软件加密技术从入门到精通

史子荣　编著

清 华 大 学 出 版 社
北　京

FOREWORD

前言

越来越多的软件可以轻易地被破解掉，破解后的软件注册机、注册码、破解补丁在网上被公布出来，使软件作者蒙受了巨大的经济损失。造成这种情况的原因主要是软件开发者对软件加密的认识不足或是对软件加密不知道从何入手。他们要么太过于相信成熟的加密算法或是过于相信加密工具，要么没有分清注册码的明码比较和非明码比较，要么把加密算法的使用方法弄错了等等，如此做出来的软件保护方案在解密者手中简直就是不堪一击。

关于本书

本书将从最基本的基础知识开始讲起，逐步深入地介绍如何对软件进行加密，以及软件加密过程中所用到的各种知识，引导软件开发者设计出难以破解的软件保护方案。

本书各章的具体内容安排如下：

第 1 章　基础知识

讲述软件加密过程中用到的各种基础知识，包括注册表的读写操作、INI 文件和自定义文件的读写操作、DLL 文件和 BPL 控件的设计、结构化异常处理、防止应用程序的多个事例运行，以及在 Delphi 嵌入汇编进行编程。

第 2 章　加密算法

先介绍 HASH 算法（CRC32、MD5、SHA 算法）、对称算法（DES、双重 DES、三重 DES、Blowfish、AES 算法）、公开密钥算法（RSA 算法）、其他算法（BASE64 算法）的基本原理，然后用实例演示各种加密算法的实现。

第 3 章　软件试用期

以原理和实例来讲述软件加密过程的试用期设置技术，这些设置技术包括软件试用次数、软件试用天数、软件最后试用日期、限制软件启动后的执行时间、NAG 窗口提示等。

第 4 章　反跟踪技术

先介绍软件破解过程需要用到的各种工具，包括：SoftICE、OllyDbg、FileMon、RegMon、IDA Pro、W32DASM、DeDe，然后介绍检测这些工具的技术以及防脱壳技术、信息隐藏技术等。

第 5 章　注册认证和注册机

介绍软件的注册认证部分，包括如何选用加密算法和校验方式、硬件系列号保护方式、用户名保护方式、硬件系列号与用户名保护方式、随机注册码模式、KeyFile 保护方式、用 DLL 实现注册认证、控件的注册认证方式、完整的软件保护方案实现，同时以实例演示各种技术。

第 6 章　网络验证

以 Web 服务器验证和本地服务器验证来讲述基于网络的注册验证技术，用网络验证技术实现在线升级的验证。

第 7 章　PE 文件知识

讲述 Win32 平台下 PE 文件结构的知识，让读者能洞悉操作系统的秘密。

示例代码

本书是以实际应用作为写作依据的，在书中会出现大量的示例代码，限于篇幅，并没有把所有的代码放到书中，书里只放了关键的部分代码，书中省略代码处以“//…”标记，完整的示例代码可参看随书光盘。

本书全部的示例代码都是用 Delphi 7.0 进行编写的，书中的十六进制是以 Delphi 方式的$表示而不是 0x 表示，使用其他开发语言的读者需要区分一下。

致谢

首先要感谢母校的钱波、藤春芳、吴文虹，另外要感谢在本书写作过程中帮助与支持我的王国彦，最后感谢给予我支持和帮助的所有朋友。

史子荣
2007 年 2 月于昆明
http://www.pefine.com

此为试读，需要完整PDF请访问：www.ertongbook.com

第1章 基础知识

在对软件进行加密之前，首先要了解一些在加密过程中经常用到的基础知识：如何将加密信息保存到注册表？如何保存加密信息到文件？在Delphi 中如何封装 DLL 和 BPL？如何防止应用程序的多个实例运行？如何在 Delphi 中嵌入汇编进行编程？本章将详细讲述这些知识。

1.1 注册表知识

注册表是 Windows 的核心数据库，保存着各种硬件、软件的配置信息和参数。Windows 系统及 Windows 系统下的应用程序都可以对注册表进行操作，包括对注册表数据的读取、写入、删除、修改。

1.1.1 注册表结构

注册表按树状进行分类，逐层地往下分，形成根、根主键、主键、子键、值项的分层结构，如同 Windows 的资源管理器一样。注册表的根主键不能被删除，也不可以新建根主键，但可以对根主键下的各项进行删除、修改、增加操作。在“开始”菜单中的“运行”对话框里面输入“regedit”，打开注册表，就可以看到注册表的结构，注册表结构如图 1-1 所示。

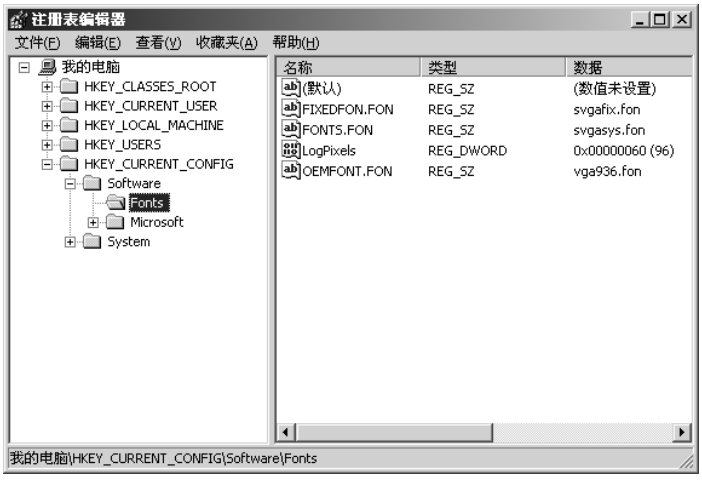

图 1-1 Windows 注册表

各个根主键的含义如下：

1. HKEY_CLASSES_ROOT（简称 HKCR）

HKCR 根主键为系统的每个资源进行归类，这些资源是文件扩展名和 COM 组件的注册信息。

2. HKEY_CURRENT_USER（简称 HKCU）

HKCU 根主键包括了当前用户的一些具体信息，这些信息是当前登录用户相关软件的配置、信息、参数。

3. HKEY_LOCAL_MACHINE（简称 HKLM）

HKLM 根主键里存放着本机的软件、硬件、系统和安全性等信息。

4．HKEY_USERS（简称 HKU）

HKU 根主键里包括了一些针对每个具体用户的有关信息，这些信息是动态加载的用户配置文件和默认配置文件的信息，同时还包含了出现在 HKEY_CURRENT_USER 根主键中的信息。

5．HKEY_CURRENT_CONFIG（简称 HKCC）

HKCC 根主键里存放着当前除了本地配置之外的一些系统配置。

注册表有它自身的数据类型，用于存储不同类型的数据。注册表有存储字符串数据的类型、存储二进制数据的类型、存储十六进制数据的类型等。注册表数据类型及描述见表 1-1。

表 1-1　注册表数据类型及描述

类型	描述
REG_SZ	以空字符结尾的字符串
REG_BINARY	二进制原始数据类型
REG_DWORD	十六进制信息
REG_MULTI_SZ	以空字符结尾的字符串数组，它将以两个字符串结尾
REG_EXPAND_SZ	以空字符结尾的字符串，其中包括对环境变量的未展开的引用

1.1.2　注册表相关函数

可以直接使用 Delphi 封装好的函数来对注册表进行操作，而不用直接使用 API。在使用注册表操作函数之前，应先引用 Registry 单元，该单元封装了对注册表进行操作的各种 API 函数。

常用函数的定义以及功能如下：

```
function OpenKey(const Key: String; Cancreate: Boolean): Boolean;
```

功能：打开指定的主键。

参数：Key 表示主键，Cancreate 表示如果指定的主键不存在时是否创建，True 为创建，False 为不创建。函数调用成功返回 True。当 Cancreate 使用 True 的时候可以省去用 CreateKey 函数来创建主键。

```
procedure WriteString(const Name, Value: string);
```

功能：把一个字符串值写入到指定的名称中。

参数：Name 表示名称，Value 表示要写入的键值。

```
procedure WriteInteger(const Name: string; Value: Integer);
```

功能：把一个整数值写入到指定的名称中。

参数：Name 表示名称，Value 表示要写入的键值。

```
function ReadString(const Name: string): string;
```

功能：从指定的字符串类型的名称中读取键值。

参数：Name 表示键的名称，返回值为键值。

```
function ReadInteger(const Name: string): Integer;
```

功能：从指定的整数类型的名称中读取键值。
参数：Name 表示键的名称，返回值为键值。


```
procedure CloseKey;
```

功能：关闭打开的注册表键。

上面仅仅介绍了在加密过程需要用到的操作注册表的几个函数，如果需要了解更多关于操作注册表的函数的资料可以查阅 Delphi 的联机帮助文件。

1.1.3 注册表读操作

从注册表中读取信息的步骤如下：
（1）先创建 TRegistry 对象。
（2）用 OpenKey 打开想要操作的主键。
（3）用相关读取函数如 ReadString 读取指定名称的键值。
（4）操作完毕以后，用 CloseKey 关闭打开的主键和使用 Destroy 释放创建的对象。
示例代码如下：

```
uses Registry;

procedure TForm1.Button1Click(Sender: TObject);
var
  Reg:TRegistry;
begin
  Reg:=TRegistry.Create;              //创建 TRegistry 对象
  Reg.RootKey:=HKEY_CURRENT_USER;     //根主键
  if Reg.OpenKey('\Software\Microsoft\Notepad',False) then  //打开指定主键
    Edit1.Text:=Reg.ReadString('lfFaceName');               //读取指定名称的键值
  Reg.CloseKey;                       //关闭打开的键
  Reg.Destroy;                        //释放内存
end;
```

1.1.4 注册表写操作

往注册表中写入信息的步骤如下：
（1）先创建 TRegistry 对象。
（2）用 OpenKey 打开想要操作的主键。
（3）用相关写入函数（如 WriteString）写入指定名称的键值。
（4）操作完毕以后，用 CloseKey 关闭打开的主键并使用 Destroy 释放创建的对象。

此为试读，需要完整PDF请访问：www.ertongbook.com

示例代码如下：

```
uses Registry;

procedure TForm1.Button2Click(Sender: TObject);
var
  Reg:TRegistry;
begin
  Reg:=TRegistry.Create;                  //创建 TRegistry 对象
  Reg.RootKey:=HKEY_CURRENT_USER;         //根主键
  if Reg.OpenKey('\Software\Microsoft\Notepad',False) then      //打开指定主键
    Reg.WriteString('BitEncrypt',Edit2.Text);  //把 Edit2 的内容写入到 BitEncrypt 中
  Reg.CloseKey;          //关闭打开的键
  Reg.Destroy;           //释放内存
end;
```

1.2 文件读写知识

除了把软件加密信息和用户信息保存到注册表中之外，还可以把这些信息保存到文件中。对不同的文件有不同的操作方法，在这里主要介绍对 INI 文件和自定义文件的操作方法。

1.2.1 INI 文件知识

INI 文件在 Windows 3.1 时就存在，它在系统配置及应用程序参数保存与设置方面具有很重要的作用，它同时也可以在软件加密过程中用于保存加密信息和用户信息。INI 文件的格式是文本文件，它的大小限制为 64KB。

1. INI 文件的结构

;注释
[节名]
关键字 1=值
关键字 2=值
…

INI 文件就是按上面给出的结构来进行设置的，它同时可以有多个节名，每个节下面又可以有多个关键字。使用“;”可以注释 INI 文件的内容。关键字后面的值有字符串型、整型、布尔型等数据类型。

2. INI 文件相关函数

同样，在操作 INI 的时候可以不直接使用 API 函数，可以直接使用 Delphi 封装好的函数。在使用 INI 文件操作函数之前需要先引用 IniFiles 单元，它对操作 INI 文件的 API 函数进行了封装。

常用函数的定义及功能如下：

```
function ReadString(const Section, Ident, Default: string): string;
```

功能：读取指定节名下面的关键字的值。

参数：Section 表示节名，Ident 表示关键字，Default 表示默认值，该值为当 INI 文件或是节名、关键字不存在时返回的值，这个值可以根据自己的需要来设定。函数调用成功后，返回值为关键字的值（字符串型）。

```
function ReadInteger(const Section, Ident: string;Default: Longint): Longint;
```

功能：读取指定节名下面的关键字的值。

参数：Section 表示节名，Ident 表示关键字，Default 表示默认值，该值为当 INI 文件不存在或是节名、关键字不存在时返回的值，这个值可以根据自己的需要来设定。函数调用成功后，返回值为关键字的值（长整型）。

```
procedure WriteString(const Section, Ident, Value: string);
```

功能：写入指定的字符串信息到 INI 文件中。

参数：Section 表示节名，Ident 表示关键字，Value 表示要写入的值。写入时如果节名不存在，那么会创建一个以 Section 内容命名的节。

```
procedure WriteInteger(const Section, Ident: string; Value: Longint);
```

功能：写入指定的长整型信息到 INI 文件中。

参数：Section 表示节名，Ident 表示关键字，Value 表示要写入的值。写入时如果节名不存在，那么会创建一个以 Section 内容命名的节。

以上介绍的 4 个函数同样也只是在加密过程中经常用得到，需要了解更多的关于操作 INI 文件的函数的资料可以查阅 Delphi 的联机帮助文件。

3．INI 文件的读操作

从 INI 文件中读取信息的步骤如下：

（1）创建一个 TIniFile 对象。

（2）使用 INI 文件读操作函数如 ReadString 读入信息。

（3）操作完毕以后使用 Destroy 释放创建的对象。

示例代码如下：

```
uses IniFiles;

procedure TForm1.Button1Click(Sender: TObject);
var
  IniFileName:String;
  MyIniFile:TIniFile;
begin
  //Ini 文件名和路径
```

```
  IniFileName:=ExtractFilePath(Application.ExeName)+'MyIni.ini';
  MyIniFile:=TIniFile.Create(IniFileName);//创建 TIniFile 对象
  Edit1.Text:=MyIniFile.ReadString('软件加密','加密内容','错误');//读取信息
  MyIniFile.Destroy;//释放对象
end;
```

4．INI 文件的写操作

往 INI 文件中写入信息的步骤如下：

（1）创建一个 TIniFile 对象。

（2）使用 INI 文件写操作函数如 WriteString 写入信息。

（3）操作完毕以后，使用 Destroy 释放创建的对象。

示例代码如下：

```
uses IniFiles;

procedure TForm1.Button2Click(Sender: TObject);
var
  IniFileName:String;
  MyIniFile:TIniFile;
begin
  //Ini 文件名和路径
  IniFileName:=ExtractFilePath(Application.ExeName)+'MyIni.ini';
  MyIniFile:=TIniFile.Create(IniFileName);//创建 TIniFile 对象
  MyIniFile.WriteString('软件加密','技术内容',Edit2.Text);//写入信息
  MyIniFile.Destroy;//释放对象
end;
```

1.2.2 自定义文件知识

在保存加密信息或者使用 KeyFile 方式对程序进行保护的时候，可以按照自己的需要来定义文件内容格式，这样非常方便、灵活。由于开发人员可以按照自己的需要来定义各种各样的文件内容格式，在这里不可能把所有的方式都列出来，所以只以一种简单的方法来讲解如何设计和使用自定义文件。

文件内容格式定义如下：

```
系列号*用户名*注册码
```

文件格式说明：这种方法采用“*”把信息分割开来，从文件读入信息以后，只要以“*”作为分割点进行分割就可以得到各条信息，写入的时候只要把各条信息用“*”连起来写到文件即可。需要注意的是，需要分割的内容里不能含有与分割符相同的字符，选择分割符的时候，要确认分割符不会与需要分割的内容包含的字符相同。在实际应用中，不仅可以采用字符作为分割符，也可以采用自定义的字符串作为分割符。

读入信息并分割信息的示例代码如下：

```
//分割字符串
function GetMsg(RootStr,ChildStr: string): TStringList;
var
  iTemp: integer;
begin
  result := TStringList.Create;
  iTemp := pos(ChildStr,RootStr);
  while iTemp>0 do begin
    if iTemp>1 then result.Append(copy(RootStr,1,iTemp-1));
    delete(RootStr,1,iTemp+length(ChildStr)-1);
    iTemp := pos(ChildStr,RootStr);
  end;
  if RootStr<>'' then result.Append(RootStr);
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  CustFileName:String;
  MsgStr,TempList:TStrings;
begin
  CustFileName:=ExtractFilePath(Application.ExeName)+'Key.lic';
  if FileExists(CustFilename) then
    begin
      MsgStr:=TStringList.Create;            //创建 TStringList 对象
      TempList:=TStringList.Create;
      MsgStr.LoadFromFile(CustFileName);     //读入信息
      TempList:=GetMsg(MsgStr[0],'*');
      //获取各信息
      Edit1.Text:=Templist[0];
      Edit2.Text:=TempList[1];
      Edit3.Text:=TempList[2];
      //释放 TStringList 对象
      MsgStr.Free;
      TempList.Free;
  end;
end;
```

写入信息示例代码如下：

```
procedure TForm1.Button2Click(Sender: TObject);
var
  CustFileName:String;
  MsgStr:TStrings;
begin
  CustFileName:=ExtractFilePath(Application.ExeName)+'Key.lic';
  MsgStr:=TStringList.Create;                //创建 TStringList 对象
```

此为试读，需要完整PDF请访问：www.ertongbook.com

```
  MsgStr.Add(Edit1.Text+'*'+Edit2.Text+'*'+Edit3.Text);      //增加信息到 MsgStr 中
  MsgStr.SaveToFile(CustFileName);       //把 MsgStr 中的信息保存到指定文件中
  MsgStr.Free;                //释放 TStringList 对象
end;
```

1.3 动态链接库（DLL）设计

动态链接库（Dynamic Link Library，DLL）在 Microsoft 推出第一个版本的 Windows 操作系统时就存在，它一直是操作系统的基础。DLL 也是 PE 文件，但它不会主动执行，需要映射到其他应用程序的地址空间，DLL 的函数或过程才能被调用。一旦 DLL 文件被映射到调用进程的地址空间中，DLL 函数或过程就可以供进程中运行的所有线程使用。

1.3.1 创建 DLL 文件

在 Delphi 开发环境中选择 File–>New–>Other 命令，在打开的对话框中选择 New 页面，然后选择 DLL Wizard 命令，单击 OK 按钮就创建了一个空的 DLL 工程文件。创建的 DLL 新工程文件的内容如下所示：

```
library Project1;

uses
  SysUtils,
  Classes;

{$R *.res}

begin
end.
```

DLL 工程文件以 library 关键字开始，区别于一般的应用程序工程文件。一般的应用程序工程文件是以 program 关键字开始的。library 关键字通知编译器把工程文件编译成 DLL 文件，program 关键字通知编译器把工程文件编译成 EXE 文件。DLL 文件的函数和过程的实现可以放到工程文件中，也可以单独地写在一个单元中。如果是在单元中实现，那么在工程文件的 uses 部分引用单元即可。在需要导出的函数或过程的实现部分后面加上 Stdcall 关键字，再在工程文件的尾部加入 exports 关键字，把需要导出的函数或过程的名称写在该关键字的后面，多个名称之间以“,”分割，结尾的名称后面加上“;”。完整的 DLL 文件的示例代码如下：

```
library DllTest;

uses
  SysUtils,
  Classes;

{$R *.res}
```

```
//计算两数之和使用 Stdcall 调用约定
function Sum(a,b:Integer):Integer;Stdcall;
begin
  Result:=a+b;
end;

//连接两个字符串使用 Stdcall 调用约定
function LinkStr(Str1,Str2:PChar):PChar;Stdcall;
begin
  Result:=PChar(StrPas(Str1)+StrPas(Str2));
end;

//判断两数大小使用 Stdcall 调用约定
function IfCheck(Int1,Int2:Integer):Boolean;Stdcall;
begin
  if Int1>Int2 then
    Result:=True
  else
    Result:=False;
end;

//导出函数接口
exports
  Sum,LinkStr,IfCheck;

begin
end.
```

以上代码编译以后即可生成一个完整的 DLL 文件。在 DLL 文件的输出部分，也就是 exports 部分，可以使用 3 个标准指示字，即 name、index、resident。

1．name 指示字

该指示字为需要导出的函数或过程名设置一个别名，DLL 文件编译以后导出的就是这个别名。在其他应用程序中要调用该函数或过程，如果使用原来的函数或过程的名称调用将出错，而应该用别名来进行调用。name 指示字的设置格式如下：

```
exports
  Sum name 'MySum';
```

导出的是 Sum 的别名 MySum，调用的时候用 MySum。

2．index 指示字

index 指示字为函数或过程分配一个顺序号，它的范围在 1～32767 之间。如果不使用 index 指示字，则由编译器按顺序分配，使用 index 指示字可以提高调用 DLL 文件中函数或过程的速度。index 指示字的设置格式如下：

```
exports
  Sum index 1;
```

3．resident 指示字

resident 指示字是使当 DLL 载入时指定的导出函数或过程始终保持在内存当中，这样调用该函数或过程时，可以比利用名字扫描 DLL 的入口降低时间的开销。对于经常需要调用的函数或过程，使用 resident 指示字是比较合适的。resident 指示字的设置格式如下：


```
exports
  Sum name 'MySum' resident;
```

1.3.2 隐式调用

隐式调用又称静态调用，是指在应用程序开始执行的时候就把 DLL 导出的所有接口载入到内存中，不管该接口有没有被使用。隐式调用的示例代码如下：

```
implementation

function Sum(a,b:Integer):Integer;stdcall;external 'DllTest.dll';
function LinkStr(Str1,Str2:PChar):PChar;Stdcall;external 'DllTest.dll';
function IfCheck(Int1,Int2:Integer):Boolean;Stdcall;external 'DllTest.dll';

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit3.Text:=IntToStr(Sum(StrToInt(Edit1.Text),StrToInt(Edit2.Text)));
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit6.Text:=StrPas(LinkStr(PChar(Edit4.Text),PChar(Edit5.Text)));
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  if IfCheck(StrToInt(Edit7.Text),StrToInt(Edit8.Text)) then
    Edit9.Text:='数一大于数二'
else
    Edit9.Text:='数一小于数二';
end;
```

隐式调用只要在调用单元的 implementation 部分加上函数或过程的声明语句，然后在函数或过程的声明语句的后面加上 external 指示字，并把需要调用的 DLL 文件名放在 external 指示字的后面。

1.3.3 显式调用

不管 DLL 的导出函数或过程是否被使用，隐式调用都会加载到内存中。如果不用的导出函数或过程很多，会浪费大量的内存去载入这些函数或过程。如果是在使用到某个函数或过程的时候才把需要的函数或过程载入到内存中，这样将会大大节省内存空间，这种方式就是显式调用，又称为动态调用。动态调用是通过 LoadLibrary、FreeLibrary 和 GetProcAddress 这 3 个 API 函数来实现的。示例代码如下：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  hDll: HMODULE;
  Sum:function(a,b:Integer):Integer;stdcall;
begin
  hDll:= LoadLibrary('DllTest.dll');
  if hDll <> 0 then
  begin
    @Sum := GetProcAddress(hDll, 'Sum');
    if @Sum <> nil then
      Edit3.Text:=IntToStr(Sum(StrToInt(Edit1.Text),StrToInt(Edit2.Text)))
  else
    ShowMessage('加载功能模块出错！');
  FreeLibrary(hDll);
  end
  else
    ShowMessage('无法加载 DLL！');
end;
```

显式调用在每次调用函数或过程时都会载入和释放 DLL。如果在整个应用程序执行过程中只调用了一次函数或过程，那么显式调用是很节省内存资源的。如果在整个应用程序执行过程中调用函数或过程非常频繁，那么重复地载入和释放 DLL 会带来额外的负担。所以，在调用 DLL 时，应注意选择合适的载入方式，对于经常需要调用的 DLL，可以采用隐式调用的方式载入，对于很少使用到的 DLL，可以采用显式调用的方式载入。

1.4 BPL 组件设计

包可以把应用程序的一部分放到一个单独的模块中，让其他应用程序共享，实现程序设计模块的分离与共享。包可以称为 Delphi 的 DLL，它与 DLL 一样都是在运行期动态地链接到应用程序中的，唯一的不同点就是包只能使用在 Delphi 开发环境中。

1.4.1 包的基础知识

Delphi 的包有如下 4 种类型：

1．运行期包

运行期包包含了应用程序在运行时所需的代码，如果应用程序使用了运行期包，那么在发布应用程序的时候，需要连同所使用的运行期包一起发布。发布时如果没有连同运行期包一起发布，则应用程序无法正常运行。

2．设计期包

设计期包包含了在 Delphi IDE 中设计应用程序所必需的组件、属性等。设计期包只能用于 Delphi 中，而不用和应用程序一起发布。

3．设计期和运行期包

它既是设计期包又是运行期包，通常用在没有具体的设计因素（组件、属性等）时。使用这种包可以简化应用程序的开发和配置，如果包里面包含设计因素，建议单独使用设计期包或是运行期包。

包文件及含义见表 1-2。

表 1-2　包文件及含义

扩展名	文件类型	描述
.dpk	包的源文件	该文件是调用包编辑器时创建的，可以把它当作项目文件
.dcp	包的符号文件	这是已编译了的包，它包含了包和单元的标识信息以及所需的头信息
.dcu	编译过的单元	包里所包含的单元文件编译后得到的文件，包里每个单元都有一个对应的 dcu 文件
.bpl	包的库文件	这个是运行期或是设计期的包文件，相当于 Windows 中的 DLL 文件

1.4.2　包的设计

包里面包含一个或多个组件。在设计包之前应该先把需要用到的组件设计完成，然后增加到包里。组件的创建有两种方式：手动创建和自动创建。虽然创建过程不一样，但所得的结果都是一样的。下面以手动的方式建立一个非可视化组件为例来进行讲述。

首先在 Delphi 环境中关闭所有的工程，选择 File–>New–>Unit 命令得到一个空单元文件，按如下的方式对单元文件进行扩展。

```
unit MyClass;

interface

uses
  SysUtils, Classes;

type
  TMyClass = class(TComponent)
  private
    FMul:Integer;
  protected
```

```
    { Protected declarations }
  public
    function sum(a,b:Integer):Integer;
    procedure ReadMul(a,b:Integer);

    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    property Mul:Integer read FMul write FMul;
  end;

procedure Register;

implementation

constructor TMyClass.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FMul:=0;
end;

destructor TMyClass.Destroy;
begin
  inherited Destroy;
end;

procedure TMyClass.ReadMul(a,b:Integer);
begin
  FMul:=a*b;
end;

function TMyClass.sum(a,b:Integer):Integer;
begin
  Result:=a+b;
end;

procedure Register;
begin
  RegisterComponents('MyClass', [TMyClass]);
end;

end.
```

上面的代码演示了组件的创建方法，读者可以根据自己的需要进行扩展，详细的设计可参看其他书籍。

接下来创建包文件，在 Delphi 中选择 File–>New–>Other 命令，在打开的对话框中选择 New 页面，然后选择 Package 命令，出现如图 1-2 所示的对话框。

此为试读，需要完整PDF请访问：www.ertongbook.com