新编21世纪高职高专计算机系列规划教材

C#程序设计

北京希望电子出版社　总策划
李德奇　主　编
罗雅丽　何　颖　副主编

科学出版社
www.sciencep.com

新编21世纪高职高专计算机系列规划教材

C#程序设计

北京希望电子出版社　总策划
李德奇　主　编
罗雅丽　何　颖　副主编

科学出版社
www.sciencep.com

内 容 简 介

本书通过大量实例，详细介绍了C#的基础知识和Windows应用程序的编程方法。全书共7章，主要内容包括：.NET的概念和C#编程基础、Windows应用程序的用户界面设计方法、图形图像处理基础、C#下的数据库编程，最后有一个综合应用实例，详细讲述了用C#开发管理信息系统的过程和方法。

本书内容翔实，实例丰富，注释详细，便于读者尽快掌握C#的编程方法和技巧。

本书可作为高职高专学校计算机及相关专业的教材，也可供具有一定面向对象程序设计和.NET编程经验的读者参考使用。

需要本书或技术支持的读者，请与北京中关村083信箱（邮编：100080）发行部联系，电话：010-82702660 010-82702658 010-62978181 转103，传真：010-82702698，E-mail：tbd@bhp.com.cn。

图书在版编目（CIP）数据

C#程序设计 / 李德奇主编. —北京：科学出版社，2005.8
新编21世纪高职高专计算机系列规划教材
ISBN 7-03-015754-0

Ⅰ.C… Ⅱ.李… Ⅲ.C语言—程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字（2005）第066012号

责任编辑：王超辉 / 责任校对：周 玉
责任印刷：双 青 / 封面设计：刘孝琼

科学出版社 出版
北京东黄城根北街16号
邮政编码:100717
http://www.sciencep.com
双青印刷厂 印刷

科学出版社发行 各地新华书店经销

*

2005年8月第 一 版 开本：787×1092 1/16
2005年8月第一次印刷 印张：18
印数：1-3000册 字数：414 000

定价：26.00元

新编21世纪高职高专计算机系列规划教材编委会

总　序

一本好书，是人生前进的阶梯；一套好教材，就是教学成功的保证。为满足培养应用型人才的需要，我们成立了本编委会。在明确高职高专应用型人才培养模式、培养目标、教学内容和课程体系的框架下，我们组织编写了本套规划教材。

为了使本套教材能够达成目标，编委会做了大量的前期调研工作，在广泛了解各高职高专学校的教学现状、学生水平、培养目标的情况下，认真探讨了课程设置，研究了课程体系。为了编写出符合教学需求的好教材，我们除了聘请一批计算机知名专家、教授作为本套教材的主审和编委外，还组织了一批具备较高的学术水平、丰富的教学经验、较强的工程实践能力的学术带头人和骨干教师来承担具体编写工作，从而编写出特色鲜明、适用性强的教材，以真正满足目前高职高专学校应用型人才培养的需要。教材编写采用整体规划、分步实施、在实践中检验提高的方式，分期分批地启动编写计划。编写大纲以及教材编写方式的确定均经过编委会多次认真讨论，以确保该套教材的高质量和实用性。

本套规划教材的主要特点是：

（1）以服务教学为最高宗旨，认真做好教学内容的取舍、教学方法的选取、教学成果的检验工作。本套教材在教学过程中的有益反馈，都将及时体现在后续版本。

（2）面向应用型高职高专，在保证学科体系完整的基础上把握好理论的深度和难度。注重理论知识与实践相结合，使学生通过实践深化对理论的理解，学会并掌握理论方法的实际运用。从而较好地培养学生的专业技能和实施工程的实用技术能力。

（3）教材在内容编排上，力求由浅入深，循序渐进；举一反三，突出重点；语言简练，通俗易懂。采用模块化结构，兼顾不同层次的需求，在具体授课时可根据具体教学计划适当取舍内容。

（4）教材采用“任务驱动”的编写方式，以实际问题引出相关原理和概念，在讲述实例的过程中将本章的知识点融入，通过分析归纳，介绍解决工程实际问题的思想和方法，同时，引入案例教学和启发式教学方法，便于激发学习兴趣。

（5）在教材中加大实训部分的比重，使学生能比较熟练地应用计算机知识和技术解决实际问题，既注重培养学生分析问题的能力，也注重培养学生解决问题的能力。

（6）大部分教材配有电子教案，从而更好地服务教学。

为编写本套教材，作者们付出了艰辛的劳动，编委会的各位专家进行了悉心的指导和认真的审定。书中参考、借鉴了国内外同类的优秀教材和专著，在此一并表示感谢。

我们衷心希望更多的优秀教师参与到教材建设中来，真诚希望广大教师、学生与读者朋友在使用本丛书过程中提出宝贵意见和建议。

若有投稿或建议，请发电子邮件到 textbook@bhp.com.cn。谢谢！

新编 21 世纪高职高专计算机系列规划教材编委会

前　言

自从Microsoft公司展示它的新一代软件开发工具——Visual Studio .NET以来，就备受IT界的关注。目前，学习和使用Visual Studio .NET的计算机软件爱好者和从业人员越来越多，可见其技术的先进性和旺盛的生命力。Visual Studio .NET可以支持20多种语言编写程序，最常用的有Visual Basic、Visual C#、Visual J#和Visual C++，加强了语言的平台无关性，提高了程序的可移植性。

Visual C#作为Visual Studio .NET的原生语言，更是体现了Visual Studio .NET的特点。C#并不是从C或C++派生而来，而是Microsoft彻底从头开发的一种可视化的完全面向对象的编程语言。它汇集了Microsoft公司的技术精华，具有简单、先进、类型安全、面向对象和可视化等特点。使用C#可以完成任何程序设计需要，程序规模从小型的桌面应用程序到大型的分布式应用系统，程序运行形式可以是Windows应用程序、Web应用程序、Windows服务程序、Web服务程序等，也可以使用C#开发组件和进行网络编程。

较之其他可视化的面向对象编程语言，C#除了编译器的功能强大之外，吸引程序员的动人之处还在于开发工具的易用性和编辑器的方便性。C#程序是运行在.NET Framework下的受控代码，它是Visual Studio .NET下一种全新的编程语言，因而在编程概念、程序设计方法和程序表现形式上与其他编程语言既有许多共性，也有许多独特之处。C#程序代码的形式与C++极其相似，但从编程概念上更接近Java，因此具有C++或Java编程基础的人学习C#也并不是十分困难的事。

本书主要介绍了.NET Framework的基本概念、C#语言基础、控件和对话框、Windows应用程序用户界面设计、图形图像处理基础、数据库编程方法，最后通过一个综合应用实例，详细讲述了用C#开发管理信息系统的过程和方法。学完本书的内容，读者可以掌握使用C#编写Windows桌面应用程序和C/S模式的管理信息系统的程序设计方法。全书共分7章，其结构安排如下：

第1章“.NET概述”，介绍了.NET的基本概念和Visual Studio .NET的使用方法。

第2章“C#语言编程基础”，全面介绍了C#语言的编程规范和面向对象程序设计方法，特别是类的结构和类机制。

第 3 章“控件和对话框设计”，详细介绍了常用控件的应用和对话框的设计方法，主要包括窗体设计和各种控件的应用。

第 4 章“Windows 应用程序界面设计”，详细介绍了菜单、工具栏和状态栏等 Windows 应用程序窗口组成元素的设计，以及协调窗口之间关系的方法。

第 5 章“图形图像处理”，介绍了图形绘制方法和简单的图像处理方法。

第 6 章“ADO.NET 数据库编程”，详细介绍了 C#下数据源的连接、数据访问和数据处理方法。

第 7 章“综合应用实例——图书馆管理信息系统”，通过“图书馆管理信息系统”这个完整的实例，详细介绍了使用 C#开发管理信息系统的过程和方法。

本书可以作为高职高专院校相关专业的教材和教学参考用书，也可以作为相关领域技术人员的参考资料。

本书写作中融合了作者多年的软件开发和教学经验，加入了近年来对 C#的理解和实践。本书力求做到深入浅出、联系实际，使读者通过实例来学习和掌握 C#这门面向对象编程语言。参加本书编写工作的有李德奇、罗雅丽、何颖、翁建红、刘志成、袁开鸿、周启亚、张杰、张群哲、田梦等人，对于参与本书策划和组织工作的彭勇、梁洁婷等同志，在此一并表示衷心的感谢。

由于水平和经验有限，本书的不足和纰漏之处在所难免，恳请广大读者批评指正。

编　者

目　录

第 1 章　.NET 概述

讲述 C#语言不可以将它孤立地对待，而必须与.NET Framework 一起讨论。C#编译器不能脱离.NET，这表明用 C#开发的程序必须在.NET Framework 中运行。.NET Framework 的运行机制制约着 C#编译器，C#编译器主导着 C#语言规范，通常 C#被集成于 Visual Studio .NET（简称 VS.NET）之中。

由于这种依赖性，在学习 C#语言编程之前，了解.NET 的结构和方法论就显得非常重要了。但对于不熟悉面向对象编程类机制的读者，不容易理解本章前两小节的内容，建议学完第 2 章后再来阅读这两节，可能有助于理解。

在这一章，将学习到：

- .NET Framework 的基本知识
- .NET 基类库的结构
- Visual Studio .NET 的使用方法
- C#程序调试

1.1　.NET Framework 和公共语言运行库

1.1.1　.NET 结构

.NET Framework 称为.NET 框架，它是一种托管的、类型安全的代码执行环境。.NET Framework 管理程序执行的各个方面：启动代码，给它赋予相应的权限，为数据和指令分配内存，对不再需要的内存进行回收管理。图 1.1 显示了.NET 的层次结构。

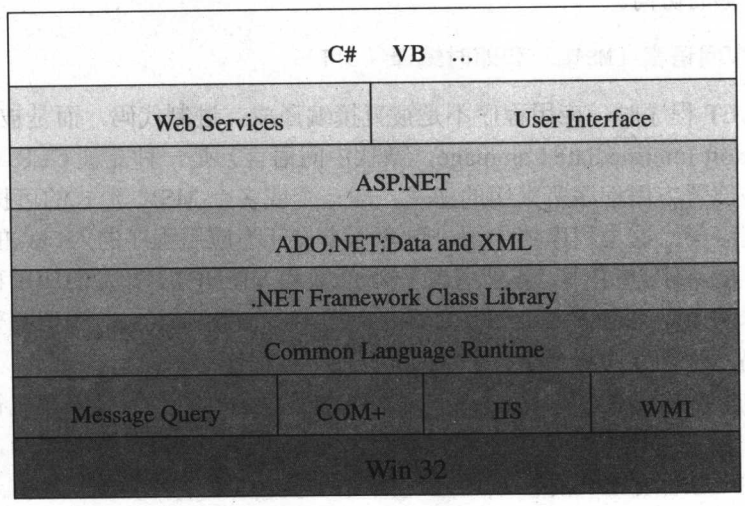

图 1.1　.NET 的层次结构

图中的第 3 层和第 4 层分别称为公共语言运行库和.NET 框架类库，它们是.NET 的核心基础。

在公共语言运行库和.NET 框架类库之上有 ADO.NET、ASP.NET 和 Web Services 等基类库，这个庞大的基类库为编程任务提供了对象模型和服务。.NET 基类库提供的大多数类型是安全和可扩展的，也可以通过继承这些基类派生出自己的类来。

图 1.1 的顶层是 Visual Studio .NET 的应用程序开发工具，支持像 C#、VB.NET 和 Visual C++ .NET 等 Microsoft 语言编程。此外，也有其他公司提供的用于 Visual Studio .NET 的编译器。

图 1.1 中最下面两层被.NET 封装，意味着在.NET 中编程时，托管代码中不能调用 Win32 API 函数，也不能使用指针。

1.1.2 公共语言运行库

CLR（Common Language Runtime，公共语言运行库）负责管理和执行.NET 框架代码，符合 Visual Studio .NET 编译器规则的代码在执行时需要 CLR 的支持，这些代码称为托管代码（Managed Code）。CLR 的管理工作包括以下几项：

- 确定加载代码的方式和时机，并且管理对象在内存中的分配。代码执行时，CLR 将决定代码所需要的类和方法，并且按照需要将原先编译过的中间语言（IL）进行即时编译（JIT），形成可执行代码。
- 对托管代码所占用的内存进行管理，通过垃圾收集器（GC）实现无用内存的回收。
- 确保代码（由 CLR 执行的代码）的类型安全。CLR 负责确保以安全的方式使用类和其他类型。
- 通过使用基于异常处理机制的通用错误处理方式，可以对托管代码中的错误进行处理和传递。
- 维护 CLR 和应用程序的安全性。在.NET 框架中，安全性有两种形式：代码访问的安全性和角色的安全性。前者确保在安全的上下文中执行代码，后者控制对系统资源的访问。

1.1.3 微软中间语言（MSIL）和即时编译（JIT）

当编译.NET 程序时，应用程序不是被直接编译成二进制代码，而是被编译成 MSIL。MSIL（Microsoft Intermediate Language，微软中间语言）是一种能被 CLR 理解的相当低级的指令集，是部署应用程序所采用的形式，即一个或多个 MSIL 形式的可执行文件和 DLL 文件组成的程序集。这个程序集中至少包含一个设计为应用程序的入口点的可执行文件。

在 Visual Studio .NET 下，各种语言（例如 C#和 VB.NET）开发的应用程序都会被编译成运行时环境的通用语言——中间语言，当应用程序加载后，CLR 根据需要将其再次编译成可执行代码。这种编译称为 JIT（Just-in-Time）即时编译。

MSIL 有点像 Java 的字节代码，不同之处在于，Java 的字节代码在执行时是逐字节解释，而 MSIL 会被以模块为单位编译为本地二进制代码。代码一旦被编译，就开始执行并且作为本地代码被保存于内存区。因此，代码的每一部分仅被编译一次，无论何时程序执行扩展到被执行的代码，JIT 编译器将在执行前编译它并且将它作为本地二进制代码存储在

内存中。是说，不需要运行的代码不会被编译，执行过的代码不需要再次编译，应用程序的性能得到了最大的优化。

由于中间语言在.NET 下的通用性，就使得在 Visual Studio .NET 下开发程序时，在一种语言中可以使用另一种语言开发的类。例如，C#可以引用 VB.NET 的类，原因是 VB.NET 的类被编译成 MSIL，C#程序也被编译成 MSIL，它们在 CLR 中当然是通融的。

1.1.4　无用内存单元收集器（GC）

GC（Garbage Collection）称为无用内存单元收集器，也叫垃圾收集器。有了 GC，当一个对象不再被使用时，.NET 框架能够自动地回收那个对象使用的内存，这个过程就称为垃圾回收。

垃圾回收的原理来自于.NET 的数据类型与内存分配机制。.NET 将数据类型分为值类型和引用类型两大类：值类型的例子包括基元类型，例如整型（int）、布尔型（bool）、字符型（char）等，还有一些用户定义的结构类型（struct）和枚举类型（enum）。引用类型包括类、接口、委托和数组。值类型与引用类型的主要区别在于变量数据的访问方式。要理解这个区别，下面介绍一些内存分配的背景知识。

应用程序内存数据主要存放在两个区域：栈（Stack）和堆（Heap）。堆栈其实是一块内存，不过一端为堆，另一端为栈。

栈供函数调用使用。当某函数 fn1 被调用时，其所有的非静态变量分配在栈中，称为压栈。当又有函数 fn2 被调用时，fn2 也被压栈，这时 fn1 被压在 fn2 的底下。当对 fn2 的调用结束时，fn2 的所有变量被从栈中清除，称为弹出。等到 fn1 调用结束，fn1 也被弹出。所以栈是“先进后出”机制。而堆为创建可重用的对象而保留。

所有的值类型变量在栈上分配内存，当值类型的变量脱离了作用域时，它就会被销毁，收回其内存。创建引用类型对象时，对象的数据保存在堆中，但在栈中有一个指向那个对象的引用。由于对象是可重用的，一个对象也可能不止一个引用存在。当一个引用的生命期结束时，这个引用被从栈中弹出，这时并不能去销毁堆中与之联系的那个对象，因为可能还有别的引用与之联系着。GC 不时来搜索堆中对象的引用，当发现某个对象无任何引用与之联系时，断定这个对象是不必要的，就可将其从堆中销毁，收回其堆内存。

在正常情况下，垃圾回收是一个低优先级的线程。当处理器时间没有被更重要的任务占用时，它才会运行。但是当内存不够时，垃圾回收线程的优先级被提升，使无用内存被迅速回收，以缓解内存的紧张。此后，垃圾回收器线程的优先级又被降低了。

得益于垃圾回收机制，程序员不必像原先那样使用 new 操作后时时要记得使用 delete 操作，这样也使类的析构函数的任务大大减轻，内存泄露的可能性几乎不存在了。

采用这种非确定性的内存回收方法，是为了尽量提高应用程序的性能，并为应用程序提供出错较少的运行环境。但这样做也要付出代价。鉴于垃圾回收机制，所以不能确定对象何时被回收，所以无法控制何时执行一个类的析构函数，这样析构函数也就不能包含在指定时间运行的代码。实际上，需要较高资源开销的类通常要调用 Dispose()方法，在该类不再被使用时，显式地释放资源。

1.2 .NET 基类库

.NET 基类库是一个面向对象的类型和接口的集合，它为很多复杂的编程任务提供了对象模型和服务。.NET 基类库提供的大多数类型是安全可扩展的，使程序员可以由此派生新的类型，加入自己的功能合并到托管代码之中。

1.2.1 .NET 的命名空间

命名空间又称名空间或名字空间，它是组织应用程序的一种结构。使用命名空间是为了避免程序中类名的冲突。例如，某个程序员定义一个类来表示一个顾客，称这个类为 Customer。又有另一个程序员也定义了一个 Customer 类。编译器如何来区分是哪一个 Customer 呢？只要将它们放在两个不同的命名空间就可识别两个不同的 Customer。

namespace 关键字

命名空间用关键字 namespace 来声明一个范围，可以在命名空间的范围内来组织程序代码。任何一段代码都必须放在某一个命名空间内，以防止类名的冲突。使用命名空间的语法为：

```
namespace name[.name1...]
{
        type declarations          //类型声明
}
```

其中，name、name1 是命名空间的名字，它可以是任何合法的标识符，命名空间的名字中可以包含圆点，圆点确定命名空间的层次。

命名空间在默认情况下会自动创建，它的名字就是项目名，此时的命名空间称为全局命名空间。每个文件都有这样一个命名空间。命名空间可以被公开访问，但不能被修改。

在命名空间中，可以声明一个或多个以下的类型：

- 另一个命名空间
- class
- interface
- struct
- enum
- delegate

既然可以在一个命名空间中创建另一个命名空间，表明命名空间是可以嵌套的。嵌套是为了理清名空间的层次，嵌套也可以用圆点表示。例如命名空间

```
namespace name
{
        namespace name1
        {
            class Customer
{
    ...
}
}
    }
```

与以下命名空间等价：

```
namespace name.name1
{
        class Customer
{
    ...
}
}
```

有了命名空间，类名可以用命名空间引导。例如 name.name1.Customer 出现在程序的任何地方都不会混淆它的出处。

using 关键字

using 用于物理打包和部署程序。using 具有两个用途，一是为一个命名空间创建一个别名，一是允许在一个命名空间中使用另一个命名空间的类，这样不会受到在命名空间中使用类的限制。

为一个命名空间创建一个别名可以简化命名空间。例如，如果觉得 System.Windows.Forms 这个命名空间太长不容易写，可以使用 using 关键字来简化它：

```
using myspace=System.Windows.Forms;
```

此后，在程序的任何地方见到 myspace 就是指 System.Windows.Forms。

using 的另一个作用是告诉编译系统，它下面使用的类名出自哪些命名空间。例如 C# 的 Windows 应用程序的头部总会见到以下几行：

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
```

这表明此后的程序中使用的类名来自以上的名空间。这样，如有程序行：

```
private System.Windows.Forms.Button button1;
```

则可以简化成：

```
private Button button1;
```

因为名空间 System.Windows.Forms 在程序头部已经声明过，编译系统知道 Button 类的出处。若在程序中使用.NET 的基类名而编译时报错，请检查该类名的拼写是否有错，再检查该类所在的名空间是否已声明。若是缺少名空间的声明，请补加 using 声明，或者在类名前一并写上它的名空间。

using 语句是惟一允许写在任何花括号{}之外的语句，并且其后必须带有分号“;”。

1.2.2 .NET 基类库

.NET 基类库是 Microsoft 已经编写好的一个内容丰富的受管制的类代码集合，它可以完成以前要通过 Windows API 来完成的绝大多数任务，使程序员从烦琐的 API 函数调用中解脱出来，专心于应用程序事务的处理。这些类派生于与中间语言相同的对象模型，也基于单一继承性，每个类都可以实例化对象，也可以从中派生自己的类。在较简单的 Windows 编程中，程序员很少从头设计自己的新类。要么继承.NET 基类派生自己的类（设计窗体类就是最好的例子），要么由.NET 基类直接创建对象（在窗体中使用控件正是如此）。

.NET 基类的另一个优点就是它的易用性，且都是自描述性的。例如求一个数的平方根，可以调用 Math 类的静态方法 Sqrt()（Math.Sqrt()），要取得当前的日期和时间,可以使用 DateTime 类的静态属性 Now（DateTime.Now），要使用一个按钮，只需创建一个 Button 类的对象等等。

Microsoft 尽量代替从事应用系统开发的程序员去做那些重复而繁杂的、有时又是难度很大的工作，其结果是将成果以.NET 基类的方式公开出来供程序员共享。.NET 基类库是一个庞大的类集，基本涵盖了编程的各个方面：

- IL 提供的核心功能，例如公共类型系统中的基本数据类型
- Windows GUI 支持、控件等
- Web 窗体及控件（ASP.NET）
- 数据访问（ADO.NET）
- 目录访问
- 文件系统和注册表访问
- 联网和 Web 浏览
- .NET 属性和反射
- 访问 Windows 操作系统和环境变量
- 访问不同语言的源代码和编译器
- COM 互操作性
- 图形技术（GDI+）

这些基类被组织在一系列的名空间中，这些名空间是按照类的功能层次划分的。.NET 框架的根（Root）是 System，其他的名空间都在其下若干层次，用“.”运算符进行层次访问。典型的名空间结构如下：

```
System
System.Data
System.Data.SQLClient
```

表 1.1 介绍了一些通用的基类命名空间。

表 1.1　一些有代表性的.NET 命名空间

命名空间	说　明
System	它是.NET 框架需要的很多低级别类型的根，也是基元数据类型的根，它还是.NET 基类库中所有其他命名空间的根
System.Collections	它包含表示不同容器和集合类型的类，例如 ArrayList 等，也可以找到对实现自己的集合功能非常有用的抽象类，例如 CollectionBase
System.ComponentModel	它包含涉及组件创建和包容的类，例如特征、类型转换器和许可提供程序等
System.Data	它包含数据库访问和操作的类，其下还有附加的命名空间
System.Data.Common	它包含一组由.NET 托管数据提供程序共享的类
System.Data.OleDb	它包含用于为 OLE DB 数据访问的类
System.Data.SqlClient	它包含用于为 SQL Server 数据访问的类
System.Drawing	它公开了 GDI+功能并提供用于绘图的类
System.IO	它提供用于文件系统 I/O 的类

续表

命名空间	说　明
System.Math	它包含常用的数学函数
System.Reflection	它支持在运行时获取信息和动态地创建类
System.Security	它包含用于处理权限、密码系统以及代码访问安全的类型
System.Threading	它包含用于实现多线程应用程序的类
System.Windows.Forms	它包含与创建标准 Windows 应用程序有关的类型。表示窗体和控件的类也在此名空间

1.3　Visual Studio .NET 的初步使用

Microsoft Visual Studio .NET 除了包含有.NET Framework 外，还带有多种编程语言，如 C#（读音为 C Sharp）、Visual Basic .NET、Visual C++ .NET 和 J#等。除了 Microsoft 外，其他公司也提供了用于 Visual Studio .NET 的编译器。所有基于这些语言的编程都可以使用 Visual Studio .NET 所提供的工具和功能。这些工具和功能包括集成开发环境（Integrated Development Environment，IDE）的所有设计器和工具窗口，以及集成帮助系统。

1.3.1　起始页

启动 Visual Studio .NET 后，首先看到的是一个起始页（见图 1.2），通过该页既可以使用本地计算机上的.NET 项目，也可以访问到本地计算机以外的信息和服务。起始页上的有些选项卡需要连接到 Internet 才能使用，这些选项卡提供了关于 Visual Studio .NET 的最新信息和最新下载的链接。此外，用户也可以简单地查找和注册 Web Service。若启动 Visual Studio .NET 后看不到起始页，可以单击帮助菜单中的“显示起始页”；若想隐去起始页，可单击起始页上的“×”按钮关闭起始页。

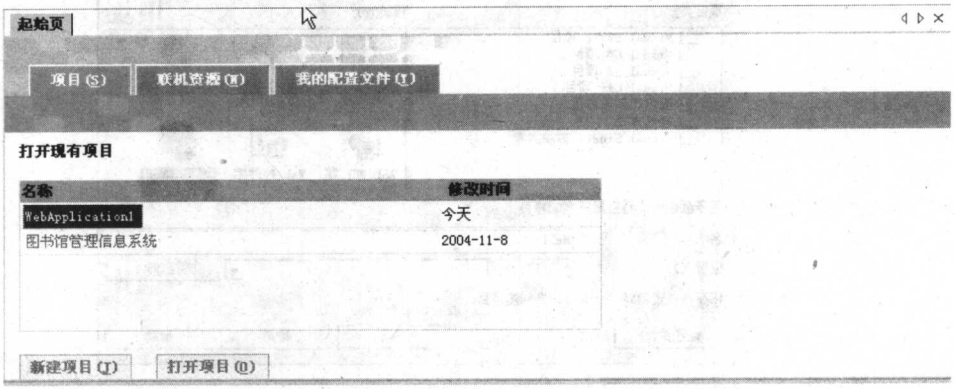

图 1.2　起始页