《电脑编程技巧与维护》杂志十周年庆典暨真情回馈读者活动
《电脑编程技巧与维护》杂志社策划

编程技巧典型案例集锦系列

Java 编程技巧 典型案例解析

Java Programming

Java Programming Java Programming

《电脑编程技巧与维护》杂志社 编著

- 多线程、序列化等重要概念深入分析
- 开发学生信息管理系统
- 收发电子邮件
- 编写 Web 服务器
- P2P 网络通信
- 制作图像切换特效
- 网络文件下载及上传系统
- 邮件群发系统
- 跨平台的代理服务器及其计费

赠送1CD，含书中实例源代码，编程高手经验汇集，即学即用

中国电力出版社
www.infopower.com.cn

《电脑编程技巧与维护》杂志十周年庆典暨真情回馈读者活动
《电脑编程技巧与维护》杂志社策划

编程技巧典型案例集锦系列

1162425

《电脑编程技巧与维护》杂志社　编著

Java 编程技巧

典型案例解析

中国电力出版社
www.infopower.com.cn

内 容 简 介

Java是一种网络编程语言，随着互联网的爆炸式发展，它也成为了当今最主流的程序设计语言之一，因其跨平台、多线程、面向对象等多种优点，受到了广大编程人员的青睐。

本书精选了《电脑编程技巧与维护》杂志近年来的精彩文章，内容涉及Java基础技巧与应用、数据库应用、网络编程应用和图形图像处理等热门领域，通过大量实用案例讲解了Java的重要概念（如多线程、序列化、构造方法等）、典型应用（如连接数据库、开发学生信息管理系统、收发电子邮件、编写Web服务器、实现P2P网络通信、制作图像切换特效等）和编程技巧（如动态类载入、UDP多点传送、数据库连接池、网络文件下载及上传系统、邮件群发、带滚动条图像的缩放等），所有实例均来自实际项目并调试通过，并对其中的关键技术进行了详细点评，以方便读者理解。配书光盘中包含了实例源代码。

本书定位于有Java应用基础的编程人员和应用开发人员，对初学Java编程的新手也有一定的参考价值，是进行课程项目开发、毕业项目设计的高等院校学生的必备读物，也是相关高等培训学校的理想案例教程。

图书在版编目（CIP）数据

Java编程技巧典型案例解析／《电脑编程技巧与维护》杂志社编著. —北京：中国电力出版社，2005
（编程技巧典型案例集锦系列）
ISBN 7-5083-3256-3

Ⅰ.J... Ⅱ.电... Ⅲ.JAVA语言－程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字（2005）第016110号

版权声明

本书由中国电力出版社独家出版。未经出版者书面许可，任何单位和个人均不得以任何形式复制或传播本书的部分或全部内容。

本书内容所提及的公司及个人名称、产品名称、优秀作品及其名称，均为所属公司或者个人所有，本书引用仅为宣传之用，绝无侵权之意，特此声明。

策　　划：裴红义
　　姚贵胜
责任编辑：姚贵胜
责任校对：崔燕菊
责任印制：邹树群

丛 书 名：编程技巧典型案例集锦系列
书　　名：Java编程技巧典型案例解析
编　　著：《电脑编程技巧与维护》杂志社
出版发行：中国电力出版社
　　地址：北京市三里河路6号　邮政编码：100044
　　电话：（010）88515918　传真：（010）88518169
印　　刷：北京市铁成印刷厂
开本尺寸：185 × 260　　印　张：20.25
书　　号：ISBN 7-5083-3256-3
版　　次：2005年7月北京第1版
印　　次：2005年7月第1次印刷
印　　数：1-5000
定　　价：35.00元（含1CD）

丛 书 序

在《电脑编程技巧与维护》杂志创刊10周年之际，为了真诚回报多年来一直关爱和支持本刊的广大读者，《电脑编程技巧与维护》杂志社和中国电力出版社共同策划出版了《编程技巧典型案例集锦系列》丛书。《电脑编程技巧与维护》杂志是为从事电脑编程、系统应用开发人员创办的专业性和实用性都很强的技术刊物，它从1994年创刊，十多年来始终遵循着“实用第一，智慧密集”的办刊宗旨，紧跟计算机软硬件技术发展和应用趋势，不断求变创新，针对软件开发过程中许多关键技术问题，着重提供各类解决方案。对电脑编程人员来说，程序开发能力的提高，除了对语言和算法的学习外，还要集思广益，充分借鉴参考别人的长处，深入透彻地理解其中的精髓，然后融入到自己的设计方案中去，这样无论是对于自身还是整体都有莫大的提高，这也正是我们编写这套系列丛书的初衷。

本丛书包括《Visual C++编程技巧典型案例解析——基础与应用篇（上）》、《Visual C++编程技巧典型案例解析——基础与应用篇(下)》、《Visual C++编程技巧典型案例解析——图形图像处理与数据库篇》、《Visual C++编程技巧典型案例解析——网络与通信及计算机安全与维护篇》、《Visual Basic编程技巧典型案例解析》、《Delphi编程技巧典型案例解析》、《C#编程技巧典型案例解析》、《Java编程技巧典型案例解析》、《PowerBuilder管理信息系统编程技巧典型案例解析》9册共545个典型案例。每册书的编程案例，均依不同的编程应用分成若干章，条目清晰可查，使用极为方便。

本丛书选编了《电脑编程技巧与维护》杂志近一两年发表的和一部分尚未发表而又极为实用、精彩的典型编程实例，特点是：其各册内容均来自编程高手的智慧，凝结了500余位编程高手与名家的心血，关键技术专家点评；其案例是从实际项目提炼出的开发范例，超过800个技术要点的经典解决方案。案例讲解部分先给出设计目标，然后介绍实现目标的基本思想和方法，最后详细给出其核心程序的源代码，对程序的关键部分进行讲解并给出程序的运行效果；其编程技巧新颖实用，构思巧妙，汇集了众多顶级程序员和业界知名专家的成功经验，告诉读者最好的创意和最实用的方法。全套书既讲究内容的深入性、专业性和权威性，同时兼顾轻松、通俗易懂、时效性强的特点，带给读者的是一份清

新、纯粹的体验感受。

本丛书是《电脑编程技巧与维护》杂志资源的二次开发，浓缩了当前主流编程语言Visual C++、Visual Basic、Delphi、Java、C#、PowerBuilder等程序设计的精华，其目的是力求为读者建造一个真正的知识整合，是编程思想、编程技术、技巧交流的平台，让读者从中学习到编程高手的诀窍，丰富读者的编程技巧，拓宽读者的编程思路，迅速提升读者的程序开发能力。该丛书可作为高等院校学生进行课程项目开发、毕业项目设计的参考教材，软件从业人员及编程爱好者的珍藏宝典，也可作为高等培训学校的案例教程。

实例导航学编程，自学成才成高手，思想、智慧、理念、经验、技巧无处不在……

《电脑编程技巧与维护》杂志社

2005年1月

此为试读，需要完整PDF请访问：www.ertongbook.com

前　言

Java具有平台独立、面向对象、多线程等许多优点，在网络应用编程中已成为首选的编程语言，受到广大编程人员的喜爱。

在《编程技巧典型案例集锦系列》丛书中，《Java编程技巧典型案例解析》精选了《电脑编程技巧与维护》杂志近两年半共30期已发表的精彩编程实例59例。根据Java的不同应用对象，将精选的59个实例共分为4章：第1章“Java编程基础与应用”是为初学Java编程和应用的读者提供一些入门的实例；第2章“数据库应用编程”为编程人员提供使用Java进行数据库编程方法和技巧；第3章“网络应用编程”介绍Java在网络编程方面的应用实例和技巧；第4章“图形图像处理应用编程”介绍Java实现图形图像处理的编程技巧的一些实例。全书每一章都本着实用第一的原则，紧紧围绕一个主题展开，循序渐近、由浅入深地介绍了使用Java进行应用程序开发的思想方法与编程技巧。书中的每一个实例都给出了开发过程、技术难点及其解决的方法和技巧。这些典型案例所涵盖的编程技巧是经验的总结，具有一定的代表性，很值得参考和借鉴。

本书的主要特色如下：第一，每一章都是通过一个个的实例来介绍Java应用编程方法和技巧，避免了枯燥、空洞的理论，并且每一个实例都具有很强的实用性和代表性。在实例的讲解上一般都是先给出设计目标，然后介绍实现该目标的基本思想和方法，最后详细给出其核心程序的源代码，并对程序的关键部分进行讲解，给出程序的运行效果。第二，所选的每一个实例都是从事Java应用编程人员的经验总结，具有很强的实用性，其中很多编程技巧可供借鉴。第三，每一个实例的程序源代码都经过上机调试通过，给程序开发人员移植源代码带来了方便，加快编程应用的步伐。

本书是《电脑编程技巧与维护》杂志的二次开发，浓缩了Java程序设计的精华，其目的是提升读者Java程序开发能力，把应用Java进行编程的心得体会、经验与读者共享。该书定位于有Java应用基础的编程人员和应用开发人员，对初学Java编程的新手也有一定的参考价值。该书内容全面、概念清晰、层次分明，实例典型而实用，但不足甚至疏漏之处在所难免，恳请广大读者批评指正。

《电脑编程技巧与维护》杂志社

2005年1月

目　录

第 2 章 数据库应用编程

第 3 章 网络应用编程

第 4 章 图形图像处理应用编程

第 1 章
Java 编程基础与应用

■实例 1
Java 中的构造方法

Java 是一个纯粹的面向对象的编程语言，而面向对象的一个重要特征就是将复杂的事物抽象为类，再由实例对象反过来将类具体化。经过这样一个由具体到抽象，再由抽象到具体的转化过程，面向对象编程的特点就不难理解了。在 Java 中，有系统已经定义好的类，也有由程序员根据需要所定义的自己的类，将类具体化为一个实例对象，就必须用到构造方法这个概念。

一、构造方法的概念

依据 SUN 公司的说法，构造方法是用于初始化对象实例的一组指令集，其目的是进行对象的初始化。但是，构造方法又不是方法，因为它没有返回值，也不能被继承。因此，本文在不影响理解的情况下，称构造方法为构造器(Constructor)。

在类中，构造器以三种形式存在。首先就是 Java 提供的默认构造器。若程序中没有显式定义构造器，那么生成类的实例对象时就使用该类的默认构造器。默认的构造器是不带参数的，也没有程序体。一旦程序员显式定义了一个构造器，无论它是否携带参数，默认的构造器就自动丢失了。因此，构造器的第二种形式就是程序员定义的不带参数的构造器，它不可以重复定义，而且即使它也没有程序体，它与默认的构造器仍然是两个不同的概念。构造器的第三种形式是程序员定义的带参数的构造器。其中带参数的构造器会因为参数个数的不同、参数顺序的不同、参数类型的不同而可以定义多个。当一个类有多个构造器时，我们称之为“构造方法的重载”。

二、一个类的不同构造方法之间的调用

如前所述，一个类可以有多个构造器，在这多个构造器之间是可以产生调用关系的。只不过在一个构造器中只能调用一次一个其他的本类构造器，而且必须将这个调用语句放在该构造器的第一句，否则将无法通过编译。

例如：

```
public class A
{
   A()
   {
      System. out. println("This constructor of A has not  parameter. ");
   }
   A(int i)
   {
      this(); //这是非常关键的调用语句
      System. out. println("This constructor of A has parameter,  it is " +i);
```


此为试读，需要完整PDF请访问：www.ertongbook.com

```
    }
    public static void main(String args[])
    {
      int n=8;
      A aInstance=new A(n);
    }
}
```

该程序的运行结果如下：

```
This constructor of A has not  parameter.
This constructor of A has parameter,  it is 8
```

从上例中可以看出，类 A 的带参构造器在第一行使用了 this()语句，表示调用前面定义的类 A 的无参构造器。

三、类与类之间构造方法的调用

假设程序员编写了多个彼此间既无继承关系，也无包含关系的类，那么当各个类之间产生调用关系（不失一般性，假设类 A 需使用类 B 的对象）时，构造器的调用是必然的。只不过这个调用比较简单，它只需在类 A 中根据需要正确地使用类 B 的合适构造器创建一个类 B 的对象即可。

四、子类与其超类之间构造方法的调用

在面向对象的编程语言中，继承是一个重要特征。因而关于彼此间构造方法的调用，也相对复杂。

1. 子类构造方法的特点

在 Java 中，子类创建对象时，会自动调用其父类不带参数的构造器。也就是说，使用子类的任何一种形式的构造器（无论带参数与否）来创建一个实例对象时，都会自动调用其父类不带参数的构造器。如果这个类是层层继承下来的，则这个调用会回溯至级别最高的超类，然后依次调用这棵继承树的各类的无参构造器。

例如：

```
class A
{
   A()
   {
      System.out.println("This is the constructor of class A. ");
   }
   A(int n)
   {
      System.out.println("This is the constructor of class A,  its parameter is " +n);
   }
}
class AA extends A
{
   AA()
   {
      System.out.println("This is the constructor of class AA. ");
   }
   AA(int n)
   {
```

```
        System. out. println("This is the constructor of class AA, its parameter is " +n);
    }
}
public class AAA extends AA
{
    AAA()
    {
        System. out. println("This is the constructor of class AAA. ");
    }
    AAA(int n)
    {
        System. out. println("This is the constructor of class AAA, its parameter is " +n +". ");
    }
    public static void main(String args[])
    {
        int m =20;
        AAA child1 =new AAA();
        AAA child2 =new AAA(m);
    }
}
```

该例的运行结果如下：

```
This is the constructor of class A.
This is the constructor of class AA.
This is the constructor of class AAA.
This is the constructor of class A.
This is the constructor of class AA.
This is the constructor of class AAA, its parameter is 20.
```

在这里应注意一个问题：当使用子类构造器时，一定要保证其各级超类无参构造器的存在与正确性。显然在上例中，如果没有定义类 A 与类 AA 的无参构造器是无法通过编译的。针对这个问题，有三种方案解决：一是保证提供各级超类的无参构造器；二是不提供任何构造器，则各级超类的默认构造器将起作用；三是各级超类没有提供无参构造器，但提供了有参构造器，则在子类构造器中以 super（参数）来调用各级超类的有参构造器，从而避免了自动调用无参构造器的默认方式。

2. 多态性

子类不但继承了超类的一切，还可以重置超类的方法，拥有自己的数据成员。也就是说，子类是对超类的扩展。因此而成就了面向对象程序设计的多态性，这是面向对象编程的另一个重要特征：创建一个子类的实例对象当作父类。形式是：父类名　对象名 = 子类构造器。以这种形式创建的对象在编译时被当作父类，而在运行时却指向子类，灵活实现了“以不变应万变”的面向对象的特征。

五、内部类与外部类的构造方法之间的调用

1. 非静态内部类

当在一个类的定义未结束而又定义另一个类时，我们称这个新定义的类为内部类，原来的类为外部类。当外部类含有的是非静态内部类时，对于该外部类的非静态方法，可以通过直接调用内部类的构造器而创建内部类的对象，从而达到访问内部类的目的。

但对于外部类的静态方法，要想访问它的内部类，就必须先调用自身的构造器创建一个实例对

象，然后以格式“外部类名．内部类名 内部类对象名 = 外部类对象名．内部类构造器”来创建内部类的实例对象，从而达到访问内部类的目的。

2. 静态内部类

当一个内部类被定义为 static（即静态的）时，则该内部类自动升级为一个正常的类。因此当它与包含它的类发生调用关系时，就像普通的类与类之间的调用一样，需要创建彼此的实例对象。但对于外部类的静态方法而言，仍然要遵从前述的步骤与格式创建内部类的对象，或者使用格式“外部类名．内部类名．内部类方法名”达到使用内部类方法的目的。

3. 当内部类是超类时

当内部类是另一个类的超类时，问题就会变得比较复杂。因为如前所述，要创建类的对象，会首先调用它的超类的构造方法，但这个超类此时是内部类，是无法自动调用的，必须要通过该内部类的外部类的实例对象来访问。所以应在子类中使用格式“外部类构造器．super()”来直接访问作为超类的内部类的构造方法，或者创建子类以其超类（即内部类）的外部类为参数的构造器。

总的来说，在 Java 中，构造器是一个非常基本而重要的内容，对它的理解与正确使用对于帮助读者进一步搞清 Java 的面向对象的特征有着重要的意义。

（张廷香）

■实例 2
Java UDP 编程及应用

随着 Internet 技术的发展，互联网给我们提供了越来越多的服务，使我们的工作、学习及生活越来越方便，而这些服务都架构在信息传输服务之上。TCP（Transmission Control Protocol，传输控制协议）和 UDP（User Datagram Protocol，用户数据报协议）就是在网络传输层提供的两种最常用的协议。这两种协议都有自己的优缺点，具体采用哪种传输协议要根据所需网络传输服务的特点（如可靠性、带宽、定时等）进行选择，以获取最佳的网络传输效率。Java 语言深受关注的一个重要原因就是它是一种面向网络编程的语言，为网络编程提供方便高效的编程接口，本文主要对 Java UDP 的编程及应用进行论述。

一、UDP TCP 及端口

TCP 是 Internet 的一个重要协议。Internet 的许多常用服务如 HTTP（Hyper – Text Transfer Protocol，超文本传输协议）、FTP（File Transfer Protocol，文件传输协议）、Telnet 等就是通过 TCP 来实现的。它是一种基于连接的通信协议，当两台计算机之间需要进行可靠的数据传输时，它们通过网络建立起一个稳定可靠的连接。与打电话相类似，这种连接是点对点的，通信的双方则通过这条数据连接来回传输数据。在这条稳定的连接基础上，TCP 协议通过信息校验，能够保证接收方所接收到的数据和发送方所发送的数据在内容和顺序上是完全一致的，从而实现了数据的可靠传输。

UDP 与 TCP 协议之间的不同之处在于 UDP 不是一种基于稳定连接的通信协议。UDP 协议将独立的数据包从一台计算机传输到另外一台计算机，但是并不保证接收方能够接收到该数据包，也不保证接收方所接收到的数据和发送方所发送的数据在内容和顺序上完全一致。因此，UDP 协议更类似于普通邮政服务，寄信人不能够保证所寄出去的信能够被收信人及时收到，后发出的信也许会比先发出的信更早到达。

对于很多应用程序来说，在互相通信的两台计算机之间保证一个可靠与稳定的数据链是至关重要的。在这种情况下，就应该首先考虑使用 TCP 协议在两台计算机之间建立起 TCP/IP 连接。在 HTTP、FTP 以及 Telnet 应用程序中，均要求在通信的双方之间建立起稳定可靠的数据链，因此它们都使用了 TCP 协议来进行数据传输。

在 TCP 协议中，发送方和接收方必须交换额外的信息，以保证接收方已经接收到发送方所发送的数据包，并且所接收到的数据和发送方所发送的数据在内容和顺序上是完全一致的。这些额外的信息交换提高了数据传输的可靠性，但是也给网络带来了额外的负担，导致数据交换的延迟，从而降低了整个网络的数据交换能力。对于某些对实时性要求较高的应用程序来说，这样的延迟有可能是不可接受的。例如一个毫秒级的时钟服务器按照一定的频率向客户机提供当时的时间数据，如果这些时间数据在传输过程中受到了较大的延迟，那么这些过时的时间数据是完全没有意义的，即使

客户机准确无误地接收到了这些数据。相反，如果客户机所接收到的每一个数据包都是实时的，那么即使客户机错过了一两个数据包也是可以接受的，因为它总是可以根据后面所接收到的数据包来对自己进行校正。因此，对于对实时性要求比较高但是对传输可靠度要求比较低的应用程序来说，UDP 协议显然是一个合适的选择。

在传统的以太网(Ethernet)构架下，计算机与计算机之间的数据交换都是通过交换机来完成的。如果一份数据需要被传送给多个接收者，在使用 TCP/IP 连接的情况下，数据发送者需要向交换机发送 N 个同样的拷贝，而交换机则负责将这 N 个拷贝分发给所有的接收者；在使用 UDP 数据广播的情况下，数据发送者只需要向交换机发送一个拷贝，交换机负责将这个信息制作 N 个拷贝发送给所有的机器。在这种情况下，使用 TCP/IP 连接会大大地增加网络的负担。

通常来说，一台计算机只有一个物理接口与网络相连接，所有的应用程序均通过该物理接口从网络接收数据或者将数据发送到网络。由于一个网络上同时存在多台计算机，并且一台计算机上有可能同时存在多个应用程序需要与网络进行数据交换，所以我们通常使用 IP 地址和端口号来识别需要进行数据交换的计算机和应用程序。每台计算机由一个 32 位的 IP 地址来识别，在一个网络中，每台计算机的 IP 地址都是惟一的，因此应用程序能够根据 IP 地址来将数据发送到正确的计算机。每个需要与网络进行数据交换的应用程序均被系统分配一个 16 位的端口号，系统根据这个端口号将从网络接收到的数据转发给相对应的应用程序。端口号的范围是 0～65535，其中 0～1023 被系统所保留，主要是用来提供 HTTP、FTP 以及 TELNET 等系统服务，因此用户自己的应用程序不应该试图去使用小于 1023 的端口。TCP 和 UDP 协议就是使用端口号来把数据包发送给运行在计算机上的某个处理进程的。

Java UDP 有两种通信方式，一种是点对点的传播方式，另一种叫多点传播方式。下面就两种方式的编程进行论述。下列程序都在 JDK 1.4.0 环境下编译通过。

二、Java UDP 点对点传送

这种方式的特点是两个应用程序通过 UDP 进行数据的传送。我们用一个关于网络时间的例子进行说明。程序由服务器端和客户端两部分组成：服务器端用来提供网络标准时间，当收到客户端的一个请求包以后就向客户端发送一个当前时间的数据包。客户端用于向服务器发送时间请求，并把服务器发送过来的时间在客户端显示出来。程序运行说明：先在一台电脑上启动服务器程序，启动方法为运行“java TimeServer”；然后启动客户端程序运行“java TimeClient < 服务器的 IP 地址>”。

1. 服务器程序 TimeServer.java

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.text.DateFormat;
public class TimeServer{
   protected static DatagramSocket socket = null;
   public static void main(String []args){
     while(true){
       try {
         socket = new DatagramSocket(5555);
         byte[] buf = new byte[128];
         //创建一个数据包
         DatagramPacket packet = new DatagramPacket(buf, buf.length);//
         socket.receive(packet); //监听来自 Client 的数据包
```

```
            String dString = null;
            Date d = new Date(); //取得当前的日期与时间
            dString = DateFormat.getDateTimeInstance().format(d);
            buf = dString.getBytes();
            InetAddress address = packet.getAddress(); //取得 Client 的 IP 地址
            int port = packet.getPort(); //取得 Client 的端口号
            //构造一个发送到 Client 的数据包
            packet = new DatagramPacket(buf, buf.length, address, port);
            socket.send(packet); //发送数据包
            System.out.println("Requiring from: " + address.getHostAddress());
            //打印 Client 的 IP 地址
          } catch (IOException e) {
            e.printStackTrace();
          } finally {
            socket.close();
          }
        }
      }
    }
```

2. 客户端程序 TimeClient.java

```
import java.io.*;
import java.net.*;
import java.util.*;
public class TimeClient {
  public static void main(String[] args) throws IOException {
    if (args.length != 1) {
      System.out.println("Usage: java TimeClient <hostname>"); //程序的使用方法
      return;
    }
    DatagramSocket socket = new DatagramSocket();
    byte[] buf = new byte[128];
    InetAddress address = InetAddress.getByName(args[0]); //取得 Server 的 IP 地址
    //创建一个发送到 Server 的数据包
    DatagramPacket packet = new DatagramPacket(buf, buf.length, address, 5555);
    socket.send(packet); //发送数据包
    packet = new DatagramPacket(buf, buf.length);
    socket.receive(packet); //接收来自 Server 的数据包
    // 显示数据包
    String received = new String(packet.getData());
    System.out.println("Net Time is: " + received.trim());
    socket.close();
  }
}
```

三、Java UDP 多点传送

IP 多点传送是通过类 MulticastSocket 来实现的，MulticastSocket 是 DatagramSocket 的子类。所谓多点传送是指在一个组内对其成员进行的广播，是一种有限的广播。发往组的数据包，组内的成员都能收到。为了支持 IP 多点传送，某些范围的 IP 地址被单独留出专门用于这个目的，这些 IP 地址是 D 类地址，其地址的最高四位的位模式为 "1110"，即 IP 地址的范围为 224.0.0.0～

此为试读，需要完整PDF请访问：www.ertongbook.com