21世纪高等学校规划教材 | 计算机应用

网络程序设计实验教程(Java语言)

何怀文 彭政 编著

清华大学出版社

21世纪高等学校规划教材 | 计算机应用

网络程序设计实验教程(Java语言)

何怀文 彭政 编著

清华大学出版社
北京

内 容 简 介

本书基于Java语言介绍C/S架构的网络通信应用程序开发技术，以实验和应用案例为主，讲解Java网络通信程序编写的相关知识。全书分为两部分。第一部分为实验内容，共9章，主要内容包括：数据编码、多线程技术、I/O流、主机网络信息获取、DNS解析、TCP Socket编程、UDP Socket编程、组播编程、网络服务器编程模型、报文封装技术、Java底层网络报文捕获和分析技术、电子邮件编程技术等。第二部分为附录，介绍常用TCP/IP网络协议报文结构以及Java窗体开发环境Eclipse的Swing Designer的基本使用。

本书针对网络编程中的难点和重点进行详细的图文讲解，为初学者学习和理解网络编程提供了有力的帮助。本书实例丰富，每章最后都附有一个综合性应用实例讲解，并提供完整可编译实现的源代码，同时每章附有思考题目。本书可以作为独立学院和一般本科计算机相关专业“网络程序设计”课程的配套实验教材，也可以单独使用，同时也可作为Java网络编程爱好者和技术人员的参考用书。

图书在版编目(CIP)数据

网络程序设计实验教程：Java语言/何怀文，彭政编著. --北京：清华大学出版社，2016
21世纪高等学校规划教材·计算机应用
ISBN 978-7-302-42298-3

Ⅰ. ①网… Ⅱ. ①何… ②彭… Ⅲ. ①JAVA语言－程序设计－高等学校－教材 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2015)第287011号

责任编辑：刘向威
封面设计：傅瑞学
责任校对：焦丽丽
责任印制：宋　林

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn，010-62795954
印 装 者：北京鑫海金澳胶印有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：12　　**字　　数**：292千字
版　　次：2016年4月第1版　　**印　　次**：2016年4月第1次印刷
印　　数：1～2000
定　　价：29.00元

产品编号：055813-01

前言

“网络程序设计”课程是计算机学科重要的编程课程之一，网络编程技术也成为目前非常流行的编程技术。由于该课程需要学生具有一定的编程语言基础和掌握基本网络工作原理，同时还涉及多线程、IO 流等操作系统知识，对初学者具有一定的难度。

本书以培养和增强学生的动手编程实战能力为目标，对 Java 网络编程内容进行了精心的挑选和安排，对难点、重点进行了详细的图文阐述，力求做到深入浅出、通俗易懂。同时，本书包含了大量网络开发应用案例，通过有效的编程实验教学，培养学生程序设计思维、提升学生实战能力。

全书分为两部分：第 1 部分为实验内容，共 9 章；第 2 部分为附录。本书各章内容介绍如下。

第 1 章介绍数据编码和解码的基本概念、常见的字符编码以及 Java 语言处理字符编码的方法，目的是让读者对数据有一个初步的认识。

第 2 章介绍 Java 多线程编程技术。包括 Java 线程创建、线程同步、线程间通信、线程池、定时器等概念。多线程技术是网络编程的基础，也是开发高效率的网络通信程序必备技术之一，要求读者必须熟悉。

第 3 章重点介绍网络编程中用到的各种 Java IO 流。IO 流是网络程序通信中读取数据的基本技术，也是后续章节用到的预备知识。本章还介绍了对象序列化技术，最后给出一个文件分割器的综合案例。

第 4 章介绍 Java 网络地址类 InetAddress 以及 NetworkInterface 类。本章知识涉及到网络地址表示、DNS 解释、获取主机网络信息等内容。最后介绍一个多线程网络主机扫描综合案例，方便读者了解网络技术的应用。

第 5 章介绍 TCP 套接字编程技术，是本书的难点和重点。本章内容包括：文本数据传输、二进制数据传输、报文封装格式等。重点在于 TCP 报文粘包、TCP 报文边界划分技术等。本章还介绍了文本传输、二进制传输以及序列化传输等不同传输技术，最后介绍文件服务器程序和网络聊天室的综合案例。

第 6 章介绍 UDP 编程技术和 UDP 报文封装技术。本章包括 TCP、UDP 编程区别和应用选择。最后介绍远程主机唤醒和 UDP 文件传输的综合案例。

第 7 章介绍组播编程技术并给出一个基于组播的聊天室案例。

第 8 章介绍 JavaMail 编程技术，同时给出了基于 JavaMail 电子邮件接收和发送示例程序。

第 9 章介绍基于 JPcap 网络报文捕获和分析应用程序，用于进行底层网络报文分析。

附录部分包括常见的 TCP/IP 网络协议的报文格式以及 Eclipse 的 Swing Designer 的窗体设计和控件使用。

本书的所有实验代码均在 Eclipse 3.7 和 JDK 1.7 上通过编译和正常运行。本书由何

怀文组织和设计,第2～7章由何怀文编写;第1、8～9章以及附录部分由彭政编写。何怀文负责全书审核。本书编写过程中参考了Java编程和网络编程的著作文献,同时还查阅了大量的网络资料,在此对所有的作者表示感谢。

由于编者水平有限,书中不妥和错误之处还望读者批评指正,作者联系地址为E-mail:he_huai_wen@aliyun.com。

编　者

2015年10月

目 录

第1章 数据的编码和解码

编码与解码是计算机表示数据的重要技术之一。在网络通信中，很多情况下通信双方传达的都是字符信息。但是，字符信息并不能直接从网络的一端传递到另一端，这些字符信息首先需要被转换成二进制 bit 序列后才能在网络中传输。将字符序列转换为二进制 bit 序列的过程称为编码。当这些字节传送到网络的接收方时，接收方需要反过来将二进制 bit 序列再转换为字符序列，这个过程称为解码。本章主要从字符的编码和解码出发，介绍数据编码和解码的概念、常见的字符集以及 Java 语言处理字符编码、解码的方法。

1.1 实验目的

(1) 理解数据编码与解码的概念。
(2) 了解常见的字符编码。
(3) 掌握 Java 语言处理字符编码、解码的方法。

1.2 实验原理

1.2.1 数据编码与解码

编码与解码是计算机世界的主要技术之一。计算机只能存储和处理二进制数据，所以编码是指用二进制 bit 序列来表示声音、图片、字符等信息；解码是指将表示声音、图片、字符等信息的二进制 bit 序列还原为原来的信息。

在处理编码与解码的关系时，可能会出现以下两种错误的情况：①由于不知道某种编码算法而无法对使用这种编码的信息进行解码，比如使用 Windows Media Player 默认的情况下，无法打开 ape 的音频文件。②使用编码 a 对使用编码 b 的信息进行了错误解码，比如字符编码解码不匹配的情况下就会出现字符乱码。在网络应用程序开发中特别要注意字符编码解码不匹配的情况。

1.2.2 常见的字符编码

本次实验中，重点练习字符的编码与解码。首先要对字符的常见编码有一定的了解。

ASCII 编码：计算机是英语国家发明的，英语只有 26 个英文字母，加上特殊符号也不超过 100 种字符，采用一个字节存储一个字符的方式，8 个 bit 序列有 256 种组合，即一个字

节最多能表示256种字符。最初的ASCII编码只占用了一个字节的后面7位,最前面的1位统一规定为0。之后,IBM在此基础上做了扩展,用8bit表示1个字符,共256个字符,称为ISO-8859-1字符集。

GBK编码:非英语国家使用的语言较为复杂,尤其是欧亚语系,需要采用多个字节去存储一个字符。GBK编码兼容ASCII编码,所有ASCII字符的ASCII编码和GBK编码是一致的。通常,GBK编码中使用一个字节存储一个英文字符,使用两个字节存储一个汉字字符。GBK编码也兼容简体中文GB2312编码,即GB2312编码也是GBK编码的子集。在中文Windows操作系统环境下,GBK编码是操作系统的默认编码方案(ANSI)。

Unicode编码:可以想象,如果有一种编码,将世界上所有的符号都纳入其中。每一个符号都给予一个独一无二的字节序列,那么乱码问题就会消失,这就是Unicode编码方案。Unicode的学名是Universal Multiple-Octet Coded Character Set,简称为UCS,UCS可以看作是Unicode Character Set的缩写,就像它的名字所表示的,这是一种所有符号的编码。Unicode当然是一个很大的集合,现在的规模可以容纳100多万个符号,每个符号的编码都不一样。需要注意的是,Unicode只是一个符号集,它只规定了符号的二进制代码,却没有规定这个二进制代码应该如何存储。例如"汉"字的UCS编码是6C49,可以用2个字节去存储(UTF-16),也可以用3个字节去存储(UTF-8)。

UTF-8、UTF-16编码:Unicode提供了编码方案,没有提供存储方案。于是产生了很多的编码存储方案,如UTF-8,UTF-16等。例如在Windows操作系统的记事本中,如果存储为Unicode编码,实际上就是采用UTF-16的编码方式去存储。在UTF-16这种编码方案中,每个英文字符和常用汉字字符都需要两个字节来存储。UTF-8是Unicode中使用最广泛的实现方式,在很多Linux系统中和开源软件产品中都是采用这种编码方案。UTF-8编码最大的特点是与ISO-8859-1完全兼容,也就是说在UTF-8编码方案中,每个英文字符只需要占用1个字节的存储空间,但是UTF-8编码中汉字字符通常需要占用3个字节的存储空间。

1.2.3 字符串 java.lang.String 的编码、解码方法

String类中定义了涉及到字符编码、解码处理的相关方法,如表1-1所示。

表1-1 String类编码、解码的主要方法

方法声明	说明
public String(byte[] bytes)	通过使用平台的默认字符集解码指定的byte数组,构造一个新的String
public String(byte[] bytes, Charset charset)	通过使用指定的charset解码指定的byte数组,构造一个新的String
public String(byte[] bytes, String charsetName)	通过使用指定的charsetName解码指定的byte数组,构造一个新的String
public byte[] getBytes()	使用平台的默认字符集将此String编码为byte序列,将结果存储到一个新的byte数组中,并返回这个byte数组
public byte[] getBytes(Charset charset)	使用给定的charset将此String编码到byte序列,将结果存储到新的byte数组,并返回这个byte数组
public byte[] getBytes(String charsetName)	使用指定的字符集将此String编码为byte序列,将结果存储到一个新的byte数组中,并返回这个byte数组

相关示例代码如下所示：

采用系统默认字符集将一个字节数组 array 解码成字符串 str：

```
String str = new String(array);
```

采用 UTF-8 字符集将一个字节数组 array 解码成字符串 str：

```
String str = new String(array,"UTF-8");
```

采用系统默认字符集将一个字符串 str 编码为一个字节数组 array：

```
byte[] array = str.getBytes();
```

采用 UTF-8 字符集将一个字符串 str 编码为一个字节数组 array：

```
byte[] array = str.getBytes("UTF-8");
```

1.2.4　字符集 java.nio.Charset

Charset 类定义了用于创建解码器和编码器以及获取与 charset 关联的各种名称的方法，此类的实例是不可变的。对于 Charset 类的使用，要掌握以下几点：

(1) 得到一个 Charset 类的实例。

要获得一个 Charset 类的实例有两种方法：

① public static Charset defaultCharset()：通过这个类的静态方法 defaultCharset 可以得到一个 JVM 平台默认的 Charset 对象，这个 Charset 对象是根据语言环境和底层操作系统的 charset 来确定的，在中文 Windows 环境下，得到的是 GBK 字符集对象：

```
Charsetcs = Charset.defaultCharset();
```

② public static Charset forName(String charsetName)：通过这类静态方法 forName 可以得到由字符串参数 charsetName 指定的 Charset 字符集对象。比如创建一个 UTF-8 字符集对象：

```
Charset cs = Charset.forName("UTF-8");
```

(2) 由 java.nio.Charset 的实例得到对应的编码器 java.nio.CharsetEncoder 的实例和解码器 java.nio.CharsetDecoder 的实例。

需要使用 Charset 类的两个实例方法：

① public abstract CharsetEncoder newEncoder()：为 Charset 对象构造对应的编码器。比如创建 Charset 对象 cs 的编码器对象：

```
CharsetEncoder encoder = cs.newEncoder();
```

② public abstract CharsetDecoder newDecoder()：为 Charset 对象构造对应的解码器。比如创建 Charset 对象 cs 的解码器对象：

```
CharsetDecoder decoder = cs.newDecoder();
```

(3) 由编码器对象将一个字符序列编码为一个字节序列，由解码器对象将一个字节序列解码为一个字符序列。

需要用到 CharsetEncoder 类的编码方法和 CharsetDecoder 类的解码方法：

① public final ByteBuffer encode(CharBuffer in)：CharsetEncoder 类的实例方法，能把单个输入字符缓冲区 CharBuffer in 的剩余内容编码到新分配的字节缓冲区 ByteBuffer，并返回这个字节缓冲区对象。比如，由编码器对象 encoder 将一个 CharBuffer 字符缓冲对象 in 编码为一个 ByteBuffer 字节缓冲对象 out：

```
ByteBuffer out = encoder.encode(in);
```

② public final CharBuffer decode(ByteBuffer in)：CharsetDecoder 类的实例方法，能把单个输入字节缓冲区 ByteBuffer in 的剩余内容解码到新分配的字符缓冲区 CharBuffer，并返回这个字符缓冲区对象。比如，由解码器对象 decoder 将一个 ByteBuffer 字节缓冲对象 in 解码为一个 CharBuffer 字符缓冲对象 out：

```
CharBuffer out = decoder.decode(in);
```

(4) 可以通过字符集 Charset 类的实例方法直接进行编码和解码操作。

① public final ByteBuffer encode(CharBuffer cb)：Charset 类的实例方法，将此 charset 中的 Unicode 字符编码成字节的便捷方法。

② public final CharBuffer decode(ByteBuffer bb)：Charset 类的实例方法，将此 charset 中的字节解码成 Unicode 字符的便捷方法。

③ public final ByteBuffer encode(String str)：Charset 类的实例方法，将此 charset 中的字符串编码成字节的便捷方法。比如，由字符集对象 cs 将一个字符串对象 str 编码为一个 ByteBuffer 字节缓冲对象 out：

```
ByteBuffer out = cs.encode(str);
```

1.3 实验内容

应用背景：编写一个 Swing 窗体应用程序，实现的功能是：①将字符序列文本框中的字符串以对应的编码方案进行编码，并存储到字节数组中，最后将字节数组中的字节以十六进制的方式表示；②将字节序列文本框中十六进制方式表示的字节数组以对应的编码方案进行解码，并将解码后得到的字符串显示出来。实现效果如图 1-1 所示。

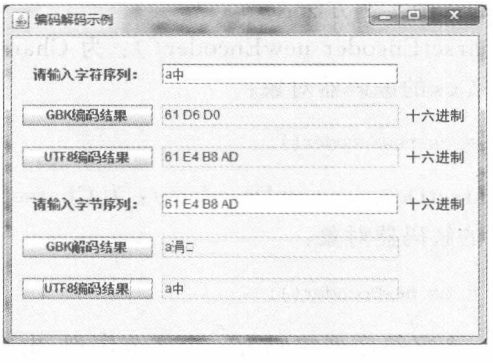

图 1-1 编码解码程序运行效果图

1.3.1　程序界面设计实现

新建一个名为 TestEncoding 的 Java 项目，在项目中使用 Swing Designer 创建一个名为 AppMain 的 Application Window 应用程序，将主窗体重命名为 frmMain，并将窗体标题改为“编码解码示例”。

拖动工具箱中的控件按图来设计窗体，主要控件的属性如表 1-2 所示。

表 1-2　AppMain 控件属性表

控　件	名称(Name)	属　性	值
按钮 JButton	btnGBKEncode	Text	GBK 编码结果
按钮 JButton	btnUTF8Encode	Text	UTF-8 编码结果
按钮 JButton	btnGBKDecode	Text	GBK 解码结果
按钮 JButton	btnUTF8Decode	Text	UTF-8 解码结果
文本框 JTextField	tfChars		
文本框 JTextField	tfBytesGBK	Enabled	False
文本框 JTextField	tfBytesUTF8	Enabled	False
文本框 JTextField	tfBytes		
文本框 JTextField	tfCharsGBK	Enabled	False
文本框 JTextField	tfCharsUTF8	Enabled	False

1.3.2　编码功能的实现

在按钮 btnGBKEncode 的单击事件中添加如下代码：

```
String str = tfChars.getText();
Charset cs = Charset.forName("GBK");
ByteBuffer buffer = cs.encode(str);
String hexStr = "";
while (buffer.remaining() > 0) {
    hexStr += Integer.toHexString(buffer.get() & 0xFF).toUpperCase() + " ";
}
tfBytesGBK.setText(hexStr);
```

在按钮 btnUTF8Encode 的单击事件中添加如下代码：

```
String str = tfChars.getText();
String hexStr = "";
try {
    byte[] bytes = str.getBytes("UTF-8");
    for (int i = 0; i < bytes.length; i++) {
        hexStr += Integer.toHexString(bytes[i] & 0xFF).toUpperCase() + " ";
    }
} catch (UnsupportedEncodingException e) {
    e.printStackTrace();
} finally {
    tfBytesUTF8.setText(hexStr);
}
```

1.3.3 解码功能的实现

在按钮 btnGBKDecode 的单击事件中添加如下代码：

```
String hexStr = tfBytes.getText();
String[] strs = hexStr.split(" ");
byte[] array = new byte[strs.length];
for (int i = 0; i < strs.length; i++) {
    array[i] = (byte)(Integer.valueOf(strs[i], 16).intValue());
}
Charset cs = Charset.forName("GBK");
CharBuffer buffer = cs.decode(ByteBuffer.wrap(array));
tfCharsGBK.setText(buffer.toString());
```

在按钮 btnUTF8Decode 的单击事件中添加如下代码：

```
String hexStr = tfBytes.getText();
String[] strs = hexStr.split(" ");
byte[] bytes = new byte[strs.length];
for (int i = 0; i < strs.length; i++) {
    bytes[i] = (byte)(Integer.valueOf(strs[i], 16).intValue());
}
String str = "";
try {
    str = new String(bytes, "UTF-8");
} catch (UnsupportedEncodingException e1) {
    e1.printStackTrace();
} finally {
    tfCharsUTF8.setText(str);
}
```

1.3.4 英文字符和中文字符编码结果比较

从图 1-2 中可以看到，对于英文字符来说，不管是 GBK 编码还是 UTF-8 编码，都是用一个字节存储表示，而且编码值都相同，且与 ASCII 编码兼容。

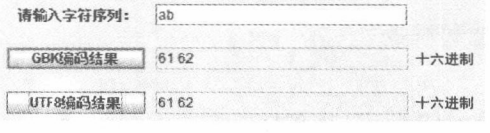

图 1-2 英文字母编码

从图 1-3 中可以看到，对于中文字符来说，GBK 编码使用了两个字节来存储表示一个中文字符，而 UTF-8 编码采用了三个字节来存储表示一个中文字符。所以在信息主要是中文字符的情况下，采用 GBK 编码可以有效地减少存储空间和提高信息的网络传输效率。

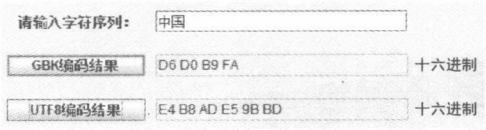

图 1-3　汉字编码

1.3.5　编码解码是否一致的效果比较

从图 1-4 中可以看到，如果编码和解码采用一致的编码方案，则不会出现乱码现象；如果采用不一致的编码方案，则可能会出现乱码现象。如用 GBK 进行编码得到的字节数组再采用 UTF-8 去解码，对有些字符来说就会出现解码失败，也就出现了乱码。

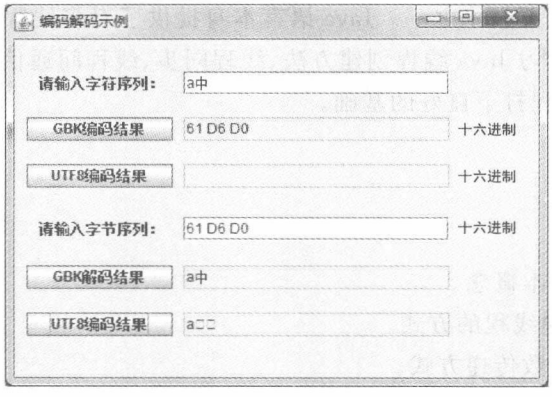

图 1-4　编码解码字符集不一致

1.4　小结与思考

在计算机世界中，字符、图片、声音等各种信息都需要以二进制的方式去存储和处理，这叫做信息的编码。各种信息都有多种不同的编码方案，在处理信息的时候要考虑采用哪种编码方案，要考虑统一编码方案、要考虑进行编码的转换等多方面的操作。本次实验重点介绍了常用的字符编码，以及在 Java 语言中处理字符编码所涉及的步骤和方法。

思考题：

1. 为什么有时看到的网页会显示乱码？如何解决这个问题。

2. 记事本程序可以选择某种编码方案存储为 txt 文件，但是记事本程序打开一个 txt 文件时，如何判断这个 txt 文件中的字符采用的编码方案？

3. 了解代码页、ANSI、BOM、Big-Endian 和 Little-Endian 等概念。

第2章 Java多线程编程

线程是操作系统分配CPU计算资源的基本单位，灵活运用线程编程技术可以提升程序性能和提供良好的用户交互，如进度条、多线程下载等。同时多线程编程技术也是构建高性能网络服务常用的重要技术之一。Java语言本身提供了对多线程的支持，不用依赖特定操作系统。本章通过学习Java线程创建方法、线程同步、线程间通信、线程池以及定时器等技术，为后面的网络编程打下良好的基础。

2.1 实验目的

(1) 掌握线程的基本概念。
(2) 掌握Java创建线程的方式。
(3) 掌握进程间参数传递方式。
(4) 掌握线程同步、线程间通信的方法。
(5) 掌握线程池的工作原理和使用。
(6) 掌握Java定时器的应用。

2.2 实验原理

2.2.1 线程概念与线程状态

线程(Thread)是CPU资源调度的最小单元，一个CPU在任何时刻只允许有一个正在运行的线程(多核CPU允许有多个线程同时运行)，系统需要决定哪个线程可以拥有CPU资源。而在目前多核CPU中，多线程技术可以更高效地利用CPU每个核的计算资源，提高程序运行速度。

和线程相似的是进程(Process)。但是进程指的是一个正在运行的应用程序，进程范围比线程更大。通常一个进程往往拥有多个线程，并伴随着进程的结束，线程也随之消亡。

由于计算机中CPU核数有限，远小于操作系统需要运行的线程数量，因此，当一个线程被创建后，不一定会马上执行，通常会进入到等待队列，等待CPU调度。大部分操作系统(如Linux、Windows等)采用分时调度策略，为了避免某些线程长期独占CPU资源，系统将时间平均分为若干个时间片，每个线程占用固定时间片，不管任务是否执行完成，都会让