Borland C ++3.0 技术丛书

BORLAND C++ 3.0

工具與實用程序指南

Borland C++ 3.0 技术丛书

Borland C++ 3.0
工具与实用程序指南

前　言

Borland C++ 3.0 是目前流行且深受用户青睐的高级语言程序设计环境 Borland C++ 2.0 的升级版本。本套丛书是 Borland C++ 3.0 完整的使用手册与技术资料，它全面而系统地介绍了 Borland C++的基础知识和高级技术，主要包括如下几方面的内容：

- C++语言参考及语言接口；
- 程序设计软件包；
- 程序设计环境；
- 各种实用工具，如调试器、Help系统编译器、资源编辑及编译器、工程管理工具等；
- 面向对象的程序设计；
- Windows 程序设计；
- 错误信息。

整套丛书包括：用户指南,程序员指南,工具与实用程序指南,库参考指南,Turbo Debugger,Turbo Vision,Turbo Profiler,Turbo Assembler,资源开发工具,Windows 环境下的面向对象程序设计。

本套丛书对 C++语言的初学者和熟练的 C++程序员而言都是一套很实用的技术指导丛书。另外提醒读者注意的是，如果希望用 Borland C++ 3.0 开发 Windows 应用程序，最好将这套丛书与 Windows 技书丛书配套使用。

简　介

Borland C++带有许多功能很强的可独立使用的实用程序，这些实用程序可以和 BorLand C++文件或其它模块一起使用，以方便于 DOS 或 Windows 程序设计。

本章介绍了 IMPDEF、IMPLIB、IMPLIBW、MAKE、TLIB、TLINK 及 WinSight，并以代码和命令行示例的形式描述了如何使用它们。Borland C++的其它实用程序文档在文本文件 UTIL.DOC 中，此文件由 INSTALL 实用程序放置在 DOC 子目录下。

本书将要介绍的实用程序：

IMPDEF	创建模块定义文件的实用程序。
IMPLIB	生成输入库的实用程序。
IMPLIBW	生成输入库的Windows应用程序
MAKE	单独运行的程序制作实用程序
TLIB	Turbo库管理程序
TLINK	Turbo连接程序
WinSight	Windows消息监视程序

联机 UTIL.DOC 文档中介绍的实用程序：

BGIOBJ	图形驱动和字体转换实用程序（联机文档）
CPP	预处理程序（联机文档）
GREP	一种文件搜索实用程序（联机文档）
OBJXREF	目标模块的交叉参考（联机文档）
PRJCFG	由配置文件修改工程文件的选项或者把工程文件转化成目标文件（联机文档）
PRJCNVT	把Borland++工程文件转化成MAKE文件（联机文档）
PRJ2MAK	把Turbo C工程文件转化成Borland C++格式的工程文件（联机文档）
THELP	Trbo求助实用程序（联机文档）
TOUCH	文件日期和时间的修改程序（联机文档）
TRANCOPY	将切换项从一个工程拷贝到另一个工程（联机文档）
TRIGRAPH	字符转换实用程序（联机文档）

目　录

第一章　输入库工具 …… (1)
1.1　IMPDEF：模块定义管理程序 …… (1)
1.1.1　DLL 中的类 …… (1)
1.1.2　DLL 中的函数 …… (2)
1.2　IMPLIB：输入库 …… (2)
1.2.1　重建 IMPORT.LIB …… (3)
1.3　IMPLIBW：Windows 输入库 …… (3)
1.3.1　选择输入库 …… (4)
1.3.2　创建输入库 …… (4)

第二章　MAKE：程序管理器 …… (5)
2.1　MAKE 是怎样工作的 …… (5)
2.2　启动 MAKE …… (6)
2.2.1　命令行选项 …… (6)
2.2.2　BULTMS.MAK 文件 …… (7)
2.3　MAKE 的一种简单运用 …… (8)
2.4　创建 makefile 文件 …… (9)
2.5　makefile 文件的组成 …… (10)
2.5.1　隐式和显式规则的命令 …… (11)
2.5.2　显式规则 …… (15)
2.5.3　隐式规则 …… (17)
2.5.4　宏 …… (19)
2.5.5　指令 …… (24)
2.5.6　点指令 …… (25)
2.5.7　文件嵌入指令 …… (27)
2.5.8　条件执行指令 …… (27)
2.5.9　出错指令 …… (30)
2.5.10　取消宏定义指令 …… (30)
2.6　兼容选项-N …… (30)

第三章　TLIB：Turbo 库管理程序 …… (32)
3.1　为什么使用目标模块库 …… (32)
3.2　TLIB 命令行 …… (32)
3.2.1　操作表 …… (33)

3.3 使用应答文件 …… (34)
3.4 创建一个扩展目录：/E 选项 …… (34)
3.5 设置页大小：/P 选项 …… (35)
3.6 高级操作：/C 选项 …… (35)
3.7 例子 …… (35)

第四章 TLINK：Turbo 连接程序 …… (37)
4.1 调用 TLINK …… (37)
4.1.1 DOS 下连接的一个例子 …… (38)
4.1.2 Windows 程序连接的一个例子 …… (38)
4.1.3 TLINK 命令行中的文件名 …… (39)
4.1.4 使用应答文件 …… (39)
4.1.5 TLINK 配置文件 …… (40)
4.1.6 使用 TLINK 连接 Borland C++模块 …… (41)
4.1.7 利用 BCC 使用 TLINK …… (43)
4.2 TLINK 选项 …… (44)
4.2.1 TLINK 配置文件 …… (44)
4.2.2 /3(80386 32 位码) …… (44)
4.2.3 /A(段对齐) …… (44)
4.2.4 /c(大小写敏感) …… (45)
4.2.5 /C(大小写敏感的 exports) …… (45)
4.2.6 /d(重复符号) …… (45)
4.2.7 /e(不使用扩展目录) …… (45)
4.2.8 /i(未初始化的尾部段) …… (46)
4.2.9 /l(行号) …… (46)
4.2.10 /L(库查找路径) …… (46)
4.2.11 /m,/s 和/x(映象选项) …… (46)
4.2.12 /n(忽略缺省库) …… (48)
4.2.13 /o(覆盖) …… (48)
4.2.14 /P(压缩代码段) …… (48)
4.2.15 /t(微模式.COM 文件) …… (49)
4.2.16 /Tt 和/Tw(目标选项) …… (49)
4.2.17 /v(调试信息) …… (49)
4.2.18 /ye(扩充内存) …… (50)
4.2.19 /yx(扩展内存) …… (50)
4.3 模块定义文件 …… (50)
4.3.1 缺省块定义文件 …… (51)
4.3.2 一个例子 …… (51)
4.4 模块定义引用 …… (52)

4.4.1 CODE(代码) …… (52)
4.4.2 DATA(数据) …… (53)
4.4.3 DESCRIPTION(描述) …… (53)
4.4.4 EXETYPE …… (53)
4.4.5 EXEPORTS …… (53)
4.4.6 HEAPSIZE …… (54)
4.4.7 IMPORTS …… (54)
4.4.8 LIBARY(库) …… (55)
4.4.9 NAME(名) …… (55)
4.4.10 SEGMENTS …… (56)
4.4.11 STACKSIZE …… (56)
4.4.12 STUB …… (56)

第五章 使用 WinSight …… (58)
5.1 入门 …… (58)
5.2 选择一个视图 …… (59)
5.2.1 挑选一个视窗 …… (59)
5.2.2 安排视窗 …… (59)
5.2.3 获取更多细节 …… (59)
5.3 使用窗口树 …… (59)
5.3.1 裁剪树 …… (60)
5.3.2 找到一个窗口 …… (60)
5.3.3 对窗口进行跟踪 …… (60)
5.4 类操作 …… (61)
5.4.1 使用 Class List 视窗 …… (61)
5.4.2 对类的跟踪 …… (61)
5.5 开关操作 …… (61)
5.5.1 关闭跟踪 …… (61)
5.5.2 挂起屏幕更新 …… (61)
5.6 选择要跟踪的消息 …… (61)
5.6.1 滤出消息 …… (62)
5.6.2 消息跟踪选项 …… (62)
5.7 WinSight 窗口 …… (67)
5.7.1 Class List 视窗 …… (67)
5.7.2 Windows Tree 视窗 …… (67)
5.7.3 Message Trace 视窗 …… (68)

第六章 RC：Windows 资源编译器 …… (69)
6.1 创建资源 …… (69)

6.2 添加资源到可执行文件中 …… (69)
6.2.1 从 IDE 中编译资源 …… (70)
6.2.2 从命令行编译资源 …… (70)
6.2.3 从制作文件中编译资源 …… (70)
6.3 资源编译器句法 …… (70)

第七章 HG: Windows Help 编译器 …… (72)
7.1 创建 Help 系统: 开发周期 …… (72)
7.1.1 如何将 Help 显示给用户 …… (73)
7.1.2 Help 是如何展示给 Help 作者的 …… (73)
7.1.3 Help 是如何展示给 Help 程序员的 …… (73)
7.2 设计 Help 系统 …… (74)
7.2.1 拟定开发计划 …… (74)
7.2.2 决定标题文件结构 …… (77)
7.2.3 设计 Help 标题的外观 …… (78)
7.2.4 图形图象 …… (80)
7.3 建立 Help 标题文件 …… (80)
7.3.1 选择一个编著工具 …… (81)
7.3.2 建立 Help 标题文件 …… (81)
7.3.3 给 Help 标题文件编码 …… (81)
7.3.4 插入图形图象 …… (88)
7.3.5 管理主题文件 …… (89)
7.4 建立 Help 文件 …… (93)
7.4.1 建立 Help 项目文件 …… (93)
7.4.2 指定项目文件 …… (94)
7.4.3 指定创建标记 …… (94)
7.4.4 指定选择项 …… (95)
7.4.5 指定备用上下文字符串 …… (100)
7.4.6 上下文关联标题映象 …… (101)
7.4.7 引用点位图 …… (102)
7.4.8 编译 Help 文件 …… (102)
7.4.9 编制能访问 Help 的应用程序 …… (103)
7.5 Help 示例 …… (109)
7.5.1 Helpex 项目文件 …… (111)

附录 错误信息 …… (112)
A-1 错误信息分类 …… (112)
A-2 信息解释 …… (116)

第一章　输入库工具

动态连接库（DLL）是 Windows 程序设计的重要组成部分。用户可以创建由常用代码组成的 DLL，也可以在应用程序中使用 DLL。

在建立 Windows 应用程序时，TLINK 使用输入库获知某个函数是否被定义或是否可由 DLL 输入进来。输入库通常取代模块定义文件。

IMPDEF 创建模块定义文件，且为 DLL 中的每个输出函数包含一条 EXPORT 语句。IMPLIBW 为 DLL 创建输入库，但它是非一个 Windows 应用程序。

1.1 IMPDEF:模块定义管理程序

IMPDEF 和 IMPLIB 实用程序用来定制满足特定应用程序需要的输入库。IMPDEF 的句法是:

IMPDEF DestName. DEF SourceName. DLL 由 SourceName. DLL 创建名叫 DestName.DEF 的模块定义文件。模块定义文件的形式为：

```
LIBRARY      FileName
DESCRIPTION 'Description'
EXPORTS
        ExportFuncName          @Ordinal

        ExportFuncName          @Ordinal
```

这里，FileName 为动态连接库的文件名，如果动态连接库先前已连接了包含 DESCRIPTION 语句的模块定义文件，那么 Description 为该 DESCRIPTION 语句的值，ExportFuncName 命名一个输出函数，Ordinal 是该函数的顺序值（是一个整数）。

1.1.1 DLL 中的类

该实用程序对使用 C++类的动态连接库尤其方便。有两个原因:首先，如果定义类时使用了__export 关键字，则该类中所有非内部成员函数和静态数据域将被输出。又因为 IMPDEF 程序列出了所有输出函数，并自动地包含了成员函数和静态数据域，因而就很容易生成一个模块定义文件。

因为这些输出函数的名称是可改变的，所以欲根据模块定义文件创建一个输入库，就必须在模块定义文件的 EXPORTS 部分乏味地列出所有名字。如果使用 IMPDEF 实用程序创建模块定义文件，它将包含每一个输出函数的顺序值，同时如果输出函数名被修改，最初的输出函数名称必须标注在函数入口的后面。例如，由 IMPDEF 使用 C++类为动态连接库创建的模块定义文件是:

```
LIBRARY      FileName
DESCRIPTION 'Description'
EXPORTS
        MangledExportFuncName @Ordinal;  ExportFuncName
        ...
        MangledExportFuncName @Ordinal;  ExportFuncName
```

在这里，FileName 是动态连接库的文件名，如果动态连接库先前已连接了包含有 DESCRIPTION 语句的模块定义文件，Description 就是该语句的值，MangledExportFuncName 是修改后输出函数的名称，Ordinal 是输出函数的顺序值（一个整数）。ExportFuncName 是输出函数的初始名称。

1.1.2 DLL 中的函数

用 IMPDEF 可建立一个列出动态连接库里的所有输出函数的可编辑的源文件。也可以编辑这种 DEF 文件，使之只包含特定应用程序所需要的那些函数，然后对该编辑过的 DEF 文件调用 IMPLIB。这样可生成一个包含动态连接库输出函数特定子集的输入信息的输入库。

例如，假定正在配置一个动态连接库，在该动态连接库中提供了许多应用程序所需要使用的函数，动态连接库中的每一个输出函数都是用__export 标识的。现在，如果应用程序需要使用动态连接库中所有输出函数，对动态连接库简单地使用 IMPLIB 实用程序，就可生成一个输入库，将动态连接库中的输出函数传送给输入库。该输入库包含了动态连接库中所有输出函数的输入信息，并能同任何一个应用程序相连接，这样就不必在每个应用程序的模块定义文件的 IMPORTS 部分中列出其用到的所有动态连接库函数了。

现在，假定你只要向某个应用程序提供动态连接库的一部分输出函数，从概念上说，应该裁剪要连接到该应用程序的输入库，使该输入库只包含该应用程序所需要的那些输出函数的输入信息，该应用程序不可以调用输入库之外的动态连接库中的其它输出函数。

要建立一个满足特定条件的输入库，首先，需要在已编译好且连接过的动态连接库上运行 IMPDEF 实用程序，生成一个模块定义文件，该文件包含有 EXPORTS 部分，且含有动态连接库中的所有输出函数。接着，用户可以编辑该模块定义文件，移走与将要定制的输入库里所不需要的某些输出函数相对应的 EXPORTS 部分的入口项。然后，在模块定义文件上运行 IMPLIB 实用程序，生成一个输入库。该输入库包含有模块定义文件的 EXPORTS 部分所列出的那些输出函数的输入信息。

1.2 IMPLIB:输入库

IMPLIB 实用程序用来创建一个输入库，该输入库包含有 Windows 应用程序模块定义文件 IMPORTS 节中的部分或全部输出函数。

如果一个模块要使用动态连接库中的输出函数，有两种方法可以向连接程序报告这种信息。

- 在模块定义文件上增加一个IMPORTS部分并列出此模块将要使用的动态连接库

中的所有输出函数。

- 或者在连接该模块时，连接输入库。
- 如果已建立了一个Windows应用程序，则至少使用了一个输入库IMPORT.LIB。IMPORT.LIB 是标准 Windows 动态连接库的输入库（当在集成环境下建立 Windows 应用程序时，IMPORT.LIB 自动被连接）。
- 一个输入库能列出一个或多个动态连接库的部分或所有输出函数。IMPLIB可直接从动态连接库或从动态连接库的模块定义文件建立输入库（或者结合使用这两种方法）。

要为动态连接库建立一个输入库，需键入:

```
IMPLIB Options LibName [DefFiles...|DLLs...]
```

在这里，Options 是可选的一个或多个 IMPLIB 选项，LibName（如果需要）是新的输入库名称，DefFiles 是在一个或多个动态连接库上的一个或多个已存在的模块定义文件名，DLLs 是一个或多个已存在的动态连接库名。用户至少必须定义一个动态连接库或者一个模块定义文件。

表 1.1 IMPLIB 选择项

选 项	含 义
-i	告诉IMPLIB忽略WEP（用以结束一个DLL的Windows退出过程）。如果在IMPLIB 命令行上定义了多个动态连接库，请选择该选项。
警告控制:	
-t	简短警告
-v	冗长警告
-W	不警告

1.2.1 重建 IMPORT.LIB

当 Miorosoft 公司提供新版本的 Windows 时，你可能需要用新的 IMPORT.LIB 替换以前版本的 IMPORT.LIB。最简单的方法是重建 IMPORT.LIB。

建立新的 IMPORT.LIB 的命令行是:

```
IMPLIB -i  IMPORT.LIB GDI.EXE KERNEL.EXE  USER.EXE
           KEYBOARD.DRV SOUND.DRV WIN87EM.DLL
```

如果 Windows 使用了其它动态连接库，那么任何新的动态连接库都须在命令行上列出。

1.3 IMPLIBW:Windows 输入库

Windows 实用程序 IMPLIBW 可以创建一个输入库，用以取代 Windows 应用程序

的模块定义文件中 IMPORTS 节的部分或全部内容。与本书讨论的其它大多数实用程序不同，IMPLIBW 必须运行于 Windows 环境下。

1.3.1 选择输入库

欲用 IMPLIBW 创建输入库，在 IMPLIBW 图标上点两下鼠标按钮。出现一个空窗口。选择 File|Create...将引出一个对话框，列出当前目录下的所有 DLL。可在列表框的目录名上按鼠标键两次，从而进入不同的目录。

1. 从 DLL 选择

选择某个 DLL 并按下 Create 按钮时，IMPLIBW 将创建一个输入库，其名称与 DLL 相同，其扩展名为.LIB。

2. 从模块定义文件中选择

如果有 DLL 的模块定义文件（.DEF），则可用它来代替 DLL 本身创建输入库。不用选择 DLL，而是在编辑控制中打入.DEF 文件名并按 Create 按钮。IMPLIBW 将创建一个输入库，其名称与所选择的.DEF 文件相同，其扩展名为.LIB。

1.3.2 创建输入库

如果 IMPLIBW 在检查 DLL 或模块定义文件并创建输入库的过程中没有发现错误，窗口中将出现消息“No Warnings.” 这样就创建了一个新的输入库，用户可以把它包含到某个工程中去。

第二章 MAKE:程序管理器

Borland 公司的命令行 MAKE 是从使用同一名称的 UNIX 程序中派生而来的，它有助于用户维护当前程序的可执行版本。多数程序都由许多源文件组成，这些程序在同其它程序连接之前，程序中的源文件可能有必要通过预处理器、汇编器、编译器和其它程序的处理。当已修改过模块或与模块相关的某些程序后，如果忘记了再编译它们，将可能导致错误的发生。另一方面，虽然重编译一遍所有的程序是非常安全可靠的，但是却非常浪费时间。

使用 MAKE 可以解决该矛盾。MAKE 是怎样处理程序的源文件和目标文件、并生成一个完整产品的呢？MAKE 首先检查文件上的描述部分和标识日期部分，然后做一些工作，生成一个最新的文件版本。在处理过程中，MAKE 调用了许多不同的编译程序、汇编程序、连接程序和实用程序，但对已完成了而不再需要修改的程序是不做任何处理的。

MAKE 的使用远远超出了程序设计应用，用户可以使用 MAKE 控制许多处理，包括根据文件名选择文件，处理文件以生成一个完整产品。一些普通的应用包括文本处理、自动备份、在用户自己的和别人的目录下对文件进行排序，并将目录中的临时文件清除掉。

2.1 MAKE 是怎样工作的

MAKE 通过执行以下步骤来保持用户程序最新:

- 读一个名为makefile的特定文件，这个文件是用户已建好的。MAKE通过这个文件了解到哪些源文件和头文件必须被编译以生成.OBJ 文件，哪些.OBJ 文件和库文件必须被连接以生成可执行文件。
- 检查每个.OBJ的文件的日期和时间以及其所依赖的源文件和头文件的日期和时间。如果源文件和头文件中的某些文件的时间比它们的.OBJ 文件的时间近，MAKE 就知道这些文件已被修改过，需要对这些修改过的文件进行重新编译。
- 调用编译器对源文件重新编译。
- 一旦所有.OBJ文件所依赖的文件都检查完毕，再检查每一个.OBJ文件的日期和时间以及相应的可执行文件的日期和时间。
- 如果有.OBJ文件比.EXE文件要时间近，就要调用连接程序重建.EXE文件。

MAKE 完全依赖放在每个文件上的 DOS 时间标记。这就意谓着为了 MAKE 能正确地检查时间，计算机的系统日期和时间必须设置正确。必须确信电池有充足的电量。微弱的电量可能导致系统时钟不能正常运行，日期与时间不正确，MAKE 就不能正常工作了。

2.2 启动 MAKE

有两种 MAKE 版本:保护模式下的 MAKE.EXE 和实模式下的 MAKER.EXE，它们的作用相同。唯一的不同之处在于，保护模式下的 MAKE 可以处理更大的制作文件。以下当提及 MAKE 时，指这两种版本之一。

欲使用 MAKE，请在 DOS 提示符下键入 make，MAKE 将查找一名为 MAKEFILE 的文件，如果 MAKE 没有发现该文件，它将继续查找 MAKEFILE.MAK 文件；如果还未找到 MAKEFILE.MAK 或 BUILTINS.MAK 文件（以后将讲到），MAKE 就停止执行，并显示出错信息。

如果想不使用 MAKEFILE 或 MAKEFILE.MAK 文件名而用别的文件名怎么办？可在 MAKE 命令后面给出 file（-f）选项，即:

```
MAKE  -f  MYFILE.MAK
```

MAKE 的一般句法是:

```
make [option...] [target...]
```

这里 option 是 MAKE 的选项（以后讨论），target 是 MAKE 的目标文件名。

以下是 MAKE 的语法规则:

- make后面需空一格，接着是make的选项。
- 相邻的选项之间必须空一格。且选项可按任何次序排列。可以键入许多选项（只要命令行有空间）。不为字符串的所有选项后面可有一选择符“-”或“+”。这决定是关闭此选项（-）还是打开此选项（+）。
- MAKE的所有选项结束后，空一格才是可选择的目标文件名。
- 相邻的目标文件名之间必须空一格。MAKE按所列的顺序来整理目标文件，有必要的话重新编制它们的组成。

如果命令不包括任何目标文件名，MAKE 将使用显式规则中提出的第一个目标文件名。如果在命令行中有一个或多个目标文件名，MAKE 将有必要创建它们。

2.2.1 命令行选项

下面是一张 MAKE 命令行选项的完整列表。注意，大小写是有特殊意义的，选项-d 不能代替选项-D。

表 2.1 MAKE 选项

选 项	用 途
-? 或-h	列出帮助信息。
-a	对.OBJ文件进行自身依赖性检查。
-B	建立所有目标文件，不考虑文件的日期。
-ddirectory	使用-S选项时，告诉MAKE交换文件写到指定的目录中。directory中可含有驱动器标识符。对保护模式下的 MAKE 无影响。

续 表

选 项	用 途
-Didentifier	将命名的标识符定义成只包含一个字符的字符串。
[-Diden=String	定义命名的标识符iden为等号后面的字符串。字符串不包含任何空格或tab键。
-e	重定义其名字与某环境变量相同的宏时，忽略此定义。
-ffilename	使用filename文件做为MAKE文件。如果filename文件不存在且也没给出扩展名文件，试试 FILENAME.MAK 文件。
-i	忽略所有运行程序的退出状态，这与在MAKEFILE文件的所有命令前放"-"意思相等价。
-Idirectory	在给定的目录下查找include文件（以及在当前目录下）。
-K	保留（不删除）MAKE创建的临时文件，所有临时文件名的格式是MAKEnnnn.$$$$ Pnnnn 从 0000 到 9999。
-m	显示MAKE所处理的文件的日期和时间。
-n	打印这些命令，但不执行它们。这对调试makefile程序是有用的。
-N	消除MAKE与Microsoft公司提供的NMAKE间的冲突，提高MAKE的兼容性。
-p	显示所有宏定义、隐式规则及执行制作文件前的宏定义。
-r	忽略BUILTINS.MAK文件中的规则。
-s	执行前不打印令。
-S	当执行命令时，将MAKE从内存中调换出来。这样显著地减少了MAKE的内存开销，内存可编译很大的模块。
-Uidentifier	取消先前的命名标识符的定义。
-W	写当前定义的非字符串选项（象-s和-a）到MAKE.EXE文件里。

2.2.2 BULTIMS.MAK 文件

经常可以发现，有许多 MAKE 宏和规则反复被使用，有三种方法可以处理它们。

- 第一，可以将它放在所创建的每个制作文件里。
- 第二，可以将它们放在一个文件里，在所创建的每个makefile 文件里使用!include 指令将该文件包括进 makefil e 文件里。
- 第三，可以将它们放在一个BUILTINS.MAK文件里。

每次运行 MAKE 时，它都要查找 BUILTINS.MAK 文件。无论如何，没有必要同时保存几个 BUILTINS.MAK 文件。如果 MAKE 找到了一个 BUILTINS.MAK 文件，它首先解释该文件，如果它找不到一个 BUILTINS.MAK 文件，它就直接开始解释 MAKEFILE 文件（或者用户定义的 makefile 文件）。

MAKE 首先在当前目录下查找 BUILTINS.MAK 文件。如果没有找到，MAKE 将到 MAKE.EXE 被唤醒的目录下寻找该文件。所以，应该将 BUILTINS.MAK 和 MAKE.EXE 放在同一目录下。

MAKE 总是在当前目录下寻找 makefile 文件，该文件中包含有建立某一可执行程序

文件的规则。BUILTINS.MAK 和 makefile 文件有同样的语法规则。

MAKE 在当前目录下还查找所有!include 文件。如果使用了-I 选项，它将到用-I 选项定义的目录下查找。

2.3 MAKE 的一种简单运用

作为第一个例子，让我们来看看不涉及程序设计的 MAKE 的一种简单运用。假定你正在写一本书，决定将手稿的每章保留在不同的文件中（为了清楚地说明我们的主题起见，我们假定这本书很短，共有三章，分别放在不同的三个文件 CHAP1.MSS、CHAP2.MSS 和 CHAP3.MSS 里）。要制作成一份最新的完整的草稿，首先每一章需输入名叫 FORM.EXE 的排版程序中，然后使用 DOS COPY 命令将各自的输出连接起来，制作一份包含完整的草稿的文件。如图 2.1:

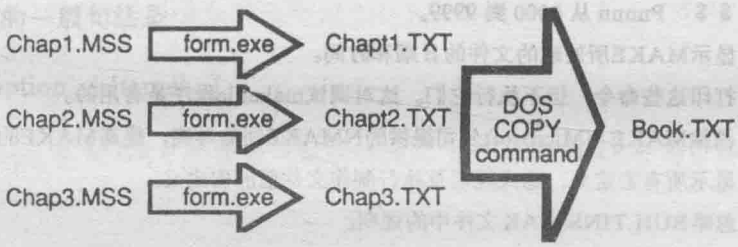

图 2.1 MAKE 的一种简单运用

正如设计程序，写一本书也需要不断修改。在写书时，可能要对一个或多个手稿文件进行修改，但在标注好你修改过的文件之前不要丢失先前的手稿文件。另一方面，不要忘记在将修改好的文件同其他文件连接之前应将该文件通过排版程序处理，否则将得不到完整的修改过的草稿文件。

解决以上问题的一种粗糙且浪费时间的方法是建一批处理文件，将每一手稿文件重新进行排版。该文件可能包括以下命令:

```
FORM CHAP1.MSS
FORM CHAP2.MSS
FORM CHAP3.MSS
COPY / A  CHAP1.TXT+CHAP2.TXT+CHAP3.TXT  BOOK.TXT
```

运行这个批处理文件总可以生成一份修改后的手稿文件。假定随着时间的推移，你的书变得越来越大，一天就有十五章，重新排版全部手稿的过程将可能是不可容忍地漫长。

MAKE 可以解决这一难题，你所要做的事仅仅是建立一个文件，常用的文件名是 MAKEFILE.MAKE。读该文件可以知道 BOOK.TXT 取决于什么文件，怎样处理这些文件。该文件还包含当 BOOK.TXT 所依赖的某些文件已经被修改后用来重建 BOOK.TXT 的规则。

对于该例子，makefile 文件的第一条规则可能是:

```
BOOK.TXT: CHAP1.TXT CHAP2.TXT CHAP3.TXT
        COPY /A CHAP1.TXT+CHAP2.TXT+CHAP3.TXT BOOK.TXT
```

这是什么意思？第一行（即 BOOK.TXT:开头的那一行）表示 BOOK.TXT 文件依赖于一、二、三章的正文格式文件。如果 BOOK.TXT 所依赖的任何文件比 BOOK.TXT 文件新，那么 MAKE 执行下一行的 COPY 命令重新生成新的 BOOK.TXT 文件。

尽管如此，该规则并不能表示所有的一切。每一章文件又依赖于每章的.MSS 的手稿文件。如果任何一个 CHAP?.TXT 文件比相对应的.MSS 文件早，那么该 CHAP?.TXT 文件就需重新生成。因而，在 makefile 文件里需要增加更多的规则:

```
CHAP1.TXT:CHAP1.MSS
        FORM CHAP1.MSS
CHAP2.TXT:CHAP2.MSS
        FORM CHAP2.MSS
CHAP3.TXT:CHAP3.MSS
        FORM CHAP3.MSS
```

这些规则表示 CHAP?.TXT 文件是最初各 CHAP?.MSS 手稿文件经过排版程序处理生成而来。

MAKE 在企图修改那个文件之前必须修改该文件所依赖的那些文件。因此，如果修改了 CHAP3.MSS，MAKE 在连接 CHAP3.TXT 文件以重建 BOOK.TXT 文件之前能很灵活地将 CHAP3.MSS 重新经过排版程序整理生成新的 CHAP3.TXT。

我们可以对该简单例子进行加工改进。三条规则看起来非常相似:其实除了每个文件名的最后一个字符不同外其它都相同。而且每次添写新的一章时很容易忘记在 makefile 文件上增添一条新的规则。为了解决这些问题，可以使用 MAKE 的隐含规则，依照文件扩展名将文件由一种形式生成另一种形式。既然是这样，我们可以用一隐含规则替换三条用于不同文章名的规则:

```
.MSS.TXT:
        FORM    $*.MSS
```

事实上，该规则表明:“如果需要将.MSS 文件制作成.TXT 文件且保持它是最新的，MAKE 是怎样做到这一步的”（还必须修改第一条规则，即生成 BOOK.TXT 文件的那条规则，以使 MAKE 能将新的一章连接成输出文件。这条规则以及后面其它规则将要使用宏来实现，详见宏的详细说明。）

一旦有了这个 makefile 文件，制作一份完整的最新的草稿文件所需的所有工作只是在 DOS 提示符下键入一条命令:MAKE。

2.4 创建 makefile 文件

从一批程序文件、包含文件、头文件、目标文件等中建立一个程序的过程与你上面的