计算机软件开发系列丛书

Visual C++ 类库深入剖析

何小路 刘 莹 编著

希望

学苑出版社

Microsoft Press 新近中文版电脑图书

序号	购书编号	书　　名	定价
1	969	软件人员必备工具书:代码大全	66.00 元
2	1214	运行 MS — Quick Basic(DOS 环境下)	46.00 元
3	979	PC 热线—微机常见问题精解	23.00 元
4	990	PC 程序员经典参考手册	58.00 元
5	623	MS — Visual Basic for Windows 初级教程	37.50 元
6	663	MS — Visual Basic for DOS 编程训练	65.00 元
7	1215	Visual Basic for Windows 编程训练	59.50 元
8	743	运行 MS — Access	48.00 元
9	858	MS — Access for Windows 循序渐近教程	54.00 元
10	621	MS — Windows for Workgroups 简明指导	19.40 元
11	622	MS — Windows for Workgroups 用户伴侣	41.00 元
12	664	MS — Windows 3.1 程序设计(含盘)	98.00 元
13	741	MS — Windows 3.1 编程经验集(含盘)	66.00 元
14	670	Microsoft 多媒体技术丛书(3 册)	98.00 元
15	1282	中文版 MS — Windows 3.1 SDKE DDKE 实用手册	99.00 元
16	838	MS — Windows 环境下字型核心技术—— True Type	98.00 元
17	826	MS 教学软件的家长指南	29.00 元
18	1138	MS — Windows 3.1 超级笔驱动图形环境程序员参考手册	49.00 元
19	1145	MS — Mouse 鼠标器程序员参考手册(含盘 2 张)	89.00 元
20	1156	MS — Profit 应用入门	35.00 元
21	997	MS — Mail for Windows 循序渐近教程(含盘)	32.00 元
22	1054	运行 MS — Mail for Windows(含盘)	24.00 元
23	791	看图例学 MS — Word for Windows	28.00 元
24	920	运行 MS — Word 2.0 for Windows	27.00 元
25	1056	MS — Word for Windows 基础教程	25.00 元
26	1109	MS — Word for Windows 6.0 循序渐近教程(含盘)	60.00 元
27	1193	MS — Word 6.0 开发人员工具包	75.00 元
28	1204	MS — Word for Windows 用户伴侣	65.00 元
29	742	MS — DOS 6.0 循序渐近教程(含盘)	44.90 元
30	1176	MS — DOS 6.2 循序渐近教程	57.00 元
31	868	MS — DOS 6.2 用户手册	48.00 元
32	1081	MS — DOS 6.0～6.2 批处理文件简明手册	25.00 元
33	981	MS — DOS 6.0 用户伴侣	53.00 元
34	1162	MS — DOS 6.0 程序员参考手册	68.00 元
35	772	MS — FoxPro 2.5 for DOS 循序渐近教程(含盘)	49.00 元
36	773	MS — FoxPro 2.5 for Windows 循序渐近教程(含盘)	49.00 元
37	859	运行 FoxPro 2.5 for DOS	45.00 元
38	860	运行 FoxPro 2.5 for Windows	45.00 元
39	1232	LaserJet 激光打印机用户伴侣	65.00 元

欲购以上图书的朋友请与 010—2541992,2562329 书刊部联系,
或传真 010—2579874,2561057 书刊部

ISBN7—5077—0779—2/TP・11
本册定价:34.00 元

计算机软件开发系列丛书

Visual C++类库深入剖析

何小路　刘　萤　编著
史小君　燕卫华　审校

学　苑　出　版　社

(京)新登字 151 号

内 容 提 要

Visual C++的基本内容大致可分成四部分：基础知识、类库、函数库和 Windows 编程。本书集中介绍了用 Visual C++开发应用程序时涉及到的各种宏、类、全局变量以及全局函数，尤其详细讨论了 Visual C++ Microsoft Foundation Class (MFC)库中的类，包括成员变量的作用、成员函数的原型、功能及用法。作为导引，第一章引入了一个利用 MFC 开发的基于 DOS 的 Forms 软件包，第二章介绍了通用的 MFC 类，第三章则先讨论了与 Windows 编程息息相关的问题，再对宏、全局函数与所有类作了详细介绍。

有关基础知识、函数库和 Windows 编程的内容，读者可参阅陆续出版的《Visual C++入门与应用》和《Microsoft C/C++ & Visual C++库函数详解》。对 C++(尤其是 Visual C++)语言程序员而言，本书是一本很实用的技术指导书籍，并可供 C、Visual C++、OOP 和 Windows 程序员参考。

需要本书者，请与北京海淀 8721 信箱书刊部联系，邮政编码 100080，电话 2562329。

计算机软件开发系列丛书

Visual C++类库深入剖析

编　　著：何小路　刘　萤

审　　校：史小君　燕卫华

责任编辑：汪亚文

出版发行：学苑出版社　　**邮政编码**：100036

社　　址：北京市海淀区万寿路西街 11 号

印　　刷：施园印刷厂

开　　本：787×1092　1/16

印　　张：26　　**字　　数**：618 千字

定　　数：1 ～ 5000 册

版　　次：1994 年 6 月北京第 1 版第 1 次

ISBN 7-5077-0779-2/TP・11

本册定价：34.00 元

学苑版图书印、装错误可随时退换

目　　录

第一章　用 MFC 库建立 DOS 应用程序 …… i

1.1　Forms 软件包 …… 1

1.2　表格的存储和检索 …… 2

1.3　表格的组成 …… 6

1.4　显示一个表格 …… 17

1.5　Form 类 …… 31

1.6　创建一个表格 …… 37

1.7　填写表格 …… 44

第二章　通用 Microsoft 基础类 …… 55

2.1　CObject 的通用功能 …… 55

2.2　通用类简介 …… 59

2.3　全局变量、通用宏和全局函数 …… 68

2.4　类参考 …… 79

第三章　用于设计 Windows 程序的 MFC 类 …… 140

3.1　Windows 程序设计类简介 …… 140

3.2　用于 Windows 程序设计的宏和全局函数 …… 146

3.3　参考 …… 177

试读结束：需要全本请在线购买：　www.ertongbook.com

第一章　用 MFC 库建立 DOS 应用程序

本章先介绍如何运用 Microsoft Foundation Class (MFC)库建立简单而完整的 C++应用程序，由此说明类库的用途。为了成功地运用 Microsoft Foundation Class 库建立 DOS 应用程序，必须事先建立必要的库。Visual C++的缺省安装将不能建立 DOS(实模式)MFC 库。有关建立实模式 MFC 库的指令，请参阅《Visual C++入门与应用》。

本章列举的实例程序非常实际地说明了大多数完整的应用程序的重要特点：用户界面和一些能完成分配给应用程序的任务的计算模块。本章列出了一个可在 MS-DOS 环境下运行的应用程序。后面的两章将详细地描述 MFC 库中的所有类、全局常量及宏。有关利用 Microsoft Foundation Class 库建立 Windows 应用程序的内容，请读者参阅《Visual C++ Windows 程序设计》。有关通用函数的用法，请参阅《Microsoft C/C++ & Visual C++库函数参考指南》。

1.1　Forms 软件包

为了说明 C++和 Microsoft Foundation Class 库在一个实际的 DOS 应用程序中的用法，下面先建立一个 Forms 软件包，通过这个 Forms 软件包可以产生并填充表格，并把完整的表格保存在文件中，然后进行检索。如果屏幕始终处于 DOS 模式下，则有必要创建 C++类以表示内存中的表格，并设计一个方案把表格存储到文件中。

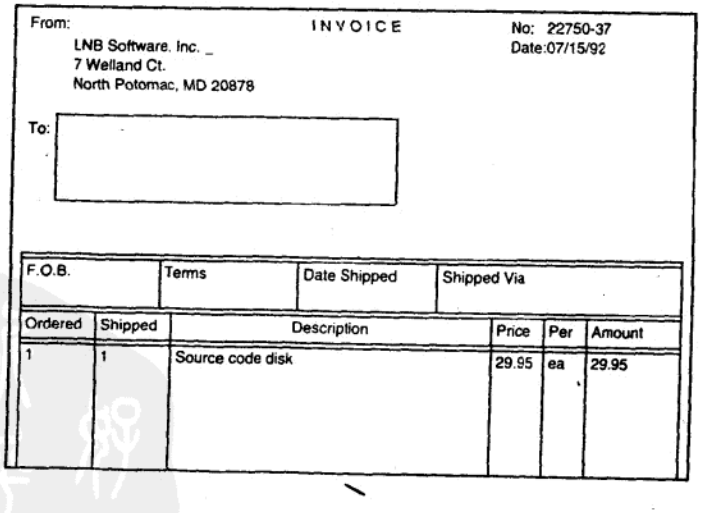

图 1.1　一个表格样本

图 1.1 以 Forms 软件的形式向用户显示了一个部分完成的表格，这个特殊的表格是一个装货清单，但这个软件却可以创建和显示任意表格。表格包含了几个域，用户可以把光标移到任何一个域中并键入信息。在设计 Forms 软件包之前，需要说明用户希望这个软件具备的功能。用户可以从以下这些要求开始：

(1) Forms 软件能使用户打开和完成一个表格，并把表格保存到磁盘文件中。

(2) 它能支持空白表格和已装满的表格，并提供一种方案把空白的和完成的表格保存到文件中。

(3) 存储方案应合理有效。特别地，此软件应仅拥有一个空白表格，但能将许多装满数据的实例放到一个表格中。

用户的要求是非常广泛的，但作为出发点，以上已经足够了。用户可以根据这些要求开始设计，随着设计的精炼，对软件的要求也越来越清晰。本章后面的部分将讲述以 Forms 软件包为基础开发和实现的 C++类。

1.2 表格的存储和检索

表格的存储和检索是软件的一项主要要求。为了建立一个 Forms 软件包，应考虑如何把表格存入文件并对其进行检索。为了能有效地存储，可把空白表格保存在一个文件中，而将装满数据的表格保存在其它文件中，这样就可以得到空白表格的一个拷贝，但装在表格中的多集合数据也存在。这两个文件可命名为：

(1) 定义文件：存有空白表格的定义。

(2) 数据文件：包含在一个特定表格中装入的数据。

图 1.2 显示了这些文件的格式。

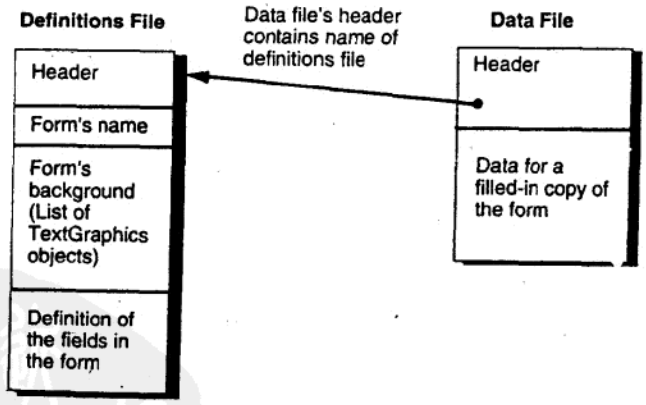

图 1.2 Forms 软件包的文件格式

这两个文件具有相似的头文件格式，从而简化了读取和解释头文件的代码。每个文件都从一个头文件开始，紧接着是表格的“数据”。在定义文件中，“数据”说明了表格的格式，而数据文件的入口则表示用户输入的数据。

在下面的章节中，通过利用 Microsoft Foundation Class 库对档案对象的支持，可对表格及其数据进行存储和检索。

1.2.1 文件头

可以把对文件头的读写封装在一个 C++类中。在 FILEHDR.H 头文件(清单 1.1)中声明的 File Header 类说明了实现类的一种方法。类的数据成员说明了下列头文件中的基本信息：

(1) 命名字符串(_signature)和版本号(_version)，它们有助于识别 Forms 软件创建的表格。

(2) filetype 字符串有两个值，说明如下：

如果文件类型为 FDEF，则文件包括表格定义(定义文件)。

如果文件类型为 FDAT，则文件是数据文件，且表格中装入了数据。

(3) filename 字符串。对于一个定义文件来说，这仅仅是文件的名字而已。而对一个数据文件，字符串是包括表格定义的文件的名字，数据文件中已经装入了数据。这种结构能使用户访问一个已有数据的表格的定义。

(4) 时间标记，它指出文件最后一次修改的时间。

(5) 两个静态字符串 form_definition 和 form_data，用于测试_filetype 域。

注意，本章广泛利用了 Microsoft Foundation Class 库中的类，例如 CString 和 CTime。这是封装一个类并到处使用它的一个很好的例子，就像是一个已建立的 C++数据类型一样。

还应注意，没有必要在一个类的私有成员变量名前加下划线。但是，使用带下划线的名字是有原因的。许多私有成员变量需要一个相应的公有成员函数，这样能使其它类访问私有变量。如果用户命名的私有成员变量名带有下划线，用户就可以使用不带下划线的相同名称访问该函数。例如，在 FileHeader 类中，将私有成员变量命名为_filetype，将成员函数命名为 filetype。通过下划线(_filetype)选择变量名，可以为访问函数提供一个更具逻辑性的名字(filetype)。

清单 1.1 FILEHDR.H：FileHeader 类的声明，它将文件头模块化

```
//----------------------------------------------------------
//  File: filehdr.h
//
//  声明 FileHeader 类

#if ! defined(__FILEHDR_H)
#define FILEHDR_H

#include <afx.h>
```

```
class FileHeader:public CObject
{
    DECLARE_SERIAL(FileHeader)

public:
    FileHeader() :_signature("LNBFORM"),_version(100) {}

// 覆盖 Serialize 函数
    virtual void Serialize(CArchive& archive);

    unsigned short version() { return version;}
    void version(unsigned short vnum) { _version = vnum;}

    CString signature() { return signature;}
    void signature(const char * sig) { signature = sig;}

    CString filename() { return filename;}
    void filename(const char * fname) { filename = fname;}

    CString filetype() { return filetype;}
    void filetype(const char * ftype) { filetype = ftype;}

    CTime mod_time() { return modtime;}
    void mod_time(CTime mt) { _modtime = mt;}

    FileHeader& operator=(const FileHeader& hdr);

    static CString _form_definition;
    static CString _form_data;

private:
    CString          signature;
    unsigned short   _version;
    CString          _filetype;
    CString          _filename;
    CTime            _modtime;          // 修改 Time
};

#endif
```

FILEHDR.H 头文件定义了几个嵌入成员函数，其余的出现在 FILEHDR.CPP 文件中，参见清单 1.2。这个文件把静态 CString form_definition 初始化为 FDEF，把 form_data 初始化为 FDAT。这些字符串用于测试头文件中的 _filetype 域，以决定文件拥有一个列表的定义还是列表的数据。

FileHeader 类是从 CObject 类中派生出来的，而 CObject 类是 Microsoft Foundation Class 库中众多类的基类。由于 FileHeader 类继承了 CObject，所以可以使用 Microsoft Foundation Class 库中已建立的对“连载”的支持，这个术语被 Microsoft 用来指示文件中对象的存储和检索。

必须在 FILEHDR.H 中包含 DECLARE_SERIAL 宏，在 FILEHDR.CPP 中包含 IMPLEMENT_SERIAL 宏，这样才能确保为文件中的 FileHeader 对象建立档案。有必要覆盖 Serialize 函数。由清单 1.2 可见，Serialize 函数首先调用基类 CObject 的 Serialize 函数，然后为 FileHeader 类的成员变量建立档案。

清单 1.2　FILEHDR.CPP：FileHeader 类的实现

```
//----------------------------------------------------------
//   File: filehdr.cpp
//   读取并解释文件头的 FileHeader 类

#include "filehdr.h"

CString FileHeader::_form_definition = "FDEF";
CString FileHeader::_form_data = "FDAT";

// 如果用户想将该类的对象归档，则需要以下这部分内容

IMPLEMENT_SERIAL(FileHeader,CObject,100)

//----------------------------------------------------------
//   Serialize
//   函数存档并保存文件头

void FileHeader::Serialize(CArchive& archive)
{
// 首先调用基类的 Serialize 函数
    CObject::Serialize(archive);

// 现在读/写该类的成员变量
    if (archive.IsStoring())
    {
        archive << signature << version << filetype;
        archive << filename << modtime;
    }
    else
    {
        archive >> signature;
        archive >> version >> filetype;
        archive >> filename >> modtime;
    }
}
//----------------------------------------------------------
//   operator=
//   将一个 FileHeader 赋给另一个

FileHeader& FileHeader::operator=(const FileHeader& hdr)
```

```
{
    if(this ! = &hdr)
    {
        signature = hdr._signature;
        version = hdr._version;
        filetype = hdr._filetype;
        filename = hdr._filename;
        modtime = hdr._modtime;
    }
    return * this;
}
```

1.3 表格的组成

一个表格有两个主要组成部分：

(1) 背景部分：包括在空白表格中所见到的直线和格子、图形以及打印在表格中的文本。

(2) 域部分：当装入表格时可用于输入信息。

当把一个表格保存在文件中时，背景信息和域信息都保存在定义文件中，但域中真正的数据则保存在数据文件中。接下来讨论一种表示表格中的背景和域的方法，阅读这段说明时可参考图 1.3。

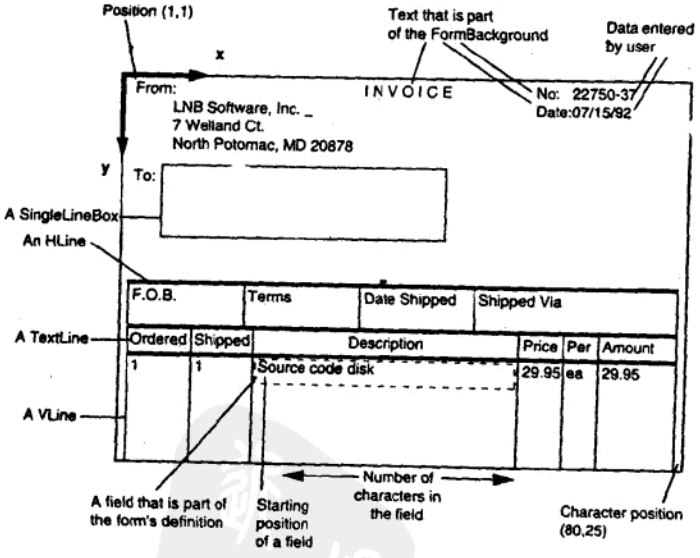

图 1.3 一个表格的组成部分

试读结束：需要全本请在线购买： www.ertongbook.com

1.3.1 FormBackground 类

表格的背景是由图形元素(即直线、框和文本)组成的。假定有一个名为 TextGraphics 的类被用来提供这样的元素,就可以创建一个 FormBackground 类把这样的图形对象保存在一个双向链表中。为了创建一个 FormBackground 双向链表,只需要在 Microsoft Foundation Class 库中从 CObList 类派生出 FormBackground 类即可。由于 CObList 类在 AFXCOLL.H 头文件中已经声明过了,所以 FORMBG.H 头文件也必须包括在内(见清单 1.3)。

清单 1.3 和 1.4 列出了 FORMBH.H 和 FORMBG.CPP 文件,其中定义了 FormBackground 类。FormBackground 提供了成员函数,例如 first,last,next 和 previous,有必要将 TextGraphics 对象保存在这个列表中。这些成员函数是根据 CObList 类的成员函数定义的,FormBackground 就是从这个类派生出来的。

FormBackground 中的独立的成员变量 pos 用于存储 TextGraphics 对象列表中当前项的位置。变量 pos 为 POSITION 类型,它是一个定义在 AFX.H 头文件中的类型。AFXCOLL.H 中也包括了这个类型。

清单 1.3 FORMBG.H:FormBackground 类的头文件

```
//-----------------------------------------------------------
//  File:formbg.h
//  表格的背景;背景是图形对象的集合

#if ! defined(__FORMBG_H)
#define FORMBG_H

#include <afxcoll.h>
#include "txtgraph.h"

class FormBackground :public CObList
{
    DECLARE_SERIAL(FormBackground)

public:
    FormBackground() {}
    ~FormBackground() { delete_all();}

    TextGraphics * first()
    {
        pos = GetHeadPosition();
        return (TextGraphics * ) GetNext(pos);
    }

    TextGraphics * last()
    {
        pos = GetTailPosition();
        return (TextGraphics * ) GetPrev(pos);
```

```
    }

    TextGraphics * next()
    {
        if(pos == NULL)
            return NULL;
        else
            return (TextGraphics * )GetNext(pos);
    }

    TextGraphics * previous()
    {
        if(pos == NULL)
            return NULL;
        else
            return (TextGraphics * )GetPrev(pos);
    }

    TextGraphics * current()
    {
        if(pos == NULL)
            return NULL;
        else
            return (TextGraphics * )GetAt(pos);
    }

    unsigned short count() { return GetCount();}

    void add(TextGraphics&. tgr)
    {
        AddTail(tgr. clone());
    }

    void delete_all();

private:
    POSITION  pos;
};

#endif
```

像 FileHeader 类一样，FormBackground 类依靠 Microsoft Foundation Class 库中的连载能力把一个表格的背景保存在一个文件中，并可随意进行检索。FormBackground 类不需要定义 Serialize 函数，然而，以后用户将发现每个 TextGraphics 对象都必须有一个 Serialize 函数来确保表格的背景能够被合适地保存。

清单 1.4　FORMBG.CPP：FormBackground 类的成员函数

```
//-----------------------------------------------------
//  File: formbg.cpp
//  实现保存在表格背景中的 FormBackground 类

#include "formbg.h"
#include "txtgraph.h"

IMPLEMENT_SERIAL(FormBackground,CObList,100);
//-----------------------------------------------------
//  delete all
//  删除 FormBackground 的所有入口

void FormBackground::delete_all()
{
    TextGraphics * tgr;
    for(tgr = first();tgr != NULL;tgr = next())
        tgr->destroy();
    RemoveAll();
}
```

1.3.2　FieldList 类

FormBackground 类包括了一个表格的背景和一个从 CObList 派生出来的 FieldList 类，并把表格的域保存在一个链表中，每个域由一个 Field 类的实例来表示。

正如 FormBackground 类为 TextGraphics 所做的，FieldList 为便于访问 FieldList 内包含的域而提供了许多函数。清单 1.5 和 1.6 所示的文件实现了 FieldList 类。

清单 1.5　FIELDLST.H：FieldList 类的头文件

```
//-----------------------------------------------------
//  File:fieldlst.h
//  定义表格结构和内容的双向链表

#if !defined(__FIELDLST_H)
#define FIELDLST_H

#include <afxcoll.h>
#include "field.h"

class FieldList :public CObList
{
public:
    FieldList() :pos(NULL) {}

    FieldList(CArchive& archive) { read(archive);}
```

```
~FieldList() { delete_all(); }

Field * first()
{
    pos = GetHeadPosition();
    return (Field * ) GetNext(pos);
}

Field * last()
{
    pos = GetTailPosition();
    return (Field * ) GetPrev(pos);
}

Field * next()
{
    if(pos == NULL)
        return NULL;
    else
        return (Field * )GetNext(pos);
}

Field * previous()
{
    if(pos == NULL)
        return NULL;
    else
        return (Field * )GetPrev(pos);
}

Field * current()
{
    if(pos == NULL)
        return NULL;
    else
        return (Field * )GetAt(pos);
}

void current(Field * f)
{
    pos = Find(f,GetHeadPosition());
}

unsigned short count() { return GetCount(); }

Field * locate(const char * id);

void add(Field&. f)
```

```
    {
        Field * pf = new Field(f);
        AddTail(pf);
    }
    void add_data(const char * id,CArchive& archive);
    void add_data(const char * id,CString& d);
    void add_data(const char * id,const char * s);

    CString data();                          // 返回当前域的数据
    void data(CString& d);                   // 设置当前域的数据

    void read(CArchive& archive);
    void write(CArchive& archive);

    void read_data(CArchive& archive);
    void write_data(CArchive& archive);

    void delete_all();

private:
    POSITION  pos;                           // 当前项的位置
};

#endif
```

清单 1.6　FIELDLST.CPP:FieldList 类的成员函数

```
//---------------------------------------------------------
//  File:fieldlst.cpp
//  保存域的列表及其内容

#include "fieldlst.h"
#include <string.h>
#include <iostream.h>
//---------------------------------------------------------
//  read
//  从一个档案中读取域定义

void FieldList::read(CArchive& archive)
{
// 读入多少元素
    unsigned short entries;
    archive >> entries;

// 逐个读取域定义
    if(entries > 0)
    {
        unsigned short i;
```

```
        for(i = 0;i < entries;i++)
        {
            Field f;
            f.read(archive);
            add(f);
        }
    }
}
//-----------------------------------------------
//  write
//  将域写入一个档案内
//  这个函数保存数据以外的所有内容

void FieldList::write(CArchive& archive)
{
// 首先写入所有元素的个数
    archive << count();

// 逐个写出域
    if(count() > 0)
    {
        Field * p_f;
        for(p_f = first();p_f != NULL;p_f = next())
        {
// 将域保存在档案中
            p_f->write(archive);
        }
    }
}
//-----------------------------------------------
//  add_data(const char *,CArchive&)
//  从一个档案中读取数据

void FieldList::add_data(const char * id,CArchive& archive)
{
    Field * p_f = locate(id);
    if(p_f != NULL)
    {
        p_f->read_data(archive);
    }
}
//-----------------------------------------------
//  add_data(const char *,CString&)
//  为域设置数据

void FieldList::add_data(const char * id,CString& d)
{
    Field * p_f = locate(id);
    if(p_f != NULL)
```


试读结束：需要全本请在线购买： www.ertongbook.com