



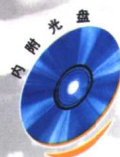
HWNDPA

temp=pl

temp=pl

VC++

编程技术与 难点剖析



辛长安 梅 林 编著

VC++ 提供给程序员的
是**猎枪**
而不是有限的
食物
但要求您是一个出色的
猎手

- 剖析 VC++ 重点难点技术
- 揭示 MFC 封装机制和体系结构
- 快速掌握 VC++ 语言精华
- 理清 VC++ 程序设计思想

delete temp

```
{  
    if(plist->Wndhandle==hWnd)
```

```
{  
    HWND* CWND*temp;
```

```
temp=plist;
```

```
inplist->Next
```

```
inplist->Dups=0
```

```
inplist->Next=plist->Next;
```



清华大学出版社

<http://www.tup.tsinghua.edu.cn>



VC++编程技术与难点剖析

辛长安 梅 林 编著

清 华 大 学 出 版 社

(京) 新登字 158 号

内 容 简 介

本书以 C++ 语法和 VC++6.0 MFC 类库编程的重点与难点内容为主线, 从实践应用的角度, 结合经典的实例阐述 C++ 类的特性, 深入剖析 VC++, 特别是 MFC 编程的体系结构, 使有初级基础的读者能够快速掌握 VC++ 的 MFC 程序设计思想和 C++ 的语言精华。

全书共分 10 章。第 1 章阐述 C++ 的关键语法, 包括 C++ 类的几个主要特性; 第 2 章介绍 MFC 类结构中的几个重要类, 并重点剖析窗口封装类 CWnd 及其派生类, 以及 Windows 窗口操作的相关内容; 第 3 章全面阐释 MFC 的消息映射与消息处理机制; 第 4 章从子窗口的角度阐述子控件的实现以及自定义控件的创建技术, 并重点学习控件的属主画与自定义画特性; 第 5 章从几个方面学习优化程序界面的技术; 第 6 章全面论述如何控制 MFC 的文档\视图框架结构, 包括单文档、多文档、视图拆分等; 第 7 章论述绘制和打印两个方面, 学习如何应用 MFC 的设备环境类, 并重点阐述脱离视图支持的 MFC 打印功能; 第 8 章专门讨论 MFC 的多线程编程控制; 第 9 章学习如何应用 VC++ 控制 Web 的 DHTML 网页; 第 10 章较全面地阐述 C++ 及 MFC 的异常处理技术。

本书并非泛讲 VC++ 编程基础的入门教材, 因此适合有一定 VC++ 学习经验或编程基础的读者。

版权所有, 翻印必究。

本书封面贴有清华大学出版社激光防伪标签, 无标签者不得销售。

书 名: VC++ 编程技术与难点剖析

作 者: 辛长安 梅 林 编著

责任编辑: 欧振旭

出 版 者: 清华大学出版社(北京清华大学学研大厦, 邮编 100084)

<http://www.tup.tsinghua.edu.cn>

印 刷 者: 北京市清华园胶印厂

发 行 者: 新华书店总店北京发行所

开 本: 787×1092 1/16 印张: 33 字数: 762 千字

版 次: 2002 年 4 月第 1 版 2002 年 4 月第 1 次印刷

书 号: ISBN 7-900641-65-3

印 数: 0001~5000

定 价: 46.00 元(附光盘)

自序

日光灯依然咝咝地亮着，电脑也嗡嗡地运转着，窗外偶尔还传来夜归者的喧嚷。我推开键盘，挺一挺脊背，昏昏沉沉地仰靠在椅子上……

经过数月的努力，本书终于完稿了，我品味到了一种长征后的喜悦。但这种喜悦稍纵即逝，迅速又化为从前的平静。

也许是热衷于学习的原故吧，为发现切合自己需要的学习用书，我时常光顾书店。而每每在流行的计算机编程语言，特别是 VC++ 编程的书海中寻觅时，总让我感到异常尴尬。好像正是因为面对大海，而无法找到饮用的水源。处在一个空前繁荣的计算机图书市场，而我们却好似沙漠中苦行的骆驼，何等悲哀！进而，我又感到一种压力，一种责任。作为一个业内的工作者，总应该为本行业的技术交流，尽微薄之力。

是一个偶然的机，使我能够为这个许久以来的愿望做一点事。这样，在参考一些相关书籍和技术文章的基础上，结合自己多年的工作和学习经验，我编写了这本书。如果本书能够作为一滴水，奉献给惯于在沙漠中求索的广大 VC++ 爱好者，使舔吸的人继续向前走一段路，也就算是它的成功了。

我写作本书的出发点是：使有初级基础的读者能够理清 VC++，特别是 MFC 编程的体系结构，快速掌握 MFC 程序设计思想和 C++ 语言精华。

因为 MFC 是以类库的形式封装 Win32 程序设计，整个体系结构不是很直观。所以往往经过很长一段时间的学习后，还不能理解 MFC 的封装机制，摸不清它的体系结构。因而难以在 MFC 中游刃有余。是的，这的确有些困难。一方面你要精通 C++ 语法，另一方面你要熟悉 Win32 API 编程，最后你还要阅读并理解大量的 MFC 源代码。本书正是以 MFC 桌面程序设计为主，兼顾 C++ 语法和 Win32 API 编程，通过剖析大量的 MFC 源代码，对 MFC 程序设计的重点难点技术内容进行彻头彻尾地阐述，从而揭示 MFC 的体系结构，阐释其程序设计思想。

由于使用 VC++ 编程很大程度地依赖于对 C++ 语法的熟练运用，而 C++ 语法又不易熟练驾驭。许多读者朋友在并不熟悉 C++ 语法，特别是 C++ 类的特性之前，就着手学习 MFC。这可能是忽视了一个事实：VC++ 并不是纯可视化的编程工具。本书注重讲解 C++ 语法基础，在第 1 章中对 C++ 类的几个关键特性进行较详尽的阐述。同时又考虑到学习 C++ 语法是为 MFC 服务的，所以针对这些语法现象在 MFC 类库中的应用，分别进行了例举和总结。

窗口操作是 Windows 编程的主要内容，所以窗口基类 CWnd 在 MFC 类结构中也处于核心地位。同时，Windows 操作系统采用消息驱动机制，如果将窗口看作是应用程序的实体，那么窗口消息就是流过这个实体的血液。所以理解 MFC 的窗口消息处理机制尤为重要。可以说，如果掌握了 CWnd 类封装 Windows 窗口的实质，即通过与窗口句柄建立映射关系而操作窗口，通过对窗口子类化而处理窗口消息；以及 MFC 的消息映射机制，包括各

种消息的宏定义和传递路由，也就理清了 MFC 体系结构的一个轮廓。本书第 2 章、第 3 章、第 4 章重点阐述这两方面内容。

考虑到文档\视图框架结构在 MFC 程序设计中的重要地位，第 6 章在学习 MFC 源代码的基础上，分别对单文档和多文档框架结构，以及视图拆分的实现机制进行剖析和总结。如果你一直忍受着文档\视图框架的束缚，有必要认真学习这一章。

本书并不泛讲 VC++编程的入门知识，甚至有意无意地回避那些你在其他书本中很容易发现的内容。但对于 VC++编程中的几个重点难点内容以及实战技术总是全面而深入地阐述。例如窗口操作、非窗口消息路由、控件自画、停靠浮动子窗口、多框架程序设计、打印、多线程等。

由于本书当中的许多论述是以剖析 MFC 源代码为依据的，所以你在阅读时可能并不感到轻松。但这是学习 MFC 的有效途径。你可以编写程序，亲自跟踪并测试还存有疑惑的 MFC 源代码部分，以加深理解。

本书所覆盖的内容基本限于桌面程序设计，对流行的 COM\COM+编程没有专门的论述，也不包含数据库访问部分。

书中所引用的完整示例都收录在所附光盘中。这些示例代码在书中以“示例清单”标识，区别于程序段。很多示例只在书中列出部分重点代码，**完整的工程文件以及源代码可在配书光盘中获取**。本书中的示例程序都力求简明，直接实践相关的编程技术，不进行过多的包装。

最后，我要向支持并给予我大力帮助的朋友和同事们表示感谢：李木山、王颜国、王玉伟、刘凯等。并感谢刘雪洁和欧振旭编辑，与他们合作我感到非常愉快。

我想，对于海边的贝壳，如果可以闪出光亮，经过的人总会拾起它，否则，还是让它掩埋在地下的好。

期待广大读者朋友就本书内容与我共同交流，或提出您的意见和批评。

E-mail: VcBook_TechAnal@sina.com

辛长安

2002 年 1 月

目 录

第 1 章 C++关键语法及其在 VC++中的应用	1
1.1 重载	2
1.1.1 函数重载.....	2
1.1.2 运算符重载.....	7
1.1.3 函数重载在 MFC 中的应用举例.....	16
1.1.4 运算符重载在 MFC 中的应用举例.....	17
1.2 虚拟函数.....	19
1.2.1 静态联编与动态联编.....	19
1.2.2 虚拟函数的定义.....	21
1.2.3 虚拟函数的实现机制.....	23
1.2.4 虚拟函数的应用.....	28
1.2.5 纯虚拟函数.....	41
1.2.6 虚拟函数在 MFC 中的应用举例.....	43
1.3 静态成员.....	44
1.3.1 静态成员变量.....	44
1.3.2 静态成员函数.....	47
1.3.3 静态成员变量在 MFC 中的应用举例.....	49
1.3.4 静态成员函数在 MFC 中的应用举例.....	49
1.4 类模板	50
1.4.1 类模板的定义.....	50
1.4.2 使用类模板和模板类.....	60
1.4.3 模板在 MFC 中的应用举例	64
1.5 多重继承和内嵌类.....	66
1.5.1 继承方式与访问权限.....	66
1.5.2 多重继承和虚拟基类.....	70
1.5.3 内嵌类与类合成.....	76
1.5.4 类继承与类合成的应用.....	90
1.5.5 多重继承和内嵌类在 COM 中的应用	95
第 2 章 MFC 类结构与窗口操作	101
2.1 MFC 类结构	102
2.1.1 CObject 类	103
2.1.2 CCmdTarget 类.....	112

2.1.3	CWinThread 类	114
2.1.4	CWnd 类	116
2.2	CWnd 类与 Windows 窗口的关系	119
2.2.1	使用 WIN32 API 创建窗口	120
2.2.2	亲自动手创建窗口封装类	123
2.2.3	CWnd 类如何封装 Windows 窗口	138
2.3	CWnd 的派生类	143
2.3.1	CFrameWnd 类	143
2.3.2	CView 类	148
2.3.3	CDialog 类	152
2.4	窗口操作	159
2.4.1	检索窗口	159
2.4.2	屏幕坐标与客户区坐标	163
2.4.3	窗口之间的层次关系	164
2.4.4	父窗口与子窗口	166
2.5	Windows 窗口类	172
2.5.1	窗口类的结构	173
2.5.2	系统定义的窗口类	175
2.5.3	窗口的子类化和超类化	177
第 3 章	消息映射与消息处理	181
3.1	MFC 的消息映射	182
3.1.1	消息映射机制	182
3.1.2	消息映射的宏定义	184
3.2	非窗口消息	187
3.2.1	命令消息	187
3.2.2	通知消息	188
3.2.3	反射消息	191
3.2.4	非窗口消息的传递路由	192
3.2.5	非窗口消息的扩展	206
3.3	特殊消息和处理函数	209
3.3.1	空闲消息处理	209
3.3.2	命令状态更新消息	212
3.3.3	使用 OnCmdMsg() 函数分发非窗口消息	221
第 4 章	控件子窗口	225
4.1	控件的创建和子类化	226
4.1.1	控件的创建	226

4.1.2 控件的子类化.....	231
4.2 控件的属主画与自定义画.....	233
4.2.1 属主画消息处理和虚拟函数.....	234
4.2.2 几个控件的属主画特性.....	238
4.2.3 控件的自定义画.....	249
4.3 自定义控件.....	252
4.3.1 自定义控件的窗口类.....	253
4.3.2 自定义控件的通知消息.....	255
4.3.3 自定义控件的绘制.....	256
4.3.4 自定义的分隔条控件.....	257
第 5 章 界面优化.....	267
5.1 开发使用控制条.....	268
5.1.1 为控制条按需分配客户区.....	268
5.1.2 控制条基类 CControlBar.....	272
5.1.3 控制条的停靠与浮动.....	279
5.1.4 实现停靠浮动子窗口.....	285
5.2 工具栏优化.....	296
5.2.1 添加按钮文本.....	296
5.2.2 创建工具栏的子控件.....	298
5.3 菜单优化.....	303
5.3.1 动态创建菜单.....	303
5.3.2 菜单的属主画.....	309
第 6 章 文档视图框架.....	321
6.1 CWinApp 应用类.....	322
6.1.1 应用类全局对象.....	322
6.1.2 注册表和 INI 文件操作.....	323
6.1.3 命令行参数处理.....	324
6.2 单文档模板框架.....	329
6.2.1 文档、框架、视图的动态创建.....	329
6.2.2 非拆分视图的切换.....	335
6.3 多文档模板框架.....	337
6.3.1 CMDIFrameWnd 主框架.....	337
6.3.2 CMDIChildWnd 子框架.....	341
6.3.3 CDocument 类的文档管理功能.....	344
6.4 编写多框架的应用程序.....	352
6.4.1 创建多框架的必要性.....	353

6.4.2	自动创建的多框架程序	353
6.4.3	改进自动创建的多框架程序	359
6.4.4	手工创建多框架程序	364
6.5	拆分视图	370
6.5.1	认识 CsplitterWnd 窗口拆分类	370
6.5.2	应用 Cview::OnCreate()消息处理函数实现拆分	374
6.5.3	创建非视图的拆分子窗口	376
6.5.4	拆分视图的创建删除和隐藏显示	378
第 7 章	屏幕绘图与打印	385
7.1	MFC 设备环境类	386
7.1.1	基类 CDC	386
7.1.2	CWindowDC 与 CClientDC	389
7.1.3	WM_PAINT 窗口消息与 CPaintDC 类	392
7.1.4	使用设备上下文的剪裁区提高刷新效率	395
7.2	脱离视图的 MFC 打印功能	405
7.2.1	准备打印设备和打印参数	405
7.2.2	单页打印	408
7.2.3	分页打印	409
7.2.4	调整图文打印尺寸	418
7.3	窗口的自动打印	421
7.3.1	自动打印的消息处理	421
7.3.2	自动打印客户区和非客户区	422
第 8 章	多线程的创建与控制	429
8.1	工作者线程和界面线程	430
8.1.1	创建工作者线程	430
8.1.2	创建用户界面线程	433
8.1.3	线程和进程的优先级	445
8.1.4	线程的终止	447
8.2	线程间的通信	448
8.2.1	线程间的互斥	448
8.2.2	线程间的同步	453
8.2.3	线程间的资源共享	457
第 9 章	操作 Web 网页	469
9.1	DHTML 对象模型	470
9.1.1	了解 HTML 超文本标记语言	470

9.1.2	DHTML 的定义	471
9.1.3	了解 JavaScript 和 VBScript 脚本语言	474
9.2	将 Web 浏览器嵌入应用程序	475
9.2.1	使用 MSIE ActiveX 控件	475
9.2.2	应用 CHtmlView 视图类	477
9.3	操作 DHTML 对象和网页元素	479
9.3.1	操作单框架网页元素	480
9.3.2	操作多框架网页元素	486
第 10 章	异常处理	495
10.1	异常处理的种类与应用	496
10.1.1	C++ 异常处理	496
10.1.2	结构化异常处理	500
10.1.3	异常处理中的资源释放	501
10.1.4	异常处理的选择使用	505
10.2	MFC 的异常处理	506
10.2.1	MFC 的异常类 CException	506
10.2.2	MFC 的 CException 派生类	508
10.2.3	自定义 CException 的派生类	512

C++ 关键语法及其在 VC++ 中的应用

- 重载
- 虚拟函数
- 静态成员
- 类模板
- 多重继承和内嵌类

不同于其他可视化编程工具的是，使用 VC++ 编程，在很大程度上要依赖于对 C/C++ 语法的熟练应用。如果不知道重载和虚拟，就没有掌握 C++ 语言的精华，学习 MFC 也就无从谈起。VC++ 提供给程序员的是猎枪，而不是有限的食物，但要求你是一个出色的猎手。

本章将详细阐述 C++ 语法的关键部分，并对这些语法现象在 MFC（包括 COM）中的应用进行剖析和总结。相信这是学习 MFC 的第一步。

1.1 重 载

为减轻程序设计者记忆函数名称的负担，增强程序代码的可读性，C++ 编译器允许在相同作用域内（全局或某类中）定义两个以上的同名函数，即函数重载。同时，编译器也允许对大部分常规运算符（例如 +、-、*、=）重新定义，赋予它们新的操作，即运算符重载。对于重载函数，使用者可以仅知道其一般含义，不必知道他们的实现细节。因此，重载提高了程序的抽象程度，对程序的编写和阅读都有优化作用。

1.1.1 函数重载

在实际编程中，往往将一组功能非常相近的函数定义为重载函数。例如要设计三个函数分别用来输出三种提示信息，采用 C 语法的声明代码可能如下：

```
void ShowMessageA(const char* Text);  
void ShowMessageB(const char* Text, unsigned int Type);  
void ShowMessageC(const char* Text, const char* Caption);
```

而如果采用 C++ 语法，可以这样声明：

```
void ShowMessage(const char* Text);  
void ShowMessage(const char* Text, unsigned int Type);  
void ShowMessage(const char* Text, const char* Caption);
```

这样，只需记住 ShowMessage()，即可输出各种提示信息；而代码的读者也不会被各种功能和名称相近的函数搞得晕头转向。

一组重载函数是以参数类型或参数个数加以区别的。例如上面的一组 ShowMessage() 重载函数，第一个与后两个参数个数不同，而后两个参数类型不同。**函数的返回值对区别重载函数没有意义**。每个重载函数可以有不同的返回值类型，如下面一组函数仍旧是重载函数：

```
bool ShowMessage(const char* Text);  
int ShowMessage(const char* Text, unsigned int Type);  
void ShowMessage(const char* Text, const char* Caption);
```

显而易见，下面这组函数不是重载函数：

```
bool ShowMessage(const char* Text);  
void ShowMessage(const char* Text);
```

如果它们在同一作用域内定义，将会产生编译错误。

在全局和类的范围内都可以定义重载函数。示例清单 1.1.1 和示例清单 1.1.2 分别演示了在全局范围内和类的范围内定义重载函数，两个程序的输出结果相同。

示例清单 1.1.1

```
#include "stdio.h"  
//在全局范围内定义两个 ShowMessage 重载函数  
void ShowMessage(const char* Text ,const char* Caption)  
{  
    printf("Message: Text=%s,Caption=%s\n",Text,Caption);  
}  
  
void ShowMessage(const char* Text ,unsigned int Type)  
{  
    printf("Message: Text=%s,Type=%d\n",Text,Type);  
}  
  
int main()  
{  
    //依次调用两个重载函数  
    ShowMessage("ok","welcom");  
    ShowMessage("ok",1);  
  
    return 0;  
}
```

示例清单 1.1.2

```
#include "stdio.h"  
class CMessage  
{  
public:  
    CMessage(){};  
    //在类中定义两个内联的重载函数 ShowMessage  
    void ShowMessage(const char* Text ,const char* Caption)  
    {  
        printf("Message: Text=%s,Caption=%s\n",Text,Caption);  
    }  
}
```

```

void ShowMessage(const char* Text ,unsigned int Type)
{
    printf("Message: Text=%s,Type=%d\n",Text,Type);
}

};

int main()
{
    CMessage LoginMessage;
    //依次调用对象 LoginMessage 的两个重载的成员函数
    LoginMessage.ShowMessage("ok","welcom");
    LoginMessage.ShowMessage("ok",1);

    return 0;
}

```

以上两程序的输出结果如下:

```

Message: Text=ok,Caption=welcom
Message: Text=ok,Type=1

```

通过以上学习,可以得出这样的结论:编译器依据传递的实参类型和个数确定重载函数的调用。那么,对于重载函数:

```

void ShowMessage(const char* Text , int Type)
{
    printf("Message: Text=%s,Type=%d\n",Text,Type+1);
}

void ShowMessage(const char* Text ,unsigned int Type)
{
    printf("Message: Text=%s,Type=%d\n",Text,Type);
}

```

调用程序段:

```

{
    unsigned char i=0; ShowMessage("ok",i);
}

```

会输出什么结果呢? 答案如下:

```

Message: Text=ok,Type=1

```

因为对参数个数相同的重载函数,进行调用时,编译器依次使用下面的规则将实参与形参进行匹配,确定重载函数的调用:

(1) 具有与实参类型相同的形参的重载函数。

(2) 转换实参，将 T 转换为 T&，或 T&转换为 T，或 T 转换为 const T，然后寻找形参类型匹配的重载函数（T 为某数据类型）。

(3) 实参进行标准的类型转换，如 char→int→unsigned int，然后寻找形参类型匹配的重载函数。

(4) 使用构造函数转换实参，使其与某一重载函数匹配。例如，如果类 A 有构造函数 A(int)，那么 int 型的实参可以被编译器转换为类 A 的临时对象。

(5) 使用类型转换函数进行转换，使其与某一重载函数匹配。

显然，上述例子符合第 3 种情况。又如对于重载函数：

```
void ShowMessage(const char* Text ,const char* Caption)
{
    printf("Message: Text=%s,Caption=%s\n",Text,Caption);
}
```

```
void ShowMessage(const char* Text ,unsigned int Type)
{
    printf("Message: Text=%s,Type=%d\n",Text,Type);
}
```

调用语句

```
ShowMessage("ok",0);
```

会产生编译错误，因为编译器无法判断第二个值参是空字符串 NULL，还是整数 0。可见，重载函数的定义和调用会涉及类型转换的细节问题，在设计时要考虑全面。

前面介绍过，重载函数可以定义在全局或某个类中，其实不仅如此，重载还可以发生在父类和子类中。请看示例清单 1.1.3。

示例清单 1.1.3

```
#include "stdio.h"
#include "string.h"
class CMessage
{
public:
    CMessage(){};
    void ShowMessage(const char* Text ,const char* Caption)
    {
        printf("Message: Text=%s. Caption=%s\n",Text,Caption);
    }
    void ShowMessage(const char* Text ,unsigned int Type)
    {
        printf("Message: Text=%s. Type=%d\n",Text,Type);
    }
}
```

```
    }
    void ShowMessage(const char* Text)
    {
        printf("Message: Text=%s\n",Text);
    }

};

class COKMessage :public CMessage
{
public:
    COKMessage(){};
    void ShowMessage(const char* Text ,const char* Caption)
    {
        char TextForOK[40];
        strcpy(TextForOK,"OK,");
        strcat(TextForOK,Text);
        CMessage::ShowMessage( TextForOK , Caption);
    }
    void ShowMessage(const char* Text ,unsigned int Type)
    {
        char TextForOK[40];
        strcpy(TextForOK,"OK,");
        strcat(TextForOK,Text);
        CMessage::ShowMessage(TextForOK , Type);
    }
    void ShowMessage(const char* Text )
    {
        char TextForOK[40];
        strcpy(TextForOK,"OK,");
        strcat(TextForOK,Text);
        CMessage::ShowMessage(TextForOK);
    }
};

int main()
{
    COKMessage OKMessage;
    OKMessage.ShowMessage("Welcom");
    OKMessage.CMessage::ShowMessage("Welcom");
}
```

```
    return 0;  
}
```

程序的输出结果:

```
Message: Text=OK,Welcom  
Message: Text=Welcom
```

子类与父类中存在同名的成员函数,称之为重载。在上例中,子类与父类的三个成员函数不仅名称相同,声明也完全一致。这种父子声明一致的重载,不可能通过参数类型或数量确定调用函数,而是通过在父类的成员函数前加 T::进行区分(其中 T 为父类名),如上例的 CMessage::ShowMessage("Welcom")。

1.1.2 运算符重载

1. 运算符重载的定义

C++提供了许多库函数,如字符串操作库函数、数学运算库函数、图形操作库函数等。同时也提供了许多标准运算符,如+、-、*、/、&等。这些运算符就像 C++的库函数一样,实现一定的功能,是开发程序的基本工具。

对于库函数,通过上节的学习,已经能够对其进行重载。示例清单 1.1.4 是重载库函数 strcat()的示例,实现字符串连接功能。

示例清单 1.1.4

```
#include "stdio.h"  
#include "string.h"  
  
//char *strcat( char *strDestination, const char *strSource ) 是C++声明的  
//库函数原型  
  
//下面重载 strcat()  
/*****  
功能:          连接两个字符串  
strDestination: 目标字符串  
strSource:      源字符串  
DestLength:     限制目标字符串的最大长度,避免目标数组越界  
返回值:        目标字符串的最后长度  
*****/  
int  strcat(char *strDestination, const char *strSource,unsigned int  
DestLength)  
{  
    //如果目标字符串数组已经填满,不进行连接,返回当前长度
```