPort ()	获取服务协议的默认端口
8		String getPath ()	获取资源的存储路径
9		String getFile ()	获取文件名
10		String getQuery ()	获取查询参数
11		InputStream openStream ()	创建 URL 的输入流对象
12		URLConnection openConnection ()	创建 URL 的连接对象



9.2 网络服务与网络资源

- 访问网络资源
 - 统一资源定位符类**URL**

例9-2 一个统一资源定位符类URL的Java演示程序（JURLTest.java）

```
1 import java.net.*; // 导入java.net网络包中的类
2 import java.io.*; // 导入java.io输入输出流包中的类
3 public class JURLTest { // 测试类:
4     public static void main(String [] args) { // 主方法
5         try { // 处理可能出现的勾选异常MalformedURLException
6             URL url = new URL("http://www.oracle.com/technetwork/java/index.html");
7             System.out.println("URL: " + url);
8             System.out.println("协议: " + url.getProtocol());
9             System.out.println("主机: " + url.getHost());
10            System.out.println("端口: " + url.getPort());
11            System.out.println("默认端口: " + url.getDefaultPort());
12            System.out.println("路径: " + url.getPath());
13        }
14        catch(MalformedURLException e) { e.printStackTrace(); } // 捕捉并处理勾选异常
15    } }
```

Problems Javadoc Declaration Console

<terminated> JURLTest [Java Application] C:\Java\jre1.8.0_152\bin\java
URL: http://www.oracle.com/technetwork/java/index.html
协议: http
主机: www.oracle.com
端口: -1
默认端口: 80
路径: /technetwork/java/index.html



中国农业大学

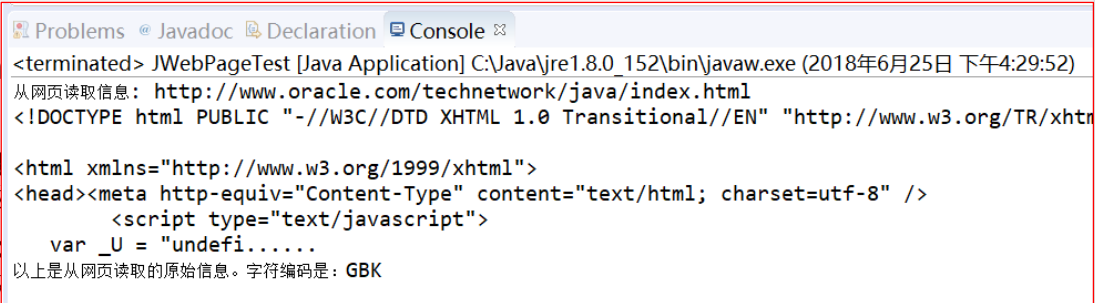
阚道宏

9.2 网络服务与网络资源

- 访问网络资源
 - 访问并读取Web服务里的网页文件

例9-3 一个访问Java语言网站主页的演示程序（JWebPageTest.java）

```
1 import java.net.*; // 导入java
2 import java.io.*; // 导入java
3 public class JWebPageTest {
4     public static void main(String[] args) {
5         try { // 处理可能出现的异常
6             URL url = new URL("http://www.oracle.com/technetwork/java/index.html");
7             System.out.println("从URL: " + url + " 读取网页文件");
8             InputStreamReader in = new InputStreamReader( url.openStream() );
9             char cbuf[] = new char[300]; // 只读300个字符
10            int len = in.read(cbuf);
11            for (int n = 0; n < len; n++)
12                System.out.print( cbuf[n]);
13            System.out.print("\n以上是从网页读取的原始信息。");
14            System.out.println( "字符编码是: " + in.getEncoding() );
15            in.close();
16        }
17        catch(IOException e) { e.printStackTrace(); } // 捕捉并处理勾选异常
18    } }
```



Problems Javadoc Declaration Console

<terminated> JWebPageTest [Java Application] C:\Java\jre1.8.0_152\bin\javaw.exe (2018年6月25日 下午4:29:52)

从网页读取信息: http://www.oracle.com/technetwork/java/index.html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1" [...]

<html xmlns="http://www.w3.org/1999/xhtml">

<head><meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<script type="text/javascript">

var _U = "undefi....."

以上是从网页读取的原始信息。字符编码是: GBK

9.2 网络服务与网络资源

例9-4 一个访问Web服务里图像文件的Java演示程序（JWebImageTest.java）

```
1 import java.net.*; // 导入java.net网络包中的类
2 import java.io.*; // 导入java.io输入输出流包中
3 import java.awt.*; // 以下为导入图形用户界面
4 import java.awt.image.BufferedImage;
5 import javax.imageio.ImageIO;
6 import javax.swing.*;
7
8 public class JWebImageTest {           // 测试
9     public static void main(String [] args) { // 主方
10         try { // 处理可能出现的勾选异常IOException
11             String netURL = "http://www.cau.edu.cn";
12             System.out.println("从网站读取图片：" +
13                 URL url = new URL(netURL);
14             BufferedImage img = ImageIO.read( url ); // 加载网络图像文件
15             // 创建框架窗口，显示加载的网络图像
16             JFrame w = new JFrame("显示网络图片"); // 创建窗口
17             w.setSize(660,360); w.setVisible(true);
18             w.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19             SwingUtilities.invokeLater( ()->{           // 在事件分发线程中绘图
20                 Graphics g = w.getGraphics();           // 获取绘图对象
21                 g.setFont( new Font("Times New Rome",0,24) ); // 设置字体
22                 g.drawString(netURL, 10, 75);           // 显示网址
23                 g.drawImage(img, 10, 100, null);         // 显示图像
24             });
25         }
26         catch(IOException e) { e.printStackTrace(); } // 捕捉并处理勾选异常
27     } }
```



9.3 程序之间的网络通信

- 网络上两台计算机之间的通信，本质上是两个**网络应用程序**之间的通信
- 网络应用程序与普通应用程序之间的最大区别在于，网络应用程序具有**通信功能**，它能与网络上的其他程序互相通信，协同工作
- TCP/IP网络传输层为网络应用程序提供了两种不同的通信方式，分别是有连接通信（**TCP**协议）和无连接通信（**UDP**协议）
- 我们将关注点**聚焦**到程序间的网络通信上，重点学习如何利用**Java API**中与网络通信相关的**类**来编写网络应用程序



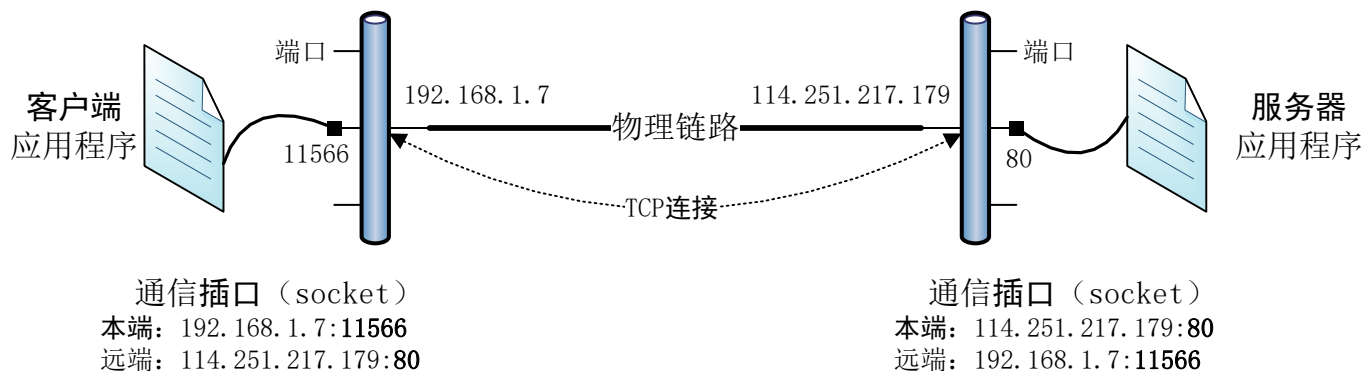
9.3 程序之间的网络通信

- TCP与Socket
 - Client/Server网络服务程序架构，简称**C/S**架构
 - 客户端程序与服务器程序的**通信流程**
 - 服务器应用程序首先确定对外服务的TCP端口，然后持续监听该端口的通信。
 - 客户端应用程序向服务器应用程序的服务端口发送连接请求，申请在双方之间建立一个**TCP连接**
 - 服务器应用程序接收连接请求，确认与客户端应用程序建立TCP连接
 - 客户端应用程序与服务器应用程序基于所建立的TCP连接开始**双向通信**，后续通信内容就是双方所要进行的网络服务的内容
 - 服务结束后，通信双方**断开**（关闭）TCP连接，通信结束
 - 客户端和服务端应用程序之间是基于**TCP协议**来进行网络服务的“**请求-响应**”（request-response）通信的
 - 为了方便程序员编写这样的程序，Java API提供了两个基于TCP协议进行网络通信的类，它们分别是套接字类**Socket**和服务端套接字类**ServerSocket**



9.3 程序之间的网络通信

- TCP与Socket
 - 套接字**Socket**
 - 通信插口



- 套接字Socket
 - **本端** (local) 和 **远端** (remote) 的网络地址和端口信息
 - 对应本端和远端的**网络应用程序**



9.3 程序之间的网络通信

- TCP与Socket
 - Java API中的套接字类**Socket**

| | | | |
|-------------------------------|-----|---|----------------|
| java.net. Socket 类说明文档 | | | |
| public class Socket | | | |
| extends Object | | | |
| implements Closeable | | | |
| | 修饰符 | 类成员（节选） | 功能说明 |
| 1 | | Socket () | 构造方法 |
| 2 | | Socket (InetAddress address, int port) | 构造方法（同时建立连接） |
| 3 | | Socket (String host, int port) | 构造方法（同时建立连接） |
| 4 | | void connect (SocketAddress endpoint) | 客户端向服务器申请连接 |
| 5 | | void connect (SocketAddress endpoint, int timeout) | 客户端向服务器申请连接 |
| 6 | | InputStream getInputStream () | 获取TCP连接的字节输入流 |
| 7 | | OutputStream getOutputStream () | 获取TCP连接的字节输出流 |
| 8 | | InetAddress getInetAddress () | 获取远端的网络地址 |
| 9 | | int getPort () | 获取远端的端口号 |
| 10 | | InetAddress getLocalAddress () | 获取本端的网络地址 |
| 11 | | int getLocalPort () | 获取本端的端口号 |
| 12 | | boolean isConnected () | 检查是否已建立TCP网络连接 |
| 13 | | boolean isClosed () | 检查TCP网路连接是否已断开 |
| 14 | | void close () | 断开TCP网络连接 |



9.3 程序之间的网络通信

• TCP与Socket

例9-5 一个使用套接字类Socket连接Web服务器的Java演示程序（JSocketTest.java）

```
1 import java.net.*; // 导入java.net网络包中的类
2 import java.io.*; // 导入java.io输入输出流包中的类
3
4 public class JSocketTest {           // 测试类：测试套接
5     public static void main(String[] args) { // 主方法
6         try { // 处理可能出现的勾选异常
7             // 创建套接字对象，向服务器申请建立TCP连接
8             Socket s = new Socket("www.cau.edu.cn", 80); //
9             System.out.println(s+"connected.....");
10            // 显示套接字中的本端信息
11            System.out.println("local: "+s.getLocalAddress()+"."+s.getLocalPort());
12            // 显示套接字中的远端信息
13            System.out.println("remote: "+s.getInetAddress()+"."+s.getPort());
14            // 建立TCP连接后可以与服务器进行通信，本例暂不通信
15            s.close(); // 断开TCP网络连接
16            System.out.println("TCP connection closed.....");
17        }
18        catch(UnknownHostException e) { System.out.println("Host not found"); }
19        catch(ConnectException e) { System.out.println("Host connection failed"); }
20        catch(IOException e) { System.out.println("IOException"); }
21    } }
```

Problems @ Javadoc Declaration Console

```
<terminated> JSocketTest [Java Application] C:\Java\jre
Server connected.....
local: /192.168.1.7:11566
remote: www.cau.edu.cn/114.251.217.179:80
TCP connection closed.....
```



中國農業大學

閻道宏

9.3 程序之间的网络通信

- C/S架构程序的代码框架
 - 客户端应用程序的代码结构
 - 算法流程：向服务器**申请**建立TCP连接；连接成功后与服务器进行通信，**发送**服务请求，然后**接收**服务响应；服务结束后**断**开TCP连接
 - 代码结构

```
try { // 处理可能出现的勾选异常
    // 创建套接字对象，向服务器申请建立TCP连接
    Socket s = new Socket(服务器域名或IP地址, 服务端口);
    ..... // 连接成功后与服务器通信，发送服务请求，接收服务响应
    s.close(); // 服务结束后断开TCP连接
}
catch(IOException e) { System.out.println("IOException"); }
```



9.3 程序之间的网络通信

- C/S架构程序的代码框架
 - **服务器**应用程序的代码结构
 - **算法流程**：首先确定对外服务的**TCP端口**，然后持续**监听**（listen）该端口的通信；**接收**（accept）客户端发来的连接请求，确认建立TCP连接；连接成功后与客户端进行通信，**接收**服务请求，然后**发送**服务响应；服务结束后**断开**TCP连接
- 服务器套接字类**ServerSocket**



9.3 程序之间的网络通信

- C/S架构程序的代码框架
 - 服务器应用程序的代码结构
 - 服务器套接字类**ServerSocket**

| java.net.ServerSocket类说明文档 | | | |
|--|-----|--|--------------------------------------|
| public class ServerSocket
extends Object
implements Closeable | | | |
| | 修饰符 | 类成员（节选） | 功能说明 |
| 1 | | ServerSocket() | 构造方法 |
| 2 | | ServerSocket(int port) | 构造方法（同时指定监听端口） |
| 3 | | ServerSocket(int port, int backlog, InetAddress bindAddr) | 构造方法（同时指定监听端口等） |
| 4 | | Socket accept() | 监听服务请求。如果有请求，则建立TCP连接，返回套接字，否则保持阻塞状态 |
| 5 | | int getLocalPort() | 获取所监听的端口 |
| 6 | | InetAddress getInetAddress() | 获取本服务的网络地址 |
| 7 | | void bind(SocketAddress endpoint) | 将服务绑定到指定的套接字地址 |
| 8 | | SocketAddress getLocalSocketAddress() | 获取本服务的套接字地址 |
| 9 | | boolean isBound() | 检查是否已绑定地址和端口 |
| 10 | | boolean isClosed() | 检查服务是否已关闭 |
| 11 | | void close() | 关闭网络服务 |
| | | | |



9.3 程序之间的网络通信

- C/S架构程序的代码框架
 - 服务器应用程序的代码结构

```
try { // 处理可能出现的勾选异常
    // 创建服务器套接字对象，用于监听某个TCP端口
    ServerSocket ss = new ServerSocket(服务端口);

    while (运行条件或true) { // 服务器应保持运行状态，随时接收客户端的连接请求
        Socket s = ss.accept(); // 如果有TCP连接请求，则确认建立连接，返回套接字对象
        ..... // 连接成功后与客户端进行通信，接收服务请求，然后发送服务响应
        s.close(); // 服务结束后断开TCP连接
        // 继续循环，准备接收下一个连接请求。可接收不同客户端的连接请求
    }

    ss.close(); // 关闭网络服务
}
catch(IOException e) { System.out.println("IOException"); }
```



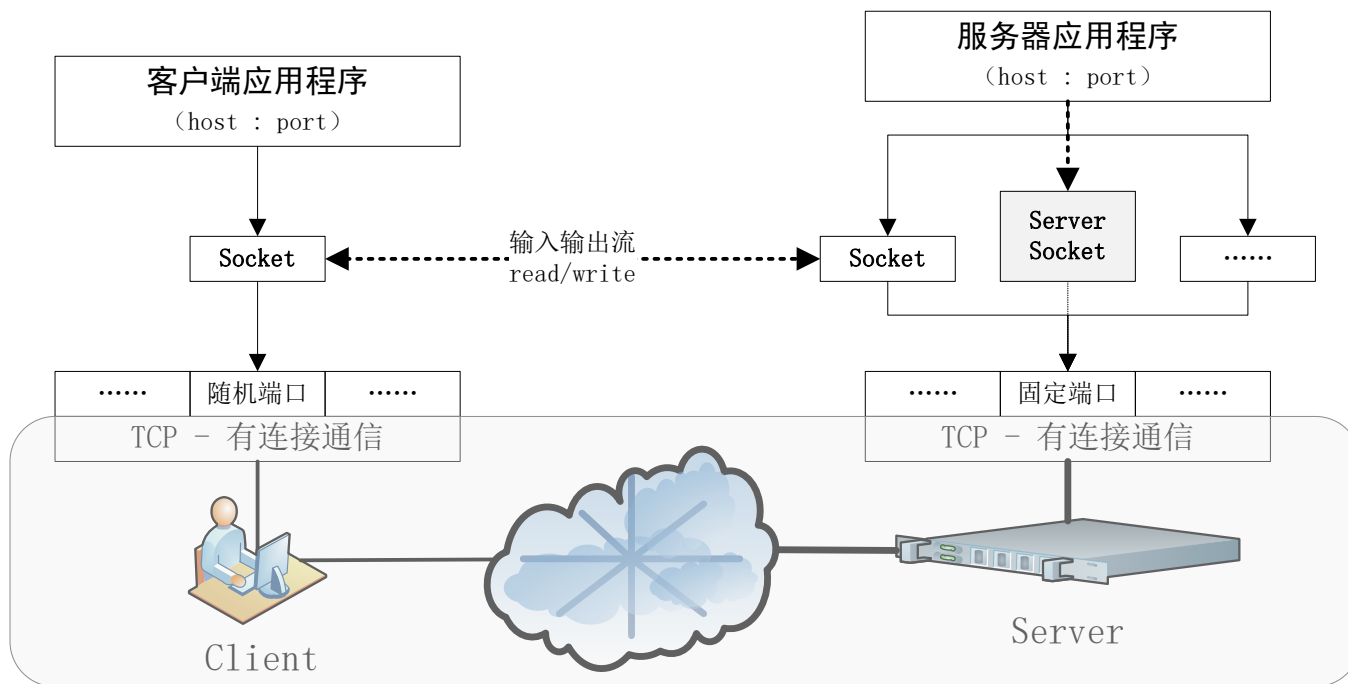
9.3 程序之间的网络通信

- C/S架构程序的代码框架
 - 套接字对象的输入输出流
 - 在TCP连接成功后，客户端与服务器应用程序各有一个套接字**Socket对象**，它们分别表示TCP连接两端的**通信插口**
 - Java API将程序从**通信插口**（即Socket对象）**接收**数据抽象成**输入流**，向通信插口**发送**数据抽象成**输出流**，这样**网络通信问题**就被转换成了**输入输出问题**
 - 套接字类Socket中的两个重要方法
 - **getInputStream()**。获得套接字的字节型输入流对象。从输入流对象读取数据，就是接收对方发来的信息
 - **getOutputStream()**。获得套接字的字节型输出流对象。向输出流对象写入数据，就是向对方发送信息
 - 可以根据需要将字节型输入输出流对象**包装**成其他流对象，例如包装成字符型流对象、带缓冲区的流对象等



9.3 程序之间的网络通信

- C/S架构程序的代码框架
 - 套接字对象的输入输出流
 - 网络编程与计算机网络之间的关系示意图



9.3 程序之间的网络通信

- 一个C/S架构演示程序：**时间服务**程序
 - 时间服务规范
 - 客户端向服务器发送服务请求，请求时需提交客户名称。假设时间服务**约定**：所提交的**客户名称**必须为7个字符长度
Client1、Client2
 - 服务器接收服务请求，然后响应请求，提供时间服务。假设时间服务**约定**：返回给客户端的**响应信息**中应当包含对客户问候，然后再给出标准时间。
返回给客户端Client1的响应信息应当是“**Hello, Client1!** 标准时间”
 - 客户端接收服务器返回的响应信息，服务结束
 - 为便于调试，程序员可以在自己的计算机上同时运行客户端和服务应用程序
 - TCP/IP网络规定本地主机的主机名为“**localhost**”，IP地址为“**127.0.0.1**”



例 9-6 使用时间服务的 Java 客户端应用程序示例代码 (JTimeClient.java)

```
1      20      // 接收服务响应：获取套接字的字节型输入流，然后包装成字符型
2      21      in = new InputStreamReader( s.getInputStream() );
3      22      char[] buf = new char[100]; // 用于存储接收到的响应（最多 100 个字符）
4      23      in.read(buf, 0, buf.length); // 读取响应，例如：Hello, Client1! 标准时间
5      24      String response = new String(buf); // 将字符数组包装成字符串
6      25      System.out.print("接收的服务响应是：" + response);
7      26      System.out.println(" 其编码是：" + in.getEncoding());
8      27      } catch(IOException e) { System.out.println("IOException"); }
9      28      finally { // 服务结束后关闭输入流、输出流以及 TCP 连接
10     29          try { // 处理可能出现的勾选异常
11         30              if (in != null)    in.close(); // 关闭接收信息的输入流
12         31              if (out != null)   out.close(); // 关闭发送信息的输出流
13         32              if (s != null)     s.close(); // 断开 TCP 网络连接
14         33          } catch(IOException e) { System.out.println("IOException"); }
15         34      }
16         35  } }

17      out.write(request, 0, request.length()); // 向服务器发送服务请求
18      out.flush(); // 立即发送
19      System.out.println("发送的服务请求是：" + request);
```



例 9-7 提供时间服务的 Java 服务器

```
1 import java.net.*; // 导
2 import java.io.*; // 导
3 import java.time.*; // 导
4
5 public class JTimeServerST {
6     public static void main(
7         // 服务器算法
8         try { // 处理可
9             // 创建服
10             ServerSoc
11             System.out
12             while (true
13                 Sock
14                 Syste
15                 Syste
16                 time
17                 // 维
18             }
19         } catch (IOExcept
20     }
21 }
```

```
22 public static void timeService(Socket s) { // 请求-响应: 接收客户名, 然后返回时间信息
23     InputStreamReader in = null; // 用于接收信息的输入流
24     OutputStreamWriter out = null; // 用于发送信息的输出流
25     try { // 处理可能出现的勾选异常
26         // 接收服务请求: 获取套接字的字节型输入流, 然后包装成字符型
27         in = new InputStreamReader( s.getInputStream() );
28         char[] buf = new char[7]; // 用于存储客户名 (约定是 7 个字符长度)
29         in.read(buf, 0, buf.length); // 读取客户名, 例如 "Client1"
30         String request = new String(buf); // 将字符数组包装成字符串
31         System.out.print("客户端的服务请求是: " + request);
32         System.out.println(" 其编码是: " + in.getEncoding());
33         // 发送服务响应: 获取套接字的字节型输出流, 然后包装成字符型
34         out = new OutputStreamWriter( s.getOutputStream() );
35         LocalDateTime t = LocalDateTime.now(); // 假设本机时间为标准时间
36         // 生成响应信息, 例如 "Hello, Client1! 标准时间", 然后发送给客户端
37         String response = String.format("Hello, %s! %s", request, t );
38         out.write(response, 0, response.length()); // 向客户端发送响应信息
39         out.flush(); // 立即发送
40         System.out.println("返回客户端的响应是: " + response);
41     } catch (IOException e) { System.out.println("IOException"); }
42     finally { // 服务结束后关闭输入流、输出流以及 TCP 连接
43         try { // 处理可能出现的勾选异常
44             if (in != null) in.close(); // 关闭接收信息的输入流
45             if (out != null) out.close(); // 关闭发送信息的输出流
46             if (s != null) s.close(); // 断开 TCP 网络连接
47         } catch (IOException e) { System.out.println("IOException"); }
48     }
49     // 此处已执行完一轮时间服务的请求-响应算法
50 }
```

9.3 程序之间的网络通信

- 一个C/S架构演示程序
 - C/S架构**时间服务**程序

```
in = new InputStreamReader(s.getInputStream(), "UTF-8");  
out = new OutputStreamWriter(s.getOutputStream(), "UTF-8");
```



The screenshot shows a Java IDE console window with the following content:

```
Problems @ Javadoc Declaration Console x  
JTimeServerST [Java Application] C:\Java\jre1.8.0_152\bin\javaw.exe (2018  
TimeServer started at 8000 .....  
  
收到一个服务请求，并建立了TCP连接： /127.0.0.1:14863  
客户端的服务请求是： Client1 其编码是： GBK  
返回客户端的响应是： Hello, Client1! 2018-06-28T14:57:18.567
```



中國農業大學

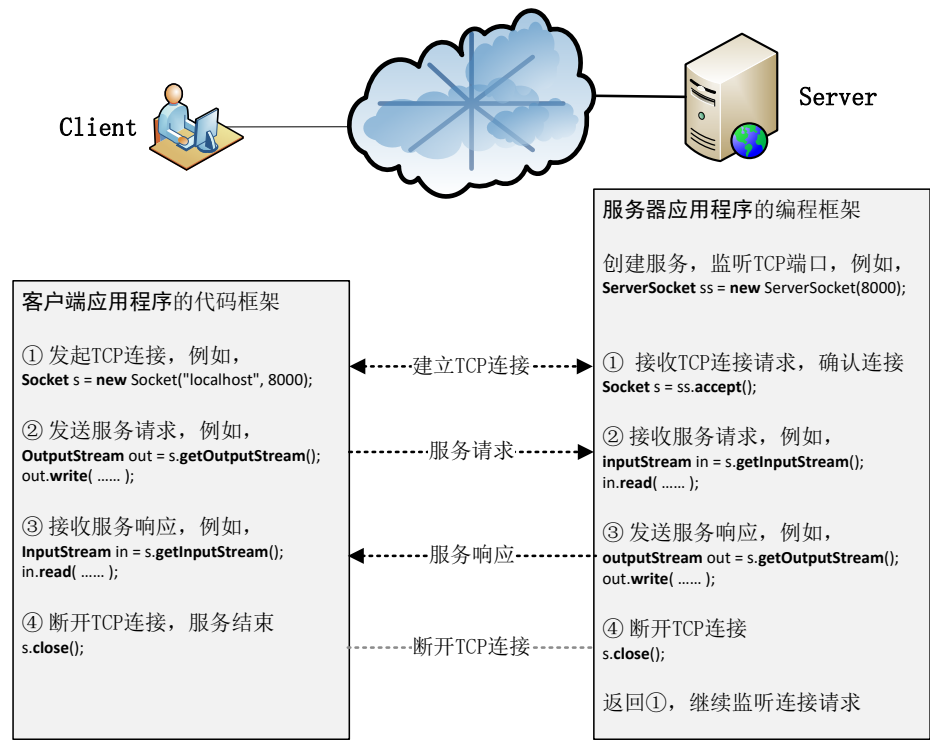
閻道宏

例 9-8 提供时间服务的 Java 多线程服务器应用程序示例代码 (JTimeServerMT.java)

```
1  import java.net.*; // 导入 java.net 网络包中的类
2  import java.io.*;  // 导入 java.io 输入输出流包中的类
3  import java.time.*; // 导入 java.time 包中与时间相关的类
4  public class JTimeServerMT { // 主类：提供时间服务的多线程服务器应用程序
5      public
6          24
7          25 class TimeService implements Runnable { // 时间服务算法类：接收客户名，返回时间信息
8              26 private Socket s = null; // 与用户连接的 Socket
9              27 TimeService(Socket userSocket) { s = userSocket; } // 构造方法
10             28 public void run() { // 实现抽象方法 run(), 编写服务请求-响应的算法代码
11                 29 ..... // 与例 9-7 中代码第 23~48 行相同，此处省略
12                 30 // 此处已执行完时间服务的请求-响应算法，线程将随即结束
13                 31 } }
14             System.out.println( "收到一个服务请求，并建立了 TCP 连接。");
15             System.out.println( s.getInetAddress() + ":" + s.getPort() );
16             //timeService(s); // 执行服务请求-响应算法
17             // 另外创建子线程并在子线程中运行时间服务的请求-响应算法
18             TimeService a = new TimeService(s); // 创建 TimeService 算法对象
19             Thread t = new Thread(a); // 在线程中运行算法对象
20             t.start(); // 启动线程
21             // 继续循环，监听下一个服务请求。可接收不同客户端的连接请求
22         }
23     } catch(IOException e) { System.out.println("IOException"); }
24 }
```


9.3 程序之间的网络通信

- 基于TCP协议的C/S架构网络服务程序
 - 提供服务的服务器应用程序
 - 使用服务的客户端应用程序
 - 两个程序基于**TCP**协议进行网络服务的“**请求-响应**”通信



9.4 基于UDP协议的网络通信

- **TCP**（传输控制协议）是一种有连接的双向通信协议，而**UDP**（用户数据报协议）是一种无连接的单向通信协议
- 基于UDP协议通信程序的**代码框架**
 - UDP协议提供了一种无连接的单向通信方式。如果两个网络应用程序之间使用UDP协议传输数据，则其中一个是**发送方**（sender），另一个是**接收方**（receiver）
 - 发送方与接收方之间的**通信流程**
 - 接收方应用程序首先准备好接收数据的**缓冲区**，然后监听某个**UDP端口**，等待接收该端口传输过来的数据
 - 发送方应用程序首先准备好接收方的**网址**、**UDP端口**和需要发送的**数据**，然后向接收方**发送**数据，通信结束。发送方**不需要**等待接收方的回复
 - 接收方应用程序**接收**数据，然后分析并处理数据，通信结束。接收方**不需要**回复发送方
- 发送方和接收方应用程序之间是基于**UDP**协议来进行数据的“**发送-接收**”（send-receive）通信的
- 为了方便程序员编写这样的程序，Java API提供了两个基于UDP协议进行网络通信的类，它们分别是数据报包裹类**DatagramPacket**和数据报套接字类**DatagramSocket**



9.4 基于UDP协议的网络通信

- 基于UDP协议通信程序的代码框架
 - 数据报包裹类 **DatagramPacket**

| java.net.DatagramPacket类说明文档 | | | |
|---|-----|--|-----------------------|
| public final class DatagramPacket
extends Object | | | |
| | 修饰符 | 类成员（节选） | 功能说明 |
| 1 | | DatagramPacket (byte[] buf, int length) | 构造方法 |
| 2 | | DatagramPacket (byte[] buf, int length,
InetAddress address, int port) | 构造方法，指定接收方的网络地址、UDP端口 |
| 3 | | InetAddress getAddress () | 获取对方的网络地址 |
| 4 | | int getPort () | 获取对方的UDP端口号 |
| 5 | | int getLength () | 获取数据报的数据长度 |
| 6 | | byte[] getData () | 读取数据报包裹中的数据 |
| 7 | | void setAddress (InetAddress iaddr) | 设置接收方的网络地址 |
| 8 | | void setPort (int iport) | 设置接收方的端口号 |
| 9 | | void setLength (int length) | 设置即将被发送的数据长度 |
| 10 | | void setData (byte[] buf) | 设置将即将被发送的数据 |
| | | | |



9.4 基于UDP协议的网络通信

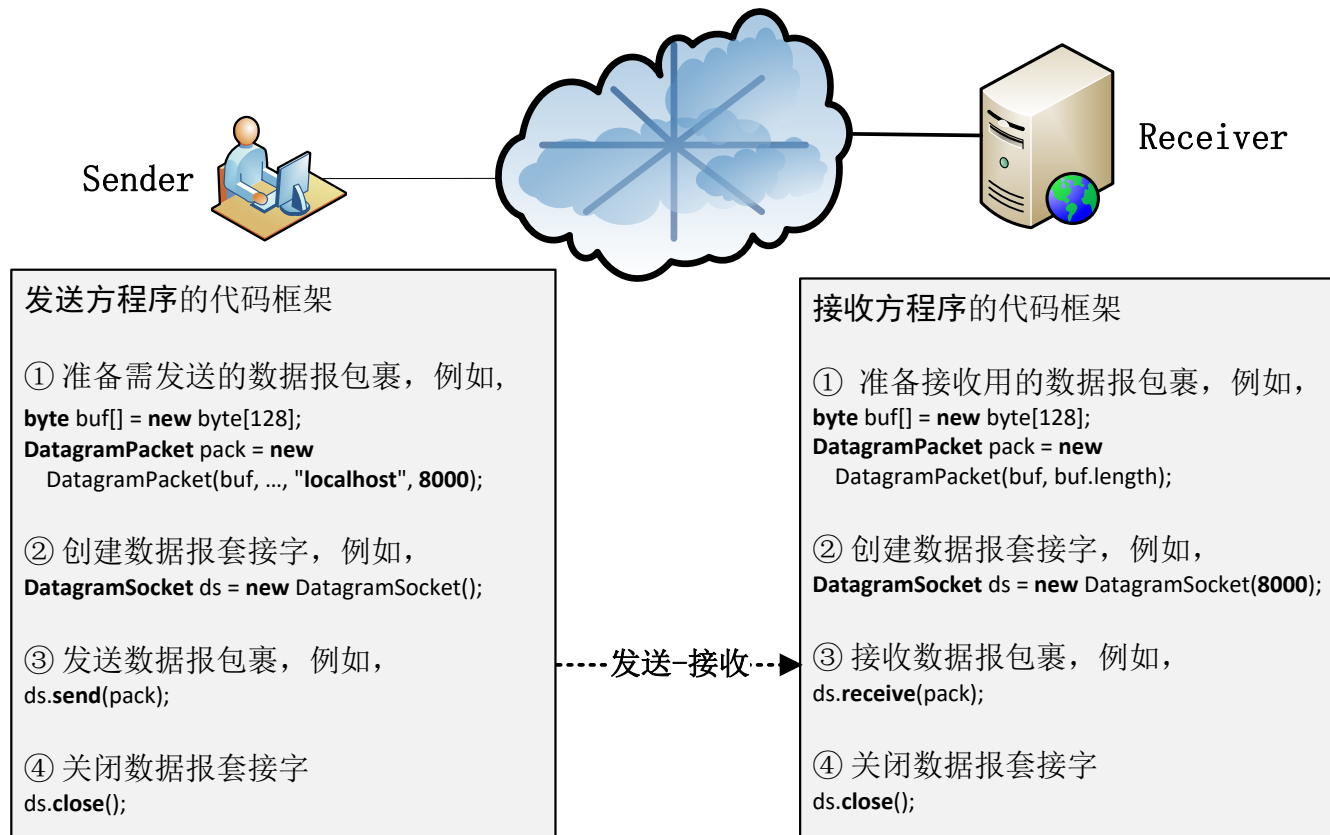
- 基于UDP协议通信程序的代码框架
 - 数据报套接字类 **DatagramSocket**

| java.net.DatagramSocket类说明文档 | | | |
|--|-----|--|-------------------|
| public class DatagramSocket
extends Object
implements Closeable | | | |
| | 修饰符 | 类成员（节选） | 功能说明 |
| 1 | | DatagramSocket() | 构造方法 |
| 2 | | DatagramSocket(int port) | 构造方法，指定本端的UDP端口 |
| 3 | | DatagramSocket(int port, InetAddress laddr) | 构造方法，指定本端的端口和网络地址 |
| 4 | | void send (DatagramPacket p) | 发送数据报包裹 |
| 5 | | void receive (DatagramPacket p) | 接收数据报包裹 |
| 6 | | void setSoTimeout (int timeout) | 设置数据报包裹的有效时间 |
| 7 | | InetAddress getLocalAddress () | 获取本端网络地址 |
| 8 | | int getLocalPort () | 获取本端端口号 |
| 9 | | void close () | 关闭套接字 |
| | | | |



9.4 基于UDP协议的网络通信

- 基于UDP协议通信程序的代码框架



中國農業大學

閻道宏

9.4 基于UDP协议的网络通信

• 基于UDP协议通信程序的代码框架

例9-9 基于UDP协议的接收方应用程序示例代码（JUDPReceiver.java）

```
1 import java.net.*; // 导入java.net网络包中的类
2 import java.io.*; // 导入java.io输入输出流包中的类
3 public class JUDPReceiver {           // 主类：UDP协议接收方的演示程序
4     public static void main(String[] args) { // 主方法
5         try { // 处理可能出现的勾选异常
6             System.out.println("Receive data at 8000.....\n");
7             // 接收前的准备工作：准备好存储缓冲区，将其包装成数据报包裹对象
8             byte buf[] = new byte[128]; // 创建一个数据缓冲区（最多接收128个字节）
9             DatagramPacket pack = new DatagramPacket(buf, buf.length);
10            // 创建数据报套接字对象，监听某个UDP端口，等待接收数据报包裹
11            DatagramSocket ds = new DatagramSocket(8000); // 监听UDP 8000端口
12            ds.receive(pack); // 接收数据报包裹
13            // 分析接收到的数据报包裹，例如发送方的网址、端口和数据等
14            InetAddress udpSender = pack.getAddress(); // 获取发送方的网络地址
15            int port = pack.getPort(); // 获取发送方的端口
16            // 数据报里的数据是字节数组，可将其转成字符串
17            String msg = new String(pack.getData(), 0, pack.getLength()); // 转字符串
18            System.out.println("Receive data from " +udpSender +":" +port);
19            System.out.println("所接收到的数据： " +msg);
20            ds.close(); // 关闭数据报套接字
21        } catch(IOException e) { System.out.println( e.getMessage()); }
22    } }
```



9.4 基于UDP协议的网络通信

- 基于UDP协议通信程序的代码框架

例9-10 基于UDP协议的发送方应用程序示例代码（JUDPSender.java）

```
1 import java.net.*; // 导入java.net网络包中的类
2 import java.io.*; // 导入java.io输入输出流包中的类
3 public class JUDPSender { // 主类：UDP协议
4     public static void main(String[] args) { // 主方法
5         try { // 处理可能出现的勾选异常
6             System.out.print("Send data to localhost:8000
7             // 发送前的准备工作：准备好接收方网址、
8             InetAddress udpReceiver = InetAddress.getByName("localhost");
9             int port = 8000; // 接收方端口（本例为8000）
10            String msg = "Hello, World!"; // 将被发送的信息（本例为"Hello, World!"）
11            byte buf[] = msg.getBytes(); // 将字符串信息转成字节数组
12            // 创建数据报包裹对象，其中包含需被发送的数据和接收方的网址和端口
13            DatagramPacket pack = new DatagramPacket(buf, buf.length, udpReceiver, port);
14            // 创建数据报套接字对象，然后发送准备好的数据报包裹对象pack
15            DatagramSocket ds = new DatagramSocket(); // 创建一个数据报套接字对象
16            ds.send(pack); // 发送数据报包裹
17            ds.close(); // 关闭数据报套接字
18            System.out.println("Done");
19        } catch (IOException e) { System.out.println( e.getMessage() ); }
20    } }
```

Problems @ Javadoc Declaration Console

<terminated> JUDPReceiver [Java Application] C:\Java

Receive data at 8000.....

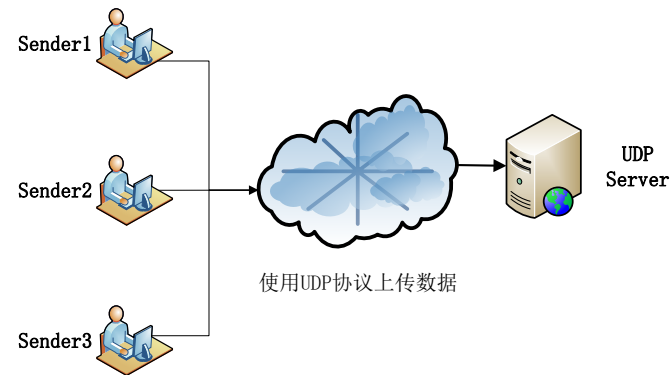
Receive data from /127.0.0.1:56778

所接收到的数据: Hello, World!



9.4 基于UDP协议的网络通信

- UDP接收服务器



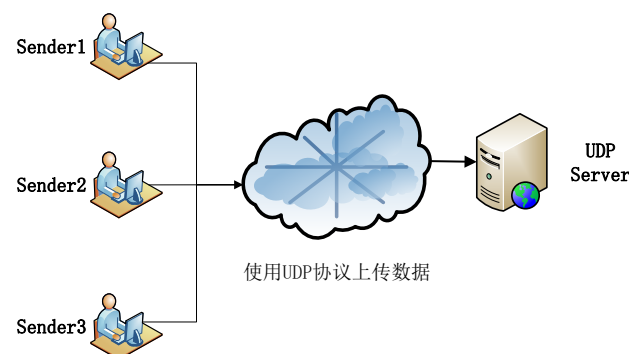
- UDP接收服务器的代码结构

```
try { // 处理可能出现的勾选异常
    // 接收数据前的准备工作：准备好存储数据缓冲区，然后包装成一个数据报包裹对象
    byte buf[] = new byte[缓冲区大小]; // 创建一个数据缓冲区
    DatagramPacket pack = new DatagramPacket(buf, buf.length); // 将缓冲区包装成数据报包裹对象
    // 创建数据报套接字对象，然后通过该套接字对象监听某个UDP端口，接收数据报包裹
    DatagramSocket ds = new DatagramSocket(UDP端口号); // 指定被监听的UDP端口
    while (运行条件或true) { // UDP接收服务器应一直处于运行状态，随时接收并处理数据
        ds.receive(pack); // 接收发送来的数据报包裹
        ..... // 处理所接收到的数据报包裹，可通过网络地址区分出不同发送方（采集点）
    }
    ds.close(); // 关闭数据报套接字
}
catch(IOException e) { System.out.println( e.getMessage() ); }
```



9.4 基于UDP协议的网络通信

- UDP接收服务器



- UDP接收服务器的特点

- **UDP接收服务器**比TCP接收服务器运行**效率高**。使用TCP协议的接收服务器与每个发送方都建立一个TCP连接。当有很多个发送方时，服务器需要维护大量的TCP连接，这会耗费很多系统资源
- **UDP接收服务器**可能会**丢失**数据。因网络拥堵等原因，UDP接收服务器可能会错过发送方上传的数据，即丢失数据。而TCP协议是一种有连接的通信，它能自动重发被丢失的数据，因此TCP是一种可靠的数据传输方式



9.4 基于UDP协议的网络通信

- UDP多播

- UDP协议中的发送方与接收方

- 一个发送方对一个接收方
 - 多个发送方对一个接收方（即UDP接收服务器）
 - 一个发送方对多个接收方，这被称为是UDP多播（或称为UDP组播）

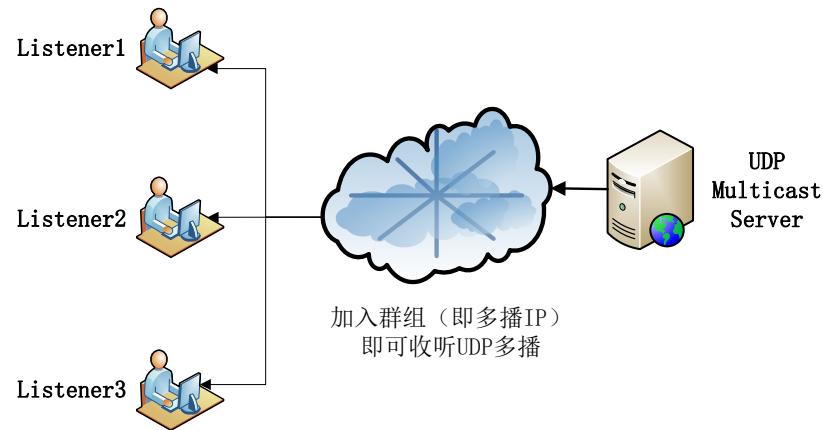
- 多播地址

- 使用UDP多播发送数据，发送方不是将数据发送给某个单一的接收方，而是发向一个群组（group）
 - TCP/IP协议使用IP地址来标识网络上的某台计算机，但是该如何标识网络上的一个群组呢？
 - TCP/IP协议预留了一些特殊的IP地址，专门用于标识网络上的不同群组。这些特殊的IP地址被称为多播地址（multicast address），或称为D类地址。多播地址的范围是224.0.0.0 ~ 239.255.255.255



9.4 基于UDP协议的网络通信

- UDP多播
 - UDP多播的通信流程

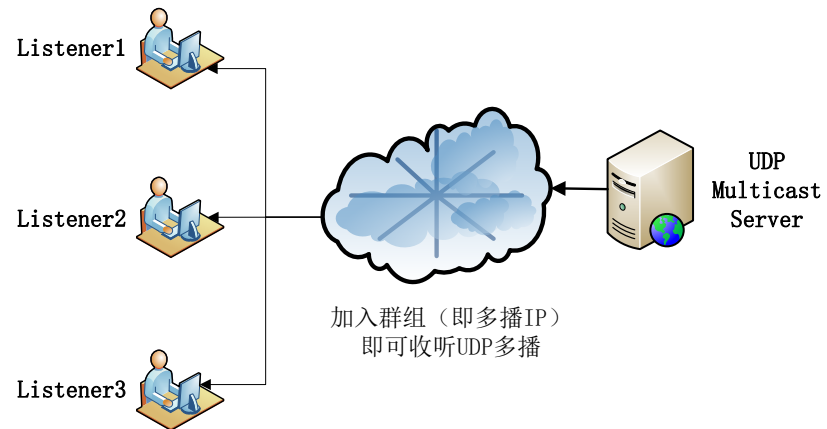


- **发送方**应用程序首先确定群组的多播地址和UDP端口号，然后使用UDP协议向群组持续发送数据。例如向群组持续发送标准时间的时间服务器。UDP多播里的“发送方”被称为“**多播服务器**”（multicast server）
- **接收方**应用程序首先准备好接收数据用的缓冲区，然后加入发送方指定的群组（用多播地址表示），监听该群组指定UDP端口的通信，接收群组中发送的数据。这有点类似于在收听某个频道的广播，例如收听时间频道播出的时间报时。UDP多播里的“接收方”被称为“**收听方**”（listener）



9.4 基于UDP协议的网络通信

- UDP多播



– 编写UDP多播程序

- **发送方**（多播服务器）应用程序需要将原来接收方的普通网络地址改成一个表示群组的多播地址
- **接收方**（收听方）应用程序则需要将原来的数据报套接字类 **DatagramSocket** 换成Java API中另一个UDP多播专用的套接字类，即多播套接字类 **MulticastSocket**



9.4 基于UDP协议的网络通信

- UDP多播
 - 多播套接字类 **MulticastSocket**

| java.net.MulticastSocket类说明文档 | | | |
|--|-----|--|-------------------|
| public class MulticastSocket
extends DatagramSocket | | | |
| | 修饰符 | 类成员（节选） | 功能说明 |
| 1 | | MulticastSocket() | 构造方法 |
| 2 | | MulticastSocket(int port) | 构造方法，指定群组的UDP端口 |
| 3 | | void joinGroup (InetAddress mcastaddr) | 加入多播群组 |
| 4 | | void receive (DatagramPacket p) | 接收数据报包裹 |
| 5 | | void send (DatagramPacket p) | 发送数据报包裹 |
| 6 | | void setTimeToLive (int ttl) | 设置数据报包裹的TTL（生存）时间 |
| 7 | | void leaveGroup (
InetAddress mcastaddr) | 退出多播群组 |
| | | | |



9.4 基于UDP协议的网络通信

- UDP多播
 - 一个UDP多播演示程序
 - 包括多播服务器和收听方两个应用程序
 - 为便于调试，程序员可以在同一台计算机主机上运行这两个程序
 - 多播服务器程序在运行时会持续向群组（多播地址224.0.1.1）的UDP 8000端口发送标准时间
 - 收听方程序启动后会加入这个群组，接收5次标准时间



9.4 基于UDP协议的网络通信

• UDP多播

例9-11 一个UDP多播服务器演示程序的Java示例代码（JMcastServer.java）

```
1  import java.net.*; // 导入java.net网络包中的类
2  import java.io.*; // 导入java.io输入输出流包中的类
3  import java.time.*; // 导入java.time包中与时间相关的类
4  public class JMcastServer { // 主类：UDP多播服务器演示程序
5      public static void main(String[] args) { // 主方法
6          try { // 处理可能出现的勾选异常
7              System.out.println("Send multicast data to 224.0.1.1:8000.....\n");
8              // 首先准备好多播群组和一个数据报套接字对象
9              InetAddress group = InetAddress.getByName("224.0.1.1"); // 生成多播地址
10             DatagramSocket ds = new DatagramSocket(); // 创建一个数据报套接字对象
11             // 开始多播：播出当前标准时间，总共播出100次，每次间隔一秒
12             for (int n = 1; n <= 100; n++) {
13                 LocalDateTime t = LocalDateTime.now(); // 假设本机时间为标准时间
14                 String msg = t.getHour() + ":" + t.getMinute() + ":" + t.getSecond();
15                 byte buf[] = msg.getBytes(); // 将字符串转换成字节数组
16                 // 将标准时间封装成数据报包裹，发送到多播群组的某个UDP端口
17                 DatagramPacket pack = new DatagramPacket(buf, buf.length, group, 8000);
18                 ds.send(pack); // 将数据报发到多播群组
19                 Thread.sleep(1000); // 休眠1秒
20             }
21             ds.close(); // 关闭数据报套接字
22         } catch (Exception e) { System.out.println( e.getMessage() ); }
23     } }
```



9.4 基于UDP协议的网络通信

例9-12 一个UDP多播收听方演示程序的Java示例代码 (JMulticastListener.java)

```
1 import java.net.*; // 导入java.net网络包中的类
2 import java.io.*; // 导入java.io输入输出流包中的类
3 public class JMulticastListener { // 主类: UDP多播收听方演示程序
4     public static void main(String[] args) { // 主方法
5         try { // 处理可能出现的勾选异常
6             System.out.println("Listen multicast data at 224.0.1.1:8000.....\n");
7             // 收听前的准备工作: 准备好数据缓冲区, 然后包装成数据报包裹对象
8             byte buf[] = new byte[128]; // 定义接收数据的缓冲区
9             DatagramPacket pack = new DatagramPacket(buf, buf.length);
10            // 创建多播套接字对象, 然后加入到多播服务器指定的多播群组
11            MulticastSocket ms = new MulticastSocket(8000); // 使用UDP 8000端口
12            InetAddress group = InetAddress.getByName("224.0.1.1"); // 生成多播地址
13            ms.joinGroup(group); // 让多播套接字对象ms加入多播群组
14            for (int n = 1; n <= 5; n++) { // 收听5次多播
15                ms.receive(pack); // 收听多播群组中发送的数据报
16                // 将数据报包裹里的字节数据转成字符串
17                String msg = new String(pack.getData(), 0, pack.getLength());
18                System.out.println(n + "-收到的数据: " + msg);
19                Thread.sleep(1000); // 休眠1秒
20            }
21            ms.leaveGroup(group); // 退出多播群组
22            ms.close(); // 关闭多播套接字
23            System.out.println("Quit the multicast group at 224.0.1.1:8000.");
24        } catch (Exception e) { System.out.println(e.getMessage()); }
25    } }
```

Problems @ Javadoc Declaration Console

<terminated> JMulticastListener [Java Application] C:\Java\

Listen multicast data at 224.0.1.1:8000.....

1- 收到的多播数据: 23:27:34
2- 收到的多播数据: 23:27:35
3- 收到的多播数据: 23:27:36
4- 收到的多播数据: 23:27:37
5- 收到的多播数据: 23:27:38
Quit the multicast group at 224.0.1.1:8000.



第9章 网络编程

- 本章学习要点

- 了解**计算机网络**的基本原理，理解网络编程中常用的网络**概念和术语**
- 学习并掌握基于**TCP**或**UDP**协议网络应用程序的**代码框架**，并能熟练运用**Java API**中相关的类进行**网络编程**
- 掌握在**单台计算机上调试**网络应用程序的方法
- 本章所学习的网络知识已基本能够**满足**网络编程的需要。如果希望深入学习**计算机网络**，读者可以进一步选修专门的**计算机网络课程**

