所有范例程序的源代码可到
www.khp.com.cn下载

Visual C++ 2005
工具书

.NET
珍藏版

新一代
Visual C++ 2005
程序设计

林俊杰　编著

- 最详尽的MFC类成员用法
- 应用Visual C++ 2005和MFC构建Win32应用程序
- 丰富示例助您打造最新的、符合设计标准的窗体应用程序

清华大学出版社

新一代

Visual C++ 2005 程序设计

林俊杰　编著

清华大学出版社

内 容 简 介

本书详细描述了使用Visual C++ 2005与MFC开发Windows窗体应用程序的方法，通过循序渐进的教学模式，一步步教读者构建功能复杂的Win32应用程序。

全书共19章，前3章简单回顾了C++语言的重点并介绍MFC库中与窗口无关的一些基础类，接下来逐章讲述窗口程序设计的基本概念、窗口类CWnd、Windows控件、键盘及鼠标的相关用法、控制菜单的方法、MFC文档/视图结构、绘图及处理位图的方法、打印及控制打印图形尺寸的方法、ActiveX控件。第18、19章是较高级的主题，分别讨论Win32模式下的内存管理、进程及多线程系统，以及registry系统。

本书面向 Visual C++初学者，要求读者必须有 C++语言基础，而且能够操作 Windows，如果在面向对象及数据结构方面有深入的了解则更佳。

图书在版编目（CIP）数据

新一代Visual C++ 2005 程序设计/林俊杰编著. — 北京：
清华大学出版社，2006

ISBN 7-302-13950-4

Ⅰ. 新… Ⅱ. 林… Ⅲ. C 语言－程序设计
Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2006）第 118761 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编**：100084
社 总 机：010-62770175　　**客户服务**：010-82896445
组稿编辑：科海
文稿编辑：何武
封面设计：林陶
版式设计：科海
印 刷 者：北京市耀华印刷有限公司
发 行 者：新华书店总店北京发行所
开　　本：787×1092　1/16　　**印张**：30.75　　**字数**：748 千字
版　　次：2006 年 11 月第 1 版　　2006 年 11 月第 1 次印刷
书　　号：ISBN 7-302-13950-4/TP・8386
印　　数：0 001~4 000
定　　价：49.00 元

访仙得书记

话说台北有个姓林的书生，因为写不出个像样的 Windows 的应用程序，感到十分烦恼，坐在桌前猛抓头皮苦思，只见头顶青丝日渐稀少，再抓下去怎么了得。不知不觉中，林生俯案卧，昏昏睡去。

睡梦中，依稀听到窗外传来窃窃私语声，林生耐不住好奇，委身以窥之，见窗外二老对坐，白发苍苍，眉长及耳，耳垂至肩，手长过膝，非常人也。一曰：近日作何生意？对曰：闲来无事，便着一书。非常书，乃教人用 MFC 设计 Win32 应用程序之宝典也。林生闻之，更加好奇，竖耳听之。

“此书有何特点？”，一老问之。对曰：凡能以 C++设计程序自如、能操作 Windows 无碍、稍具概念者，即可按文索骥，习得 MFC 之技。闻者大惊，叹曰：真奇书也！“区区小事，何需大惊小怪？”，又续曰：此书不以 AppWizard 虚浮之法介绍 MFC，而是以真枪实弹的内容介绍 MFC，但是又不令人感到困难，旦旦而习之，可扎实打好基础，高人一等！除了货真价实的内容之外，这本书还涵盖 Windows 的最新内容，此乃他人不能及也。

只见一人滔滔不绝，另一人听得目瞪口呆，连声叫好。林生暗觉惊讶，又闻：凡涉及文档/视图、SDI、MDI、各种控件与通用控件、Windows 绘图、位图、字体与打印等，本书无不涵盖。不仅是概念的介绍，还有成员函数、API 的详细介绍，因此不仅适合初学者，对应用开发人员也具有参考价值。另一老问：阅读时，有无任何特别注意事项？答曰：初学者不需要逐字研读，只需要先阅读看得懂的部分即可。但是绝不可忽略实例，它包含实战的精髓。例如，多线程的介绍就包含在实例之中，第 14 章的实例还讲述如何处理 DIB 位图！如此重要的内容，怎可跳过？

语毕，另一老面色有异，对之曰：如此奇书，若为常人得之，天下必大乱，不如寻一隐密处藏之。吾知一山罕有人迹，君不妨将宝典藏于此山。林生听之大喜，遂默记之。转眼间，铃声大作，两老化作轻烟消逝无踪。林生自桌上惊醒，方知是场梦，但梦境似真，林生心觉有异，遂计划至山中寻宝典。翌晨整装出发，经数日，抵梦中老人所言之山，山势险峻，非凡人所能至，此时突然狂风大作，飞沙走石。风止，蜿蜒小径出现眼前，林生沿路行，遂至藏书处，得梦中老人之宝典。取毕，循原路回，突然狂风大作，飞沙走石，风息，径灭，若无事然。

得奇书后，林生费数月改写之，得此书，并交出版社出版，以传世人，并作此记志之。

林俊杰

2006 年 9 月

目　录

第1章

综　述

初学者经常提出这样的问题：是学 C++好还是学 Visual C++好？

其实，这是个很奇怪的问法，就好像在问：是学开车好还是学开福特车好？开车是一种普遍的技术，用这个技术可以控制福特车或奥迪车，这在技术上没有很大的差别，最多只是一些开关的位置不同而已。同样的道理，C++是一种程序语言，现在已经成为 ISO 标准，是一种普遍的技术。Visual C++只是一种协助用户使用 C++语言设计软件的工具。

类似的工具还有很多，例如 Borland C++ Builder，甚至还有免费的 GNU C++，它们都是协助用户使用 C++设计软件的工具。既然可以使用免费的 GNU C++，为什么还要花钱买昂贵的 Visual C++呢？

首先，Visual C++有功能强大的集成开发环境，在此环境中可以方便地进行项目管理、编写 C++源程序、编译、调试，而这些在 GNU C++中都不具备。更重要的是，Visual C++提供了一套称为 Microsoft Foundation Class（MFC）的类库（classes library），这套由微软开发的库已经成为设计窗口应用程序的准工业标准了。

标准 C 语言库提供了 400 种库函数，例如 printf()、fopen()等，这些函数可以实现数据存取、数学计算、字符串处理等功能。MFC 包含了上百种事先用 C++写好的类，将这些类组合起来，可以编写功能强大的 Windows 应用程序。

除了 MFC 之外，其他厂商也有类似的产品，例如 Borland C++ Builder 的 VCL，也是用 C++设计的库，可以协助用户快速地开发窗口应用程序。这些库各有所长，MFC 的最大优势在于能抢先支持各种新功能，如 ActiveX 等，但是在面向对象技术方面略逊一筹，例如，MFC 的类难以实现多重继承。

1.1　本书结构

本书的主题是讲述如何用 MFC 设计窗口应用程序。在阅读本书之前，要求读者必须学会 C++语言，而且能够操作 Windows，如果在面向对象及数据结构方面有深入的了解则更好。

本书第 2 章简单地回顾了一些 C++语言的重点；第 3 章介绍 MFC 库中与窗口无关的一些基础类；第 4 章讲述窗口程序设计的基本概念。

第 5 章将实际动手设计一个最简单的窗口应用程序。接下来，第 6 章详细介绍窗口类 CWnd，不过第 6 章的后半部分内容稍微有点深，初学者可以先跳过不看。第 7 章讲解用 Visual C++设计以对话框为主的应用程序，并且在此处介绍 Windows 的控件。第 8 章讲述输入设备——键盘及鼠标的相关用法。第 9 章介绍控制菜单的方法。第 10 章是本书最重要的一章，介绍 MFC 的文档/视图结构，请务必搞清楚。第 11 章介绍用 MFC Application Wizard 辅助设计文档/视图结构软件的方法。第 12 章是第 7 章的延伸，详细地介绍了各种控件的用法，初学者可先跳过不看。

第 13 章的主题是绘图，Windows 绘图的所有知识都可以在这里找到。第 14 章介绍处理位图的方法。在该章后半部分，笔者设计了一个可以处理 Windows 常见的*.BMP 文档的类，十分有用，请务必仔细阅读。仅在屏幕上显示图形还不够，有时需要将其打印出来，第 15 章介绍打印以及控制打印图形尺寸的方法。

第 16 章介绍在 Windows 95 之后才出现的通用控件，该章的范例就是逐步创建一个类似文件管理器的应用程序。第 17 章介绍如何在程序中使用流行的 ActiveX 控件。第 18、19 章是比较高级的主题，分别讨论 Win32 模式下的内存管理、进程及多线程系统，以及 registry 系统，初学者可跳过不看。

1.2 本书约定

File|Open → *如果是指 File 菜单中的 Open 选项，统一以竖线分隔这些选项。*

```
1.  void main()
2.  {
3.  }
```

→ *程序代码左侧的数字是行号。*

int getch(); → *函数的原型(prototype)，也就是函数的名称、返回值类型、参数。*

第2章

C++重点回顾

如果用户对 C++程序语言还不熟悉，那么本章内容刚好适合你。因为笔者的前几本书中都假设读者对 C++有相当程度的了解，不过后来发现写信来问 C++问题的人远比对内容有疑问的人多得多。因此，本书特意增加了这一章，介绍一些 C++语言的重点概念。不过，也不要期待仅靠这一章就可以学会 C++，读者还需要阅读其他介绍 C++的书。

2.1 类、对象与实例

这 3 个名词经常困扰许多人，因此需要对它们进行重点说明。“类”就是种类、品种，比如说狗，就是泛指长了一条尾巴，会吠叫会咬人的哺乳动物，不过每一条狗都是独立、有个性的小生命，有的狗叫小花，有的叫小黄，有的叫来福。因此，狗是一种动物类，而小花、小黄、来福是狗的“实例”；在大多数情况中，实例指的就是“对象”。

2.2 构造函数（Constructor）

Constructor 在这本书中统一称为构造函数。创建实例时，计算机就会自动跳到该实例的构造函数去执行。通常，构造函数中的内容都是和初始化实例相关的程序代码。

不过，构造函数也会有一些奇怪的用法。首先，类可能会有数个父类，而父类也有它自己的构造函数，因此子类必须调用父类的构造函数，以便对从父类继承的部分进行初始化。例如下面这段程序：

```
1.  class Parent
2.  {
3.  public:
4.       int parent_var;
5.
6.       Parent ()           { parent_var = 10; }
7.       Parent ( int init ) { parent_var = init; }
8.  };
9.
10. class Child : public Parent
11. {
12. public:
```

```
13.      int child_var;
14.
15.      Child ()                          { child_var = 23; }
16.      Child ( int i ) : Parent ( i ) { child_var = 24; }
17.      Child ( int i, int j )
18.          : Parent ( i ), child_var ( j )     {}
19. };
20.
21. Child inst_1;
22. Child inst_2 ( 100 );
23. Child inst_3 ( 100, 200 );
```

在这段程序中，如果产生 Child 的实例 inst_1，inst_1 的成员变量 parent_var 会被设成 10，child_var 会被设成 23，这是因为计算机会跳到构造函数 Child::Child()中执行，而在将 child_var 设成 23 之前，会先跳到 Parent::Parent()中执行。inst_2 的 parent_var 会被设成 100，因为 Child::Child(int)冒号后的指令要求计算机先运行 Parent::Parent(int)，再把 child_var 设置成 24。

Child 第 3 个构造函数就更奇怪了，在冒号之后调用 Parent 构造函数还不难了解，但是 child_var(j)就有点奇怪。原来这一行的作用是调用 child_var 对象的构造函数，因为它只是基类，因此等价于 child_var = j。

2.3 匿名实例（Nameless Instance）

请看下面这段程序：

```
1.  int GetValue ( Child& object )
2.  {
3.       return object.child_var;
4.  }
5.
6.  void Trial ()
7.  {
8.       int x;
9.       x = GetValue ( Child ( 4, 5 ) );
10. }
```

在这段程序中，函数 GetValue(Child&)的功能是返回对象 object 成员变量 child_var 的值。那么，函数 Trial()的变量 x 会是多少呢？

最奇怪的是第 9 行，GetValue()的参数竟然只有 Child(4,5)，这是怎么回事？其实这种用法等价于：

```
1.  void Trial ()
2.  {
3.       int x;
4.       Child obj ( 4, 5 );
5.       x = GetValue ( obj );
6.  }
```

这段代码首先产生对象 obj，然后将它当成参数传给 GetValue()。前后两段代码的最大

差别在于，前者产生的对象没有名字，而后者产生的对象的名字叫 obj。再看下面一段示例。

```
return HelloApp().Run()
```

它等价于：

```
HelloApp app;
return app.Run();
```

为什么会这样呢？这要从 C++创建对象的方法谈起。在第 2 段代码中，执行到 HelloApp app 这一行时，计算机会在内存中分配一块空间当做 app 的主体。在我们的概念中，变量 app 就代表这块内存。反过来看第一段程序代码，从左向右看，当计算机执行到 HelloApp() 这一行时，它也会在内存中分配一块空间当做 HelloApp 实例的主体，只是没有一个类似 app 的变量来代表这块内存中的对象。即使如此，这个对象还是存在的，只不过它没有名字。访问此对象的唯一方法就是紧接在后面加一个点，再加上成员的名称。因此，说构造函数没有返回值是不正确的，其实构造函数的返回值就是对象本身。

2.4 虚函数

如果在类的成员函数之前加了 virtual 修饰符，那么该函数就称为虚函数（virtual function）。例如在下面这段程序中，Object::foo()就是虚函数：

```
class Object
{
    virtual void foo ( void );
};
```

虚函数和非虚函数有什么区别呢？举个真实世界中的例子，儿子的姓氏和手都是从父亲继承来的，虽然同样是继承来的东西，但儿子的姓氏实际上还是沿用父亲的姓氏，不过手却是儿子自己的手。在这个例子中，姓氏不是虚函数，只有手才是虚函数。

真实世界的例子或许不足以恰当地说明 C++的世界。再看一段示例程序：

```
1.  class Father
2.  {
3.      protected:
4.      int     my_firstname()      { return 100; }
5.      virtual int my_hand()       { return 101; }
6.
7.      public:
8.      int         get_firstname()     { return my_firstname(); }
9.      int         get_hand()          { return my_hand(); }
10. };
11.
12. class Kid : public Father
13. {
14.     protected:
15.
16.     int     my_firstname()      { return 102; }
17.     virtual int my_hand()       { return 103; }
18. };
```

```
19.
20. void Test ()
21. {
22.     int val1, val2, va3;
23.     Kid kid;
24.     Father *ptr;
25.
26.     var1 = kid.get_firstname();
27.     var2 = kid.get_hand();
28.     ptr = new Kid;
29.     var3 = ptr->get_hand();
30.     delete ptr;
31. }
```

类 Father 有两个 protected 成员函数：my_firstname()与 my_hand()。它的 public 成员函数 get_firstname()是调用 my_firstname()后再返回其值，get_hand()是调用 my_hand()后再返回其值。继承自类 Father 的新类 Kid，理所当然地继承了 Father 所有的成员函数，不过在 Kid 中又重新定义了 my_firstname()与 my_hand()这两个函数，这种行为称为重载（override）。

在函数 Test()中有个 Kid 的实例 kid。在第 28 行调用了 kid.get_firstname()，因为这个函数继承自 Father，所以它理当会调用成员函数 my_firstname()。但实际上它到底会调用谁的 my_firstname()？是 Father::my_firstname()还是 Kid::my_firstname()？

再看第 29 行，它调用 kid.get_hand()。毫无疑问，get_hand()还会再调用 my_hand()。同样的问题又发生了，它到底是调用 Father::my_hand()还是 Kid::my_hand()？

应该已经注意到 my_hand()是虚函数，而 my_firstname()不是，因此 get_hand()、get_firstname()两个函数的行为必然是不同的。因为 my_hand()是虚函数，所以 get_hand()会调用它所属的那个对象的 my_hand()。在 Father 中，get_hand()调用的是 Father::my_hand()，而在 Kid 中，get_hand()是调用 Kid::my_hand()。get_hand()选择 my_hand()的方式称为动态确定。

另一方面，因为 my_firstname()不是虚函数，因此 get_firstname()只会调用 Father::my_firstname()，而不会调用 Kid::my_firstname()。get_firstname() 选择 my_firstname()的方式称为静态确定。

由于调用虚函数时，计算机会动态确定要调用哪个函数，因此这种调用方式又被称为动态绑定（dynamic binding），而调用非虚函数时的行为则被称为静态绑定（static binding）。

觉得神奇吗？其实很简单，编译器会为每个类创建一个称为虚函数表（virtual function table）的表格，这个表格由函数指针组成，每个指针都指向一个虚函数，因此调用虚函数时，计算机会先检查这个表，找出函数指针后再跳到指定地址去执行，这样就形成类似的动态效果。严格地说，这还不够动态，Java 才是真正的动态绑定。

虚函数在本书、MFC 以及其他的类库中占有十分重要的地位。这些类库在设计时并不知道程序员会如何运用它们，因此总得留一些进入点以便插入、更改或者扩展程序代码，在以后的章节代码中，需要修改的地方通常都是虚函数，只要在从库派生的新类中重载虚函数，加上新功能，就可以构造出所需的应用程序。

2.5 异常处理

在传统的 C/C++程序设计中，函数的返回值扮演了多重角色，它有时代表了计算的结果（例如 sin()、cos()这类函数的返回值就是三角函数值），有时又代表出现的错误（例如 ferror()在文件访问发生错误时会返回非零值），甚至还有更复杂的情况，返回值综合了以上两种用途。不同公司的产品、不同时代的类库，就有不同的设计哲学，这给程序员进行程序设计增加了难度。

顺序（例如一般的算术指令）、循环（例如 for）和分支（例如 if...then...else）是结构化程序的 3 大要素，也就是说，一个遵循结构化规则的程序，不会用到 goto 语句，并且，所有用到的指令都不出以上 3 大类。

也许有人会问：为什么不能用 goto？大多数人认为源程序是写给计算机的一串指令，其实不然！源程序是人们思考后有规则、有系统地写下的剧本，这个剧本经过编译器翻译成机器码后才交给计算机执行。因此，源程序其实是写给人看的，而不是计算机，因此，如何让源程序容易被人看懂，最好连没受过训练的 3 岁小孩也能看懂，这是许多人的梦想，所以我们才会不断强调结构化程序的重要性，这也是不使用 goto 的原因。

有些人认为仅靠顺序、循环和分支 3 类指令还不足以写出明晰易懂的程序，如果可以把错误处理的部分独立于正常流程之外，源程序的结构性将会更明确，因此新一代的程序语言开始添加错误处理指令，其中包括新版的 C++及 Java 等。

```
1.  try
2.  {
3.        ......
4.        throw ObjectX;
5.        ......
6.  }
7.  catch ( Class1 Object )
8.  {
9.        .....
10. }
11. catch ( Class2 Object )
12. {
13.       .....
14. }
```

C++异常处理的基本形式与前面的程序代码差不多，也需要用到 try、catch、throw 这 3 个指令。在 try 之后的一对大括号包含了一段程序代码（其中可以再包含一层 try...catch...），在这一段程序代码中如果发生了异常，计算机就会自动跳到 try 大括号后面的 catch 代码块中执行。

如何产生异常呢？只要用 throw 指令抛出一个对象即可。异常对象被抛出之后，计算机会自动找到最近一层的 catch()处理这个异常对象。每个 catch 只能接受某类异常对象，例如，在上面的程序代码中，第一个 catch 只能接受类为 Class1 的对象，第二个 catch 只能接受类为 Class2 的对象，如果被抛出的对象和某个 catch 所能接受的类相同，或者属于它的