

# 第21章

---

## 软件安全控制

- » 软件的注册
- » 软件的加密

## 21.1 软件的注册

## 实例 593

## 利用 INI 文件对软件进行注册

光盘位置: 光盘\MR\21\593

中级

趣味指数: ★★★★★

## 实例说明

本实例主要实现使用 INI 文件对软件的用户信息进行注册的功能。运行本程序, 输入登录名称、登录口令和注册码, 单击“注册”按钮进行注册, 如果注册成功, 则给出提示; 如果信息已注册, 系统给出提示信息。实例运行效果如图 21.1 所示。

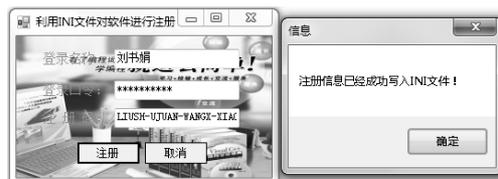


图 21.1 利用 INI 文件对软件进行注册

## 关键技术

本实例实现时主要用到了系统 API 函数 `GetPrivateProfileString` 和 `WritePrivateProfileString`, 下面分别对它们进行详细讲解。

(1) `GetPrivateProfileString` 函数

该函数主要用来读取 INI 文件的内容, 其语法格式如下:

```
[DllImport("kernel32")]
private static extern int GetPrivateProfileString(string lpAppName, string lpKeyName, string lpDefault, StringBuilder lpReturnedString, int nSize, string lpFileName);
```

`GetPrivateProfileString` 函数语法中的参数及说明如表 21.1 所示。

表 21.1 `GetPrivateProfileString` 函数语法中的参数及说明

参 数	说 明
<code>lpAppName</code>	表示 INI 文件内部根节点的值
<code>lpKeyName</code>	表示根节点下子标记的值
<code>lpDefault</code>	表示当标记值未设定或不存在时的默认值
<code>lpReturnedString</code>	表示返回读取节点的值
<code>nSize</code>	表示读取的节点内容的最大容量
<code>lpFileName</code>	表示文件的全路径

(2) `WritePrivateProfileString` 函数

该函数主要用于向 INI 文件写入数据, 其语法格式如下:

```
[DllImport("kernel32")]
private static extern long WritePrivateProfileString(string mpAppName, string mpKeyName, string mpDefault, string mpFileName);
```

`WritePrivateProfileString` 函数语法中的参数及说明如表 21.2 所示。

表 21.2 `WritePrivateProfileString` 函数语法中的参数及说明

参 数	说 明
<code>mpAppName</code>	表示 INI 文件内部根节点的值
<code>mpKeyName</code>	表示将要修改的标记名称
<code>mpDefault</code>	表示想要修改的内容
<code>mpFileName</code>	表示 INI 文件的全路径

说明: 程序中使用 API 函数时, 首先需要在命名空间区域添加 `System.Runtime.InteropServices` 命名空间, 下面遇到类似情况时将不再提示。

## 设计过程

(1) 打开 Visual Studio 2008 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 RegSoftByINI。

(2) 更改默认窗体 Form1 的 Name 属性为 Frm\_Main，在该窗体中添加 3 个 TextBox 控件，分别用来输入登录名称、登录口令和注册码；添加两个 Button 控件，分别用来执行注册和清空文本框操作。

(3) 程序主要代码如下。

Frm\_Main 窗体的后台代码中，首先对 API 函数 GetPrivateProfileString 进行重写，代码如下：

```
#region 为 INI 文件中指定的节点取得字符串
/// <summary>
/// 为 INI 文件中指定的节点取得字符串
/// </summary>
/// <param name="lpAppName">欲在其中查找关键字的节点名称</param>
/// <param name="lpKeyName">欲获取的项名</param>
/// <param name="lpDefault">指定的项没有找到时返回的默认值</param>
/// <param name="lpReturnedString">指定一个字符串缓冲区，长度至少为 nSize</param>
/// <param name="nSize">指定装载到 lpReturnedString 缓冲区的最大字符数量</param>
/// <param name="lpFileName">INI 文件名</param>
/// <returns>复制到 lpReturnedString 缓冲区的字节数量，其中不包括那些 NULL 中止字符</returns>
[DllImport("kernel32")]
public static extern int GetPrivateProfileString(string lpAppName,string lpKeyName,string lpDefault,StringBuilder lpReturnedString,int nSize,string lpFileName);
#endregion
```

对 API 函数 WritePrivateProfileString 进行重写的实现代码如下：

```
#region 修改 INI 文件中内容
/// <summary>
/// 修改 INI 文件中内容
/// </summary>
/// <param name="lpApplicationName">欲在其中写入的节点名称</param>
/// <param name="lpKeyName">欲设置的项名</param>
/// <param name="lpString">要写入的新字符串</param>
/// <param name="lpFileName">INI 文件名</param>
/// <returns>非零表示成功，零表示失败</returns>
[DllImport("kernel32")]
public static extern int WritePrivateProfileString(string lpApplicationName,string lpKeyName,string lpString,string lpFileName);
#endregion
```

自定义一个返回值类型为 string 的 ReadString 方法，该方法用来从 INI 文件中读取指定节点的内容，其实现代码如下：

```
#region 从 INI 文件中读取指定节点的内容
/// <summary>
/// 从 INI 文件中读取指定节点的内容
/// </summary>
/// <param name="section">INI 节点</param>
/// <param name="key">节点下的项</param>
/// <param name="def">没有找到内容时返回的默认值</param>
/// <param name="def">要读取的 INI 文件</param>
/// <returns>读取的节点内容</returns>
public string ReadString(string section, string key, string def, string fileName)
{
    StringBuilder temp = new StringBuilder(1024); //创建可变字符串对象
    GetPrivateProfileString(section, key, def, temp, 1024, fileName); //获取节点内容
    return temp.ToString(); //返回获取到的节点内容
}
#endregion
```

Frm\_Main 窗体加载时，从 INI 文件中获取已注册的内容，并显示在相应的 TextBox 文本框中，代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    textBox1.Text = ReadString("Regedit", "Name", "", strPath); //显示登录名称
    textBox2.Text = ReadString("Regedit", "Pwd", "", strPath); //显示登录口令
    textBox3.Text = ReadString("Regedit", "RegCode", "", strPath); //显示注册码
}
```

当用户输入登录名称、登录口令和注册码之后,单击“注册”按钮,首先判断是否已经有相同的信息注册过,如果已经注册过,弹出提示信息;否则,将用户输入的信息写入到 INI 文件中,从而完成软件的注册功能。

“注册”按钮的 Click 事件代码如下:

```
private void button1_Click(object sender, EventArgs e)
{
    string strName = ReadString("Regedit", "Name", "", strPath);           //获取登录名称
    string strPwd = ReadString("Regedit", "Pwd", "", strPath);           //获取登录口令
    string strRegcode = ReadString("Regedit", "RegCode", "", strPath);    //获取注册码
    if (strName == textBox1.Text && strPwd == textBox2.Text && strRegcode == textBox3.Text) //判断 INI 文件中是否已经存在相同信息
    {
        MessageBox.Show("此信息已注册过!");
    }
    else
    {
        WritePrivateProfileString("Regedit", "Name", textBox1.Text, strPath); //向 INI 文件中写入登录名称
        WritePrivateProfileString("Regedit", "Pwd", textBox2.Text, strPath); //向 INI 文件中写入登录口令
        WritePrivateProfileString("Regedit", "RegCode", textBox3.Text, strPath); //向 INI 文件中写入注册码
        MessageBox.Show("注册信息已经成功写入 INI 文件!", "信息");
    }
}
```

## 秘笈心法

心法领悟 593: 获取 INI 文件的节点内容。

获取 INI 文件的内容时用到了 GetPrivateProfileString 函数,该函数用来读取 INI 文件的内容,其语法格式如下:

```
[DllImport("kernel32")]
private static extern int GetPrivateProfileString(string lpAppName, string lpKeyName, string lpDefault, StringBuilder lpReturnedString, int nSize, string lpFileName);
```

## 实例 594

### 利用注册表设计软件注册程序

光盘位置: 光盘\MR\21\594

高级

趣味指数: ★★★★★

## 实例说明

大多数应用软件会将用户输入的注册信息写进注册表中,程序运行过程中,可以将这些信息从注册表中读出。本实例主要实现在程序中对注册表进行操作的功能,运行程序,单击“注册”按钮,会将用户输入的信息写入注册表中。实例运行效果如图 21.2 所示。



图 21.2 利用注册表设计软件注册程序

## 关键技术

本实例实现时主要用到了 RegistryKey 类的 OpenSubKey 方法、CreateSubKey 方法、GetSubKeyNames 方法和 SetValue 方法,下面对本实例用到的关键技术进行详细讲解。

### (1) OpenSubKey 方法

RegistryKey 类表示 Windows 注册表中的项级节点,该类是注册表封装类,其 OpenSubKey 方法用来检索指定的子项,该方法的语法格式如下:

```
public RegistryKey OpenSubKey (string name, bool writable)
```

参数说明

- ❶ name: 要打开的子项的名称或路径。
- ❷ writable: 如果需要项的写访问权限,则设置为 true。
- ❸ 返回值: 请求的子项;如果操作失败,则为空引用。

### (2) CreateSubKey 方法

该方法主要用来创建一个新子项或打开一个现有子项以进行写访问，其语法格式如下：

```
public RegistryKey CreateSubKey (string subkey)
```

参数说明

❶ subkey: 要创建或打开的子项的名称或路径。

❷ 返回值: RegistryKey 对象，表示新建的子项或空引用。如果为 subkey 指定了零长度字符串，则返回当前的 RegistryKey 对象。

### (3) GetSubKeyNames 方法

该方法主要用来检索包含所有子项名称的字符串数组，其语法格式如下：

```
public string[] GetSubKeyNames ()
```

参数说明

返回值: 包含当前项的子项名称的字符串数组。

### (4) SetValue 方法

该方法主要用来设置指定的名称/值对，其语法格式如下：

```
public void SetValue (string name, Object value)
```

参数说明

❶ name: 要存储的值的名称。

❷ value: 要存储的数据。

 说明: 程序中使用注册表相关的类（如 RegistryKey 类和 Registry 类）时，首先需要在命名空间区域添加 Microsoft.Win32 命名空间，下面遇到类似情况时将不再提示。

## 设计过程

(1) 打开 Visual Studio 2008 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 RegSoftByRegedit。

(2) 更改默认窗体 Form1 的 Name 属性为 Frm\_Main，在该窗体中添加 3 个 TextBox 控件，分别用来输入公司名称、用户名称和注册码；添加两个 Button 控件，分别用来执行注册和清空文本框操作。

(3) 程序主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    if (textBox1.Text == "") //判断公司名称是否为空
    {
        MessageBox.Show("公司名称不能为空");
        return;
    }
    if (textBox2.Text == "") //判断用户名称是否为空
    {
        MessageBox.Show("用户名称不能为空");
        return;
    }
    if (textBox3.Text == "") //判断注册码是否为空
    {
        MessageBox.Show("注册码不能为空");
        return;
    }
    Microsoft.Win32.RegistryKey retkey1 = Microsoft.Win32.Registry.CurrentUser.OpenSubKey("software", true).CreateSubKey("WXK").CreateSubKey("WXK.INI"); //创建 RegistryKey 对象
    foreach (string strName in retkey1.GetSubKeyNames()) //判断注册码是否过期
    {
        if (strName == textBox3.Text) //如果注册表中已经存在
        {
            MessageBox.Show("此注册码已经过期");
            return;
        }
    }
    Microsoft.Win32.RegistryKey retkey = Microsoft.Win32.Registry.CurrentUser.OpenSubKey("software", true).CreateSubKey("WXK").CreateSubKey
```

```

("WXX.INI").CreateSubKey(textBox3.Text.TrimEnd());
retkey.SetValue("UserName", textBox2.Text);
retkey.SetValue("capataz", textBox1.Text);
retkey.SetValue("Code", textBox3.Text);
MessageBox.Show("注册成功, 您可以使用本软件");
}

```

```

//在注册表中创建子项
//向注册表中写入公司名称
//向注册表中写入用户名称
//向注册表中写入注册码

```

## 秘笈心法

心法领悟 594: 加密应用系统软件中的所有数据。

为银行、政府和军队等机关单位开发管理软件, 由于这些单位涉及的都是机密数据, 所以需要将这些机密数据进行加密, 然后保存到数据库中。当读取数据库中的数据时, 进行解密。即使非法用户得到了数据库, 也无法浏览数据库中的数据。

## 实例 595

### 利用网卡序列号设计软件注册程序

光盘位置: 光盘\MR\21\595

高级

趣味指数: ★★★★★

## 实例说明

本实例实现了利用本机网卡序列号生成软件注册码的功能。运行本实例, 程序将自动获得本机网卡序列号, 单击“生成注册码”按钮, 生成软件注册码, 在下面的文本框中依次输入注册码, 单击“注册”按钮即可实现软件注册功能。实例运行效果如图 21.3 所示。



图 21.3 利用网卡序列号设计软件注册程序

## 关键技术

本实例实现时, 主要用到了 RegistryKey 类的 OpenSubKey 方法、CreateSubKey 方法、GetSubKeyNames 方法、SetValue 方法和 ManagementClass 类的 GetInstances 方法、ManagementObjectCollection 类和 ManagementObject 类, 下面对本实例中用到的关键技术进行详细讲解。

### (1) ManagementClass 类的 GetInstances 方法

ManagementClass 类表示公共信息模型 (CIM) 管理类。管理类是一个 WMI 类, 如 Win32\_LogicalDisk 和 Win32\_Process, 前者表示磁盘驱动器, 后者表示进程 (如 Notepad.exe)。通过该类的成员, 可以使用特定的 WMI 类路径访问 WMI 数据。ManagementClass 类的 GetInstances 方法用来返回该类的所有实例的集合, 其语法格式如下:

```
public ManagementObjectCollection GetInstances ()
```

参数说明

返回值: 表示该类实例的 ManagementObject 对象的集合。

### (2) ManagementObjectCollection 类

ManagementObjectCollection 类表示通过 WMI 检索到的管理对象的不同集合, 此集合中的对象为 ManagementBaseObject 派生类型, 包括 ManagementObject 和 ManagementClass。例如, 本实例中通过使用 ManagementClass 对象的 GetInstances 方法获取管理对象集合, 代码如下:

```
ManagementObjectCollection moc = mc.GetInstances();
```

```
//创建 ManagementObjectCollection 对象
```

### (3) ManagementObject 类

ManagementObject 类表示 WMI 实例, 本实例中用到了该类的 Item 属性, 该属性用来通过[]符获取对属性值的访问, 其语法格式如下:

```
public Object Item[
string propertyName
] { get; set; }
```

## 参数说明

- ❶ propertyName: 相关的属性的名称。
- ❷ 返回值: 返回一个 Object 值, 该值包含特定类属性的管理对象。

 说明: (1) 关于 RegistryKey 类的 OpenSubKey 方法、CreateSubKey 方法、GetSubKeyNames 方法和 SetValue 方法的详细讲解, 请参见实例 594 中的关键技术。

- (2) ManagementClass 类、ManagementObjectCollection 类和 ManagementObject 类都位于 System.Management 命名空间下, 添加该命名空间时, 首先需要在“添加引用”中添加 System.Management.dll 组件。

## 设计过程

(1) 打开 Visual Studio 2008 开发环境, 新建一个 Windows 窗体应用程序, 并将其命名为 RegSoftByNetwork Card。

(2) 更改默认窗体 Form1 的 Name 属性为 Frm\_Main, 在该窗体中添加 3 个 Label 控件, 分别用来显示计算机名称、网卡序列号和注册码; 添加 4 个 TextBox 控件, 用来输入注册码; 添加 3 个 Button 控件, 分别用来执行生成注册码、注册和退出操作。

(3) 程序主要代码如下:

```
private void button1_Click(object sender, EventArgs e)
{
    string strCode = GetNetCardMacAddress(); //调用自定义方法获取网卡信息
    strCode = strCode.Substring(0, 2) + strCode.Substring(3, 2) + strCode.Substring(6, 2) + strCode.Substring(9, 2) + strCode.Substring(12, 2) +
    strCode.Substring(15, 2);
    string strb = strCode.Substring(0, 4) + strCode.Substring(4, 4) + strCode.Substring(8, 4); //网卡信息存储
    for (int i = 0; i < strLanCode.Length; i++) //把网卡信息存入数组
    {
        strLanCode[i] = strb.Substring(i, 1);
    }
    Random ra = new Random();
    switch (intRand) //随机生成注册码的顺序
    {
        case 0: //当第一次生成随机注册码时执行
            label5.Text = strCode.Substring(0, 4) + "-" + strCode.Substring(4, 4) + "-" + strCode.Substring(8, 4) + "-" + strkey[ra.Next(0,
            37)].ToString() + strkey[ra.Next(0, 37)].ToString() + strkey[ra.Next(0, 37)].ToString() + strkey[ra.Next(0, 37)].ToString(); //生成随机注册码
            intRand = 1; //使变量 intRand 等于 1
            break;
        case 1: //第二次生成随机注册码时执行
            label5.Text = strCode.Substring(0, 4) + "-" + strCode.Substring(4, 4) + "-" + strLanCode[ra.Next(0, 11)] + strLanCode[ra.Next(0, 11)] +
            strLanCode[ra.Next(0, 11)] + strLanCode[ra.Next(0, 11)] + "-" + strkey[ra.Next(0, 37)].ToString() + strkey[ra.Next(0, 37)].ToString() + strkey[ra.Next(0,
            37)].ToString() + strkey[ra.Next(0, 37)].ToString();
            intRand = 2; //使变量 intRand 等于 2
            break;
        case 2: //第 3 次生成随机注册码时执行
            //生成随机注册码
            label5.Text = strCode.Substring(0, 4) + "-" + strLanCode[ra.Next(0, 11)] + strLanCode[ra.Next(0, 11)] + strLanCode[ra.Next(0, 11)] +
            strLanCode[ra.Next(0, 11)] + "-" + strLanCode[ra.Next(0, 11)] + strLanCode[ra.Next(0, 11)] + strLanCode[ra.Next(0, 11)] + strLanCode[ra.Next(0, 11)] +
            "-" + strkey[ra.Next(0, 37)].ToString() + strkey[ra.Next(0, 37)].ToString() + strkey[ra.Next(0, 37)].ToString() + strkey[ra.Next(0, 37)].ToString();

            intRand = 3; //使变量 intRand 等于 3
            break;
        case 3: //第 4 次生成随机注册码时执行
            label5.Text = strLanCode[ra.Next(0, 11)] + strLanCode[ra.Next(0, 11)] + strLanCode[ra.Next(0, 11)] + strLanCode[ra.Next(0, 11)] + "-" +
            strLanCode[ra.Next(0, 11)] + strLanCode[ra.Next(0, 11)] + strLanCode[ra.Next(0, 11)] + strLanCode[ra.Next(0, 11)] + "-" + strLanCode[ra.Next(0,
            11)] + strLanCode[ra.Next(0, 11)] + strLanCode[ra.Next(0, 11)] + strLanCode[ra.Next(0, 11)] + "-" + strkey[ra.Next(0, 37)].ToString() + strkey[ra.Next(0, 37)].ToString() +
            strkey[ra.Next(0, 37)].ToString() + strkey[ra.Next(0, 37)].ToString(); //生成随机注册码
            intRand = 0; //使变量 intRand 等于 0
            break;
    }
}
public string GetNetCardMacAddress()
```

```

{
    ManagementClass mc = new ManagementClass("Win32_NetworkAdapterConfiguration"); //创建 ManagementClass 对象
    ManagementObjectCollection moc = mc.GetInstances(); //创建 ManagementObjectCollection 对象
    string str = ""; //用于存储网卡序列号
    foreach (ManagementObject mo in moc) //遍历得到的集合
    {
        if ((bool)mo["IPEnabled"] == true) //判断 IPEnabled 属性是否为 true
            str = mo["MacAddress"].ToString(); //获取网卡序列号
    }
    return str; //返回网卡序列号
}

```

## 秘笈心法

心法领悟 595: 得到本地机器的计算机名称。

C#中可以使用 Environment 类的 MachineName 属性来得到本地机器的计算机名称, 代码如下:

```
label1.Text = Environment.MachineName; //得到计算机名称
```

## 实例 596

### 根据 CPU 序列号和磁盘卷标制作软件注册机

光盘位置: 光盘\MR\21\596

高级

趣味指数: ★★★★★

## 实例说明

本实例根据计算机的 CPU 序列号和硬盘卷标来制作一个简单的软件注册机。运行本实例, 首先在窗体的文本框中显示机器码, 单击“生成注册码”按钮, 根据生成的机器码自动生成 24 位注册码。实例运行效果如图 21.4 所示。



图 21.4 根据 CPU 序列号和磁盘卷标制作软件注册机

## 关键技术

本实例实现时, 主要使用 ManagementClass 类的 GetInstances 方法、ManagementObjectCollection 类和 ManagementObject 类获取 CPU 序列号和磁盘卷标, 并将获得的 CPU 序列号和磁盘卷标进行一定的运算, 从而生成机器码和注册码。

 **说明:** 关于 ManagementClass 类的 GetInstances 方法、ManagementObjectCollection 类和 ManagementObject 类的详细讲解, 请参见实例 595 中的关键技术。

## 设计过程

(1) 打开 Visual Studio 2008 开发环境, 新建一个 Windows 窗体应用程序, 并将其命名为 SoftReg。

(2) 更改默认窗体 Form1 的 Name 属性为 Frm\_Main, 在该窗体中添加两个 TextBox 控件, 分别用来显示机器码和注册码; 添加两个 Button 控件, 分别用来执行生成注册码和退出操作。

(3) 程序主要代码如下。

窗体加载时, 首先在文本框中显示机器码, 代码如下:

```
private void Frm1_Load(object sender, EventArgs e)
{
    textBox1.Text = softreg.getMNum(); //生成机器码
}

```

上面的代码中用到了 getMNum 方法, 该方法为自定义的、返回值类型为 string 的方法, 主要用来根据 CPU 序列号和磁盘卷标生成机器码, 代码如下:

```
public string getMNum() //生成机器码
{
    string strNum = getCpu() + GetDiskVolumeSerialNumber(); //获得 24 位 CPU 和硬盘序列号
    string strMNum = strNum.Substring(0,24); //从生成的字符串中取出前 24 个字符作为机器码
}

```

```

    return strMNum;
}

```

上面的代码中用到了 `getCpu` 和 `GetDiskVolumeSerialNumber` 两个自定义方法, 这两个方法分别用来获取 CPU 序列号和磁盘卷标, 代码如下:

```

public string getCpu() //获得 CPU 的序列号
{
    string strCpu = null; //记录获得的 CPU 序列号
    ManagementClass myCpu = new ManagementClass("win32_Processor"); //创建 WMI 查询对象
    ManagementObjectCollection myCpuConnection = myCpu.GetInstances(); //获得查询结果
    foreach (ManagementObject myObject in myCpuConnection) //遍历查询结果
    {
        strCpu = myObject.Properties["Processorid"].Value.ToString(); //获得遍历的 CPU 序列号
        break;
    }
    return strCpu; //返回获得的 CPU 序列号
}
public string GetDiskVolumeSerialNumber() //取得设备硬盘的卷标号
{
    ManagementClass mc = new ManagementClass("Win32_NetworkAdapterConfiguration"); //创建 WMI 查询对象
    ManagementObject disk = new ManagementObject("win32_logicaldisk.deviceid=\"d:\"); //查询磁盘信息
    disk.Get(); //获得磁盘信息
    return disk.GetPropertyValue("VolumeSerialNumber").ToString(); //返回磁盘卷标
}

```

单击“生成注册码”按钮, 在“注册码”文本框中显示注册码, 代码如下:

```

private void button1_Click(object sender, EventArgs e)
{
    textBox2.Text = softreg.getRNum(textBox1.Text); //根据机器码生成注册码
}

```

上面的代码中用到了 `getRNum` 方法, 该方法为自定义的、返回值类型为 `string` 的方法, 主要用来根据指定的机器码生成注册码。`getRNum` 方法的实现代码如下:

```

public string getRNum(string str)
{
    setIntCode(); //初始化 127 位数组
    for (int i = 1; i < Charcode.Length; i++) //把机器码存入数组中
    {
        Charcode[i] = Convert.ToChar(str.Substring(i - 1, 1));
    }
    for (int j = 1; j < intNumber.Length; j++) //把字符的 ASCII 码值存入一个整数数组中
    {
        intNumber[j] = intCode[Convert.ToInt32(Charcode[j])] + Convert.ToInt32(Charcode[j]);
    }
    string strAsciiName = ""; //用于存储注册码
    for (int j = 1; j < intNumber.Length; j++)
    {
        if (intNumber[j] >= 48 && intNumber[j] <= 57) //判断字符 ASCII 码值是否在 0~9 之间
        {
            strAsciiName += Convert.ToChar(intNumber[j]).ToString();
        }
        else if (intNumber[j] >= 65 && intNumber[j] <= 90) //判断字符 ASCII 码值是否在 A~Z 之间
        {
            strAsciiName += Convert.ToChar(intNumber[j]).ToString();
        }
        else if (intNumber[j] >= 97 && intNumber[j] <= 122) //判断字符 ASCII 码值是否在 a~z 之间
        {
            strAsciiName += Convert.ToChar(intNumber[j]).ToString();
        }
        else //判断字符 ASCII 值不在以上范围内
        {
            if (intNumber[j] > 122) //判断字符 ASCII 值是否大于 z
            {
                strAsciiName += Convert.ToChar(intNumber[j] - 10).ToString();
            }
            else
            {
                strAsciiName += Convert.ToChar(intNumber[j] - 9).ToString();
            }
        }
    }
}

```

```

    }
}
return strAsciiName; //返回生成的机器码
}

```

## 秘笈心法

心法领悟 596：如何获得字母的 ASCII 码？

根据字母获得其 ASCII 码值时，需要使用 System.Text.Encoding 类中 ASCII 码编码方式的 GetBytes 方法对字母进行编码。获得字母 ASCII 码值的代码如下：

```

byte[] array = new byte[1];
array = System.Text.Encoding.ASCII.GetBytes(textBox1.Text.Trim());
int asciiCode = (short)(array[0]);
textBox2.Text = Convert.ToString(asciiCode);

```

## 21.2 软件的加密

### 实例 597

### 制作一个 EXE 文件加密器

光盘位置：光盘\MR\21\597

高级

趣味指数：★★★★★

### 实例说明

在一些商业软件中，为了防止盗版，经常要对软件进行加密。加密的方式多种多样，既可以通过注册表加密，也可以通过 EXE 文件加密，还可以通过类似于加密狗的硬件设备进行加密。本实例制作了一个 EXE 文件加密程序，该程序可以对 EXE 文件本身进行加密，还可以限制 EXE 文件的使用期限。实例运行效果如图 21.5～图 21.7 所示。

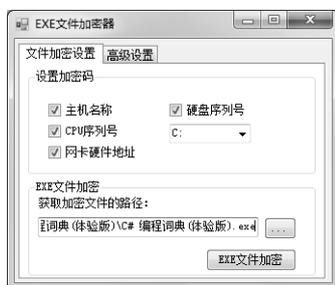


图 21.5 使用硬件信息对 EXE 文件进行加密



图 21.6 设置 EXE 文件的使用期限及次数



图 21.7 对加密的 EXE 文件进行解密

### 关键技术

本实例实现时，主要是使用 FileStream 类的 EXE 文件进行读写，下面对其进行详细讲解。

FileStream 类表示在磁盘或网络路径上指向文件流。当类提供向文件读写字节的方法时，经常使用 StreamReader 或 StreamWriter 执行这些功能，这是因为 FileStream 类操作字节和字节数组，而 Stream 类操作字符数据。字符数据易于使用，但是有些操作如随机文件访问，就必须由 FileStream 对象执行。

使用 `FileStream` 类读取数据不像使用 `StreamReader` 和 `StreamWriter` 类读取数据那么容易，这是因为 `FileStream` 类只能处理原始字节（raw byey），这使得 `FileStream` 类可以用于任何数据文件，而不仅仅是文本文件，通过读取字节数据就可以读取类似图像和声音的文件。

#### （1）使用 `FileStream` 类读取数据

读取数据是用 `FileStream` 对象中的 `Read` 方法实现的，该方法主要用来从流中读取字节块并将该数据写入指定的缓冲区中，其语法格式如下：

```
Public int Read(byte[] array,int offset, int count)
```

参数说明

- ❶ **array**: 被传输进来的字节数组，用以接收 `FileStream` 对象中的数据。
- ❷ **offset**: 字节数组中开始写入数据的位置，通常是 0，表示从数组的开始位置写入数据。
- ❸ **count**: 规定从文件中读出多少字节。

#### （2）使用 `FileStream` 类写入数据

写入数据的流是先获取字节数组，再把字节数据转换为字符数组，然后把这个字符数组用 `FileStream` 对象的 `Write` 方法写入文件中，在写入的过程中，可以确定在文件的什么位置写入、写多少字符等。

`Write` 方法是从缓冲区读取数据（字节块），并将其写入到流中，其语法格式如下：

```
public override void Write(byte[] array, int offset, int count)
```

参数说明

- ❶ **array**: 包含要写入该流的数据的缓冲区。
- ❷ **offset**: `array` 中的从零开始的字节偏移量，从此处开始将字节复制到当前流。
- ❸ **count**: 要写入当前流的最大字节数。

## 设计过程

（1）打开 Visual Studio 2008 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 `FormalityEncryet`。

（2）更改默认窗体 `Form1` 的 `Name` 属性为 `Frm_Main`，在该窗体中添加一个 `TabControl` 控件，用来作为加密设置面板；添加 4 个 `CheckBox` 控件，分别用来选择主机名称、硬盘序列号、CPU 序列号和网卡硬件地址；添加一个 `ComboBox` 控件，用来选择磁盘；添加一个 `TextBox` 控件，用来显示选择的 EXE 文件路径；添加 4 个 `RadioButton` 控件，分别用来设置截止日期、月份个数、天数和运行次数；添加一个 `OpenFileDialog` 控件，用来显示“打开”对话框；添加两个 `Button` 控件，分别用来执行选择 EXE 文件和加密 EXE 文件操作。

（3）程序主要代码如下：

```
/// <summary>
/// 将密码写入 EXE 文件中
/// </summary>
/// <param StrDir="string">EXE 文件的路径</param>
/// <param Prass="string">加密数据</param>
public void WriteEXE(string StrDir, string Prass)
{
    byte[] byData = new byte[100]; //建立一个 FileStream 要用的字节组
    char[] charData = new char[100]; //建立一个字符组
    try
    {
        Prass = Prass.Trim();
        FileStream aFile = new FileStream(StrDir, FileMode.Open); //创建一个 FileStream 对象，用来操作 data.txt 文件
        charData = Prass.ToCharArray(); //将字符串内的字符复制到字符组中
        aFile.Seek(0, SeekOrigin.End); //将指针移到文件尾
        Encoder el = Encoding.UTF8.GetEncoder(); //解码器
        el.GetBytes(charData, 0, charData.Length, byData, 0, true); //将字符数组存入到字节数组中
        aFile.Write(byData, 0, byData.Length); //将字节写入到文件中
        aFile.Dispose();
    }
    catch
    {
        MessageBox.Show("EXE 文件加密失败。");
    }
}
```

```

    }
}
/// <summary>
/// 生成 TXT 文件
/// </summary>
/// <param Prass="string">加密数据</param>
public void CreateTXT(string Prass)
{
    FileStream aFile;
    string TemDir = PFileDir.Substring(0, PFileDir.LastIndexOf("\\")); //获取当前可执行文件的路径 (不包含文件名)
    TemDir = TemDir + "\\" + PFileN + ".TXT"; //在可执行文件的路径下添加一个同名的 TXT 文件
    byte[] byData = new byte[100]; //建立一个 FileStream 要用的字节组
    char[] charData = new char[100]; //建立一个字符串组
    try
    {
        aFile = new FileStream(TemDir, FileMode.CreateNew); //创建一个 FileStream 对象, 用来操作 data.txt 文件
    }
    catch
    {
        aFile = new FileStream(TemDir, FileMode.Truncate);
    }
    try
    {
        Prass = "密码: " + Prass.Trim();
        charData = Prass.ToCharArray(); //将字符串内的字符复制到字符串中
        aFile.Seek(0, SeekOrigin.Begin); //将指针移到文件首
        Encoder el = Encoding.UTF8.GetEncoder(); //解码器
        el.GetBytes(charData, 0, charData.Length, byData, 0, true); //将密码转换成字节
        aFile.Write(byData, 0, byData.Length); //将密码写入文本中
        aFile.Dispose();
    }
    catch
    {
        MessageBox.Show("TXT 文件生成失败。");
    }
}
}

```

(4) 打开 Visual Studio 2008 开发环境, 新建一个 Windows 窗体应用程序, 并将其命名为 DispelFormality。

(5) 更改默认窗体 Form1 的 Name 属性为 Frm\_Main。

(6) 在 DispelFormality 项目中添加一个新的 Windows 窗体, 并将其命名为 Frm\_Dispel, 该窗体用来对加密的 EXE 文件进行解密。在 Frm\_Dispel 窗体中添加一个 TextBox 控件, 用来输入解码; 添加一个 CheckBox 控件, 用来控制下次登录是否显示; 添加两个 Button 控件, 分别用来执行对 EXE 文件进行解码和退出应用程序操作。

(7) 程序主要代码如下:

```

private void button_OK_Click(object sender, EventArgs e)
{
    string temStr = "";
    string TPrass = "";
    string PPrass = "";
    string FDir = "";
    string Fshow = "";
    string Str_Altitude = "";
    if (textBox_Dispel.Text.Length == 0)
    {
        MessageBox.Show("请输入解码。");
        return;
    }
    temStr = textBox_Dispel.Text;
    PPrass = ReadEXEFile(); //获取尾部信息
    PPrass = ReadAltitude(PPrass); //分离密码与高级信息
    TPrass = textBox_Dispel.Text.Trim(); //记录输入的密码
    textBox_Dispel.Text = TPrass;
    if (PPrass == textBox_Dispel.Text) //判断密码是否正确
    {
        if (checkBox_Show.Checked == true) //下次运行是否不显示该窗体
        {

```

```

        Fshow = "T";
    }
    else
    {
        Fshow = "F";
    }
    FDir = Application.ExecutablePath; //添加的注册码路径为 HKEY_CURRENT_USER-Software-LB
    FDir = FDir.Substring(FDir.LastIndexOf("\") + 1, FDir.Length - FDir.LastIndexOf("\") - 1);
    RegistryKey retkey = Microsoft.Win32.Registry.CurrentUser.OpenSubKey("software", true).CreateSubKey("LB").CreateSubKey(FDir).CreateSubKey
("Altitude"); //在注册表中添加 Altitude 文件夹
    enrolValse = ""; //记录添加注册表的值
    NewDate = ""; //记录当前的时间值
    TemporarilyDate = ""; //临时记录时间
    foreach (string sVName in retkey.GetValueNames()) //获取注册表中 Altitude 文件夹中的文件名
    {
        if (sVName == "UserName") //如果是 UserName 文件
        {
            enrolValse = retkey.GetValue(sVName).ToString(); //获取文件中的信息
        }
        if (sVName == "DateCounter") //如果是 DateCounter 文件
        {
            NewDate = retkey.GetValue(sVName).ToString(); //获取文件中的信息
        }
        if (sVName == "DateMonth") //如果是 DateMonth 文件
        {
            TemporarilyDate = retkey.GetValue(sVName).ToString(); //获取文件中的信息
        }
    }
    if (HighSgin == "C") //如果是按运行次数限制 EXE 文件
        HighValue = Convert.ToString(Convert.ToInt32(HighValue) - 1); //将总次数减 1
    Str_Altitude = HighValue + HighSgin + Fshow; //获取修改后的信息
    if (enrolValse == "" || NewDate == "" || TemporarilyDate == "") //如果有一个值为空
    {
        retkey.SetValue("UserName", Str_Altitude.Trim()); //修改注册表中的信息
        retkey.SetValue("DateCounter", System.DateTime.Now.ToString());
        retkey.SetValue("DateMonth", System.DateTime.Now.ToShortDateString());
    }
    else
    {
        ReadRegistered(enrolValse); //读取要写入注册表中的信息
        AmendEnrol(FDir); //修改注册表中的信息
    }
    this.DialogResult = DialogResult.OK; //使当前窗体的返回值为 OK
    this.Close(); //关闭当前窗体
}
else
    textBox_Dispel.Text = temStr;
}
}

```

 **说明：**上面的代码中用到了 ReadEXEFile、ReadAltitude、ReadRegistered 和 AmendEnrol 4 个自定义方法，其中，ReadEXEFile 方法用来获取当前可执行文件的最后 100 个字节，并将其转换成字符串；ReadAltitude 方法用来将在 EXE 文件尾部获取的字符串按照一定的标识将密码、高级设置的值分离出来；ReadRegistered 方法用来在 EXE 文件的尾部信息中读取高级设置的信息，并按照标识记录相应的信息；AmendEnrol 方法用来根据获取的高级设置的信息，修改注册表中指定的文件信息，如果没有指定的文件，则在注册表中创建文件后，再将信息写入相应的文件中。这里由于篇幅限制，不对这 4 个自定义方法进行详细讲解，它们的详细代码请参见本书附带光盘中的源代码。

## 秘笈心法

心法领悟 597：如何根据 ASCII 码获得字母？

开发人员可以使用 System.Text.Encoding 类中 ASCII 码编码方式的 GetString 方法对给出的 ASCII 码值进行编码，以获得其对应的字母。根据 ASCII 码值获得对应字母的代码如下：

```
byte[] array = new byte[1];
array[0] = (byte)(Convert.ToInt32(textBox1.Text.Trim()));
textBox2.Text = Convert.ToString(System.Text.Encoding.ASCII.GetString(array));
```

## 实例 598

## 限制软件的使用次数

光盘位置: 光盘\MR\21\598

高级

趣味指数: ★★★★★

## 实例说明

为了使软件能被更广泛地推广, 开发商希望能有更多的用户使用软件, 但他们又不想让用户长时间免费使用未经授权的软件, 这时就可以推出试用版软件, 限制用户的使用次数, 如果用户感觉使用方便, 可以花钱获取注册码, 以获取其正式版软件。本实例使用 C# 实现了限制软件使用次数的功能, 运行本实例, 如果程序未注册, 则提示用户已经使用过几次, 如图 21.8 所示, 然后进入程序主窗体, 单击主窗体中的“注册”按钮, 弹出如图 21.9 所示的软件注册窗体, 该窗体中自动获取机器码, 用户输入正确的注册码之后, 单击“注册”按钮, 即可成功注册程序, 注册之后的程序将不再提示软件试用次数。



图 21.8 限制软件的使用次数



图 21.9 软件注册

## 关键技术

本实例在实现限制软件的使用次数功能时, 首先需要判断软件是否已经注册, 如果已经注册, 则用户可以任意使用软件。如果软件未注册, 则判断软件是否初次使用, 如果是初次使用, 则在系统注册表中新建一个子项, 用来存储软件的使用次数, 并且设置初始值为 1; 如果不是初次使用, 则从存储软件使用次数的注册表项中获取已经使用的次数, 然后将获取的使用次数加 1, 作为新的软件使用次数, 存储到注册表中。具体实现时, 获取软件使用次数时用到了 Registry 类的 GetValue 方法, 向注册表中写入软件使用次数时用到了 Registry 类的 SetValue 方法。另外, 在对软件进行注册时, 需要根据硬盘序列号和 CPU 序列号生成机器码和注册码, 此时用到了 WMI 管理对象中的 ManagementClass 类、ManagementObject 类和 ManagementObjectCollection 类。下面对本实例中用到的关键技术进行详细讲解。

## (1) Registry 类的 GetValue 方法

Registry 类位于 Microsoft.Win32 命名空间下, 主要用来提供表示 Windows 注册表中的根项的 RegistryKey 对象, 并提供访问项/值对的静态方法, 其 GetValue 方法用来检索与指定的注册表项中的指定名称关联的值, 如果在指定的项中未找到该名称, 则返回提供的默认值; 如果指定的项不存在, 则返回 NULL。GetValue 方法的语法格式如下:

```
public static Object GetValue(string keyName, string valueName, Object defaultValue)
```

GetValue 方法中的参数及说明如表 21.3 所示。

表 21.3 GetValue 方法中的参数及说明

参 数	说 明
keyName	以有效注册表根 (如 HKEY_CURRENT_USER) 开头键的完整注册表路径
valueName	名称/值对的名称
defaultValue	当 name 不存在时返回的值
返回值	如果由 keyName 指定的子项不存在, 则返回 NULL; 否则, 返回与 valueName 关联的值; 或者, 如果未找到 valueName, 则返回 defaultValue

## (2) Registry 类的 SetValue 方法

Registry 类的 SetValue 方法用来设置注册表项中的名称/值对的值，该方法为可重载方法，它有两种重载形式，第一种重载形式的语法格式如下：

```
public static void SetValue(string keyName,string valueName,Object value)
```

参数说明

- ❶ keyName: 以有效注册表根（如 HKEY\_CURRENT\_USER）开头键的完整注册表路径。
- ❷ valueName: 名称/值对的名称。
- ❸ value: 要存储的值。

第二种重载形式的语法格式如下：

```
public static void SetValue(string keyName,string valueName,Object value,RegistryValueKind valueKind)
```

SetValue 方法中的参数及说明如表 21.4 所示。

表 21.4 SetValue 方法中的参数及说明

参 数	说 明
keyName	以有效注册表根（如 HKEY_CURRENT_USER）开头键的完整注册表路径
valueName	名称/值对的名称
value	要存储的值
valueKind	存储数据时使用的注册表数据类型

 说明：关于 ManagementClass 类、ManagementObject 类和 ManagementObjectCollection 类的详细讲解，请参见实例 595 中的关键技术。

## 设计过程

(1) 打开 Visual Studio 2008 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 LimitSoftUseTimes。

(2) 更改默认窗体 Form1 的 Name 属性为 Frm\_Main，在该窗体中添加一个 Button 控件，用来调用“软件注册”窗体。

(3) 在 LimitSoftUseTimes 项目中添加一个 Windows 窗体，并将其命名为 Frm\_Register，用来实现软件注册功能，在该窗体中添加两个 TextBox 控件，分别用来显示机器码和输入注册码；添加两个 Button 控件，分别用来执行软件注册和关闭窗体操作。

(4) 程序主要代码如下。

Frm\_Main 窗体加载时，首先判断程序是否注册，如果已经注册，则将主窗体 Text 属性设置为“限制软件的使用次数（已注册）”，否则，将主窗体 Text 属性设置为“限制软件的使用次数（未注册）”，并且提示软件为试用版和已经使用的次数，同时将注册表中记录的软件使用次数加 1。Frm\_Main 窗体的 Load 事件代码如下：

```
private void frmMain_Load(object sender, EventArgs e)
{
    //打开注册表项
    RegistryKey retkey = Microsoft.Win32.Registry.LocalMachine.OpenSubKey("software", true).CreateSubKey("mrwxk").CreateSubKey("mrwxk.ini");
    foreach (string strRNum in retkey.GetSubKeyNames()) //判断是否注册
    {
        if (strRNum == softreg.getRNum()) //判断注册码是否相同
        {
            this.Text = "限制软件的使用次数（已注册）";
            button1.Enabled = false;
            return;
        }
    }
    this.Text = "限制软件的使用次数（未注册）";
    button1.Enabled = true;
    MessageBox.Show("您现在使用的是试用版，该软件可以免费试用 30 次！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    Int32 tLong;
    try
    {
```

```

//获取软件已经使用的次数
tLong = (Int32)Registry.GetValue("HKEY_LOCAL_MACHINE\\SOFTWARE\\tryTimes", "UseTimes", 0);
MessageBox.Show("感谢您已使用了" + tLong + "次", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
catch
{
//首次使用软件
Registry.SetValue("HKEY_LOCAL_MACHINE\\SOFTWARE\\tryTimes", "UseTimes", 0, RegistryValueKind.DWord);
MessageBox.Show("欢迎新用户使用本软件", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
//获取软件已经使用的次数
tLong = (Int32)Registry.GetValue("HKEY_LOCAL_MACHINE\\SOFTWARE\\tryTimes", "UseTimes", 0);
if (tLong < 30)
{
int Times = tLong + 1; //计算软件本次是第几次使用
Registry.SetValue("HKEY_LOCAL_MACHINE\\SOFTWARE\\tryTimes", "UseTimes", Times); //将软件使用次数写入注册表
}
else
{
MessageBox.Show("试用次数已到", "警告", MessageBoxButtons.OK, MessageBoxIcon.Warning);
Application.Exit(); //退出应用程序
}
}
}

```

 说明：关于软件注册功能的实现，其实现原理与实例 596 中根据 CPU 序列号和磁盘卷标制作软件注册机的实现原理是类似的，请参见实例 596。

## 秘笈心法

心法领悟 598：如何获得汉字的区位码？

根据汉字获得其对应区位码时，需要使用 System.Text.Encoding 类中 Default 编码方式的 GetBytes 方法对给出的汉字进行编码。获得汉字区位码的关键代码如下：

```

byte[] array = new byte[2];
array = System.Text.Encoding.Default.GetBytes("" + textBox1.Text.Trim() + "");
int front = (short)(array[0]-'\0');
int back = (short)(array[1]-'\0');
textBox2.Text = Convert.ToString(front-160) + Convert.ToString(back-160);

```

## 实例 599

### 使用强名称标识软件

光盘位置：光盘\MR\21\599

高级

趣味指数：★★★★★

## 实例说明

强名称是由程序集的标识加上公钥和数字签名组成的，其中，程序集的标识包括简单文本名称、版本号和区域性信息，它使用对应的私钥从程序集文件中生成（程序集文件包含程序集清单，其中包含组成程序集的所有文件的名称及其哈希代码）。本实例实现使用 .NET 提供的强名称验证机制标识软件功能。

## 关键技术

本实例使用强名称标识软件时，首先需要知道如何创建密钥对，然后需要知道如何使用创建的密钥对标识软件。下面对本实例用到的关键技术进行详细讲解。

### (1) 如何创建密钥对

创建密钥对时需要用到 .NET 提供的 sn -k 命令，该命令用来提供用于密钥管理、签名生成和签名验证的选项。例如，本实例中生成 key.snk 密钥对时需要在“Visual Studio 2008 命令提示”中输入如下命令：

```
sn -k C:\key.snk
```

### (2) 如何使用创建的密钥对标识软件

使用密钥对标识软件时需要用到 AssemblyKeyFile 属性类，该类位于 System.Reflection 命名空间下，其构造

函数的语法格式如下：

```
public AssemblyKeyFileAttribute(string keyFile)
```

参数说明

keyFile: 包含密钥对的文件的名称。

## 设计过程

(1) 在“开始”菜单中打开“Visual Studio 2008 命令提示”，其中输入 `sn -k C:\key.snk`，如图 21.10 所示，按回车键，在 C 盘生成一个 `key.snk` 密钥文件。



图 21.10 Visual Studio 2008 命令提示

(2) 新建一个 Windows 应用程序，并将其命名为 `SNKSoft`，在其 `AssemblyInfo.cs` 程序集文件中使用 `AssemblyKeyFile` 属性类的构造函数指定生成的密钥文件，保存运行即可。代码如下：

```
[assembly:AssemblyKeyFile("key.snk")]
```

**注意：** `key.snk` 密钥文件应该存放在 `SNKSoft` 项目的根目录下。

## 秘笈心法

心法领悟 599：如何将数字转换为字符串？

将数字转换为字符串时，需要使用 `ToString` 方法，或者使用 `Convert.ToString` 方法。例如，下面的代码将数字 123 转换为字符串显示在 `TextBox` 文本框中：

```
int i=123;
textBox2.Text = Convert.ToString(i);
```

## 实例 600

### 软件加壳常用工具及使用

光盘位置：光盘\MR\21\600

高级

趣味指数：★★★★☆

## 实例说明

对于刚刚接触加密解密开发人员来说，软件加壳是一门非常高深的学问，但它又是现代社会非常流行的一门学问，为了保障软件的安全性，软件加壳显得非常重要，本实例将介绍一种常用的软件加壳工具及其使用。

## 关键技术

要使用软件加壳工具，首先应该了解什么是加壳，下面对加壳进行详细讲解。

加壳的全称是可执行程序资源压缩，它是一种保护文件的常用手段，加壳过的程序可以直接运行，但是不能查看源代码，要经过脱壳才可以查看源代码。

加“壳”其实是利用特殊的算法，对 `EXE` 或 `DLL` 文件中的资源进行压缩，它类似 `WINZIP` 的效果，只不过这个压缩之后的文件，可以独立运行，解压过程完全隐蔽，都在内存中完成。加“壳”虽然增加了 CPU 附带，但是也减少了硬盘读写时间，实际应用时，加“壳”以后的程序运行速度更快。一般软件都要加“壳”，因为这样不但可以保护软件不被破解、修改，还可以增加运行时的启动速度。

 说明: RAR 和 ZIP 都是压缩软件而不是加“壳”工具,它们解压时需要进行磁盘读写,而“壳”的解压缩是直接在内存中进行的。

## 设计过程

本实例主要对 UPX 工具进行详细讲解。

UPX 是一款先进的可执行程序文件压缩器,经 UPX 压缩过的可执行文件体积缩小 50%~70%,这样减少了磁盘占用空间、网络上传下载的时间和其他分布以及存储费用;另外,通过 UPX 压缩过的程序和程序库完全没有功能损失,和压缩之前一样可正常运行。

UPX 工具的使用步骤如下:

(1) 打开 UPX 工具,其主界面如图 21.11 所示。

(2) 单击“目录”按钮,选择要加壳的文件所在的主目录;单击“文件”按钮,选择要加壳的文件,选择完之后,会在 UPX 主界面的右下方出现一个“压缩”按钮,如图 21.12 所示。



图 21.11 UPX 主界面



图 21.12 选择加壳文件

(3) 在 UPX 主界面中选择文件的加壳方式,单击“压缩”按钮,弹出如图 21.13 所示的命令对话框,显示压缩了一个文件。



图 21.13 压缩成功

## 秘笈心法

心法领悟 600: 如何将数字转换为日期格式?

将数字转换为日期格式时,可以首先声明 DateTime 类的一个对象,并将要转换为日期格式的数字赋值给该对象,然后调用该对象的 ToLongDateString 方法或 ToShortDateString 方法进行格式转换。将数字转换为日期格式的代码如下:

```
DateTime mydatetime = new DateTime(Convert.ToInt32(comboBox1.Text), Convert.ToInt32(comboBox2.Text), Convert.ToInt32(comboBox3.Text));
MessageBox.Show("日期为: " + mydatetime.ToLongDateString(), "信息", MessageBoxButtons.OK, MessageBoxIcon.Information);
```