DocStrip 程序*

Frank Mittelbach Denys Duchier Johannes Braams
Marcin Woliński Mark Wooding

由 张泓知 翻译

2023 年 12 月 6 日

摘要

本文档描述了 DocStrip 程序的实现。该程序最初由 Frank Mittelbach 开发，用于配合他的 `doc.sty`，使得在 LaTeX 中可以进行文学编程。Denys Duchier 对其进行了重写，使其可以在 TeX 或 LaTeX 中运行，并允许在条件守卫中使用完整的布尔表达式，而不仅限于逗号分隔的列表。Johannes Braams 重新整合了这两个实现，对代码进行了文档化和调试。

在 1995 年 9 月，Marcin Woliński 改变了程序的许多部分，利用了 TeX 同时写入多个文件的能力，以避免重复读取源文件。版本 2.3 的性能提升付出了与一些更不常见的操作系统的兼容性代价，这些系统限制了进程可以保持打开的文件数量。这在 1996 年 9 月由 Mark Wooding 进行了修正，他的修改被 Marcin Woliński “创造性地合并”，同时在批处理文件处理、引导处理和引入“纯文本模式”方面进行了修改。之后，David Carlisle 将新版本合并到了 LaTeX 源文件中，并进行了一些其他更改，主要是使 DocStrip 在 initex 下工作，并删除了批处理文件中必须包含 `\def\batchfile{...}` 的需要。

*此文件版本号为 v2.6b，上次修订日期为 2022-09-03，文档日期为 2023-10-10。

1 介绍

1.1 DocStrip 程序的由来

当 Frank Mittelbach 创建了 doc 包时，他发明了一种将 TEX 代码和其文档结合起来的方法。从那时起，基本上可以在 TEX 中进行文学编程。

这种编写 TEX 程序的方式显然有很大的优势，特别是当程序变得比几个宏还要大时。然而，有一个缺点，即此类程序运行时间可能比预期更长，因为 TEX 是一个解释器，必须针对程序文件的每一行决定如何处理它。因此，通过删除所有注释，可以加快 TEX 程序的运行速度。

从 TEX 程序中删除注释会引入一个新问题。现在我们有两个版本的程序，它们都必须进行维护。因此，最好能够自动删除注释，而不是手工操作。因此，我们需要一个程序来从 TEX 程序中删除注释。这可以用任何高级语言来编程，但也许不是每个人都有合适的编译器来编译该程序。每个想要从 TEX 程序中删除注释的人都有 TEX。因此，DocStrip 程序完全是用 TEX 实现的。

1.2 DocStrip 程序的功能

创建 DocStrip 程序以从 TEX 程序中删除注释行 [1] 后，进行更多操作变得可行。

不仅可以删除注释，还可以根据某些条件包含代码的部分。另外，将 TEX 程序源代码分割成几个较小的文件并在后来组合成一个“可执行”文件，也是一个不错的选择。

所有这些愿望都已在 DocStrip 程序中实现。

2 如何使用 DocStrip 程序

有多种使用 DocStrip 程序的方法：

1. 使用 DocStrip 的常规方法是编写一个批处理文件，使其可以直接由 TEX 处理。批处理文件应包含下面描述的控制 DocStrip 程序的命令。这使您可以设置一个发行版，其中用户只需运行

   TEX ⟨批处理文件⟩

   就可以从发行版源文件生成可执行文件。大多数 LATEX 发行版都是以此方式打包的。要生成此类批处理文件，请在“批处理文件”中包含一条

[1] 请注意，仅删除注释行，即以单个 % 字符开头的行；所有其他注释保留在代码中。

指令，指示 TEX 读取 docstrip.tex。此类文件的开头将如下所示：

```
\input docstrip
...
```

按照惯例，批处理文件的扩展名应为 .ins。但实际上，这些天 DocStrip 实际上可以使用任何扩展名。

2. 或者，您可以指示 TEX 读取文件 docstrip.tex，然后看看会发生什么。TEX 将向您询问一些有关您要处理的文件的问题。当您回答完这些问题后，它会执行它的任务并从您的 TEX 代码中删除注释。

3 配置 DocStrip

3.1 选择输出目录

由于希望简化 LATEX 2ε 的重新安装并支持具有目录中文件数量上限的操作系统，DocStrip 现在允许安装脚本为其创建的文件指定输出目录。我们建议在此使用相对于 texmf 的 TDS（TEX 目录结构）目录名称。然而，这些名称应被视为标签，而不是目录的实际名称。它们将根据包含在名为 docstrip.cfg 的配置文件中的命令转换为实际的系统相关路径名。

配置文件在 DocStrip 开始处理任何批处理文件命令之前被读取。

如果此文件不存在，DocStrip 将使用一些默认设置，确保文件仅写入当前目录。但是通过使用此配置文件，站点维护者可以“启用”DocStrip 的功能，使文件被写入到备选目录。

\usedir 使用此宏包作者可以告知文件应安装到何处。在该声明范围内生成的所有 \file 都将写入由其参数指定的目录。例如，在 LATEX 2ε 安装中，使用以下声明：

```
\usedir{tex/latex/base}
\usedir{makeindex}
```

而标准宏包使用

```
\usedir{tex/latex/tools}
\usedir{tex/latex/babel}
```

等等。

\showdirectory 用于在消息中显示目录名称。如果某个标签未定义，它会扩展为 UNDEFINED (label is ...)，否则为目录名称。对于每个安装脚本，在启动时显示将要使用的所有目录列表并要求用户确认可能是个不错的主意。

上述宏由宏包/安装脚本作者使用。以下宏由系统管理员在配置文件 docstrip.cfg 中使用，用于描述其本地目录结构。

\BaseDirectory 此宏是管理员表达“是的，我想要使用你的目录支持”的方式。除非你的配置文件调用了此宏，否则 DocStrip 只会写入当前目录。（这意味着除非你告诉它，否则 DocStrip 不会写入随机目录，这很好。）使用此宏，您可以为与 TEX 相关的内容指定一个基本目录。例如，对于许多 Unix 系统，这将是

```
\BaseDirectory{/usr/local/lib/texmf}
```

而对于标准 emTEX 安装，则为

```
\BaseDirectory{c:/emtex}
```

\DeclareDir 在指定了基本目录之后，您应该告诉 DocStrip 如何解释 \usedir 命令中使用的标签。这可以通过 \DeclareDir 实现，它有两个参数。第一个是标签，第二个是相对于基本目录的实际目录名称。例如，要让 DocStrip 使用标准 emTEX 目录，可以这样做：

```
\BaseDirectory{c:/emtex}
\DeclareDir{tex/latex/base}{texinput/latex2e}
\DeclareDir{tex/latex/tools}{texinput/tools}
\DeclareDir{makeindex}{idxstyle}
```

这将导致基本的 LaTeX 文件和字体描述被写入目录 c:\emtex\texinput\latex2e，tools 包的文件被写入 c:\emtex\texinput\tools，makeindex 文件被写入 c:\emtex\idxstyle。

有时希望将某些文件放在基本目录之外。因此，\DeclareDir 有一种星号形式，用于指定绝对路径名。例如，可以这样说：

```
\DeclareDir*{makeindex}{d:/tools/texindex/styles}
```

\UseTDS 符合 TDS 标准的系统用户可能会问，“我真的需要在我的配置文件中放置一打行像下面这样的内容吗?

```
\DeclareDir{tex/latex/base}{tex/latex/base}
```

答案是 \UseTDS。这个宏会使 DocStrip 对于您没有用 \DeclareDir 覆盖的任何目录都使用标签本身。默认行为是在未定义的标签上引发错误，因为一些用户可能希望准确知道文件的位置，而不允许 DocStrip 随意写入文件。然而，我（MW）觉得这非常酷，我的配置文件只有这么几行（我在 Linux 下运行 teTEX）：

```
\BaseDirectory{/usr/local/teTeX/texmf}
\UseTDS
```

重要的是要注意，无法在 TEX 中创建新目录。因此，无论您如何配置 DocStrip，都需要在运行安装程序之前创建所有必需的目录。作者可能希望在每次安装脚本开始时显示将要使用的目录列表，并询问用户是否确定所有这些目录都存在。

由于文件名语法是与操作系统相关的，DocStrip 试图从当前目录语法中猜测它。它应该能成功识别 Unix、MSDOS、Macintosh 和 VMS 的语法。但是，只有在与 LATEX 一起使用时，DocStrip 才会最初了解当前目录的语法[2]。如果您经常使用的格式不是 LATEX，您应该在文件 `docstrip.cfg` 中的开始部分定义 `\WriteToDir`。例如，在 MSDOS/Unix 上是 `\def\WriteToDir{./}`，在 Macintosh 上是 `\def\WriteToDir{:}`，在 VMS 上是 `\def\WriteToDir{[]}`。

如果您的系统需要完全不同的配置，可以在 `docstrip.cfg` 中定义宏 `\dirsep` 和 `\makepathname`。检查它们在实现部分的定义。如果希望使用完全不同的方案将 `\usedir` 标签转换为目录名称，请尝试重新定义宏 `\usedir`。

3.2 设置最大流的数量

`\maxfiles` 为了支持一些更晦涩的操作系统，程序能打开的文件数量有一定限制。通过 `\maxfiles` 宏，可以向 DocStrip 表达这种限制。如果 DocStrip 允许打开的流的数量是 n，您的配置文件可以写成 `\maxfiles{`n`}`，这样 DocStrip 就不会尝试打开比这更多的文件。请注意，此限制不包括已经打开的文件。通常会有两个文件已经打开：您启动的安装脚本和它所包含的文件 `docstrip.tex`；您必须自己注意这一点。在命中某种最大限制之前，DocStrip 假设它可以打开至少四个文件：如果情况并非如此，那就是真正的问题所在。

`\maxoutfiles` 也许与其限制 TEX 可打开的文件数量，不如限制它可以写入的文件数量（例如，TEX 本身限制了一次性写入的文件数量为 16 个）。通过在配置文件中使用 `\maxoutfiles{`m`}` 可以表达这种情况。您必须能够同时打开至少一个输出文件；否则 DocStrip 就无法做任何事情。

这两种选项通常会放在 `docstrip.cfg` 文件中。

4 用户界面

4.1 主程序

`\processbatchFile` ‘主程序’以尝试处理批处理文件开始，通过调用宏 `\processbatchFile` 来

[2]除了处理 LATEX 分发的主要 `unpack.ins` 批处理文件之外，该文件采取了特殊措施，以便 initex 可以了解目录语法。

完成此操作。它计算它处理的批处理文件数量，因此，在调用 \processbatchFile 后，如果处理的文件数量仍然为零，则可以采取适当的操作。

\interactive 当没有处理批处理文件时，调用宏 \interactive。它提示用户提供信息。首先确定输入和输出文件的扩展名。然后询问有关可选代码的问题，最后用户可以提供要处理的文件列表。

\ReportTotals 当在 DocStrip-程序中包含 stats 选项时，它会记录处理的文件和行数。同时记录删除和传递的注释数，以及传递到输出的代码行数。宏 \ReportTotals 显示了这些信息的摘要。

4.2 批处理文件命令

本节描述的命令可用于构建 TEX 的批处理文件。

\input 所有 DocStrip 批处理文件应以以下行开始：\input docstrip

不要使用 LATEX 语法 \input{docstrip}，因为批处理文件可能与 plain TEX 或 iniTEX 一起使用。您可能会发现旧批处理文件始终在输入之前有一行 \def\batchfile{⟨*filename*⟩}。虽然仍然支持这种用法，但现在已不鼓励使用，因为它会导致 TEX 重新输入相同的文件，消耗了它有限的输入流之一。

\endbatchfile 所有批处理文件应以此命令结束。文件中此命令后的任何行都将被忽略。在旧文件中，如果以 \def\batchfile{…开始，则此命令是可选的，但是建议始终使用。如果批处理文件中省略了此命令，则通常 TEX 将进入交互式 * 提示符，因此您可以通过在此提示符中键入 \endbatchfile 来停止 DocStrip。

\generate \file \from 构建 DocStrip 命令文件的主要原因是描述应该从哪些源文件生成文件，以及应该包括哪些可选（'guarded'）代码片段。宏 \generate 用于向 TEX 提供这些信息。其语法如下：

\generate{[\file{⟨*output*⟩}{[\from{⟨*input*⟩}{⟨*optionlist*⟩}]*}]*}

其中 ⟨*output*⟩ 和 ⟨*input*⟩ 是适合您计算机系统的普通文件规范。⟨*optionlist*⟩ 是一个用逗号分隔的 '选项' 列表，指定了应该在 ⟨*output*⟩ 中包括 ⟨*input*⟩ 中的哪些可选代码片段。\generate 的参数可以包含一些局部声明（例如下面描述的 \use... 命令），这些声明将适用于之后的所有 \file。\generate 的参数在组内执行，因此当 \generate 结束时，所有局部声明都会被取消。

可以指定多个输入文件，每个文件都有自己的 ⟨*optionlist*⟩。这可以通过记号 […]* 来表示。此外，在一个 \generate 子句中可以有多个 \file 规范。这意味着应该在读取每个 ⟨*input*⟩ 文件时生成所有这些 ⟨*output*⟩ 文件。输入文件按照在此子句中首次出现的顺序读取。例如，

```
\generate{\file{p1.sty}{\from{s1.dtx}{foo,bar}}
          \file{p2.sty}{\from{s2.dtx}{baz}
```

```
                    \from{s3.dtx}{baz}}
 \file{p3.sty}{\from{s1.dtx}{zip}
                    \from{s2.dtx}{zip}}
}
```

将导致 DocStrip 读取文件 s1.dtx、s2.dtx、s3.dtx（按照该顺序）并生成文件 p1.sty、p2.sty、p3.sty。

限制同时打开最多 16 个输出流并不意味着你只能用一个 \generate 命令生成最多 16 个文件。在上面的例子中，只需要 2 个流，因为在处理 s1.dtx 时只生成了 p1.sty 和 p3.sty；在读取 s2.dtx 时只生成了 p2.sty 和 p3.sty；而在读取 s3.dtx 时只有 p2.sty 文件。然而，下面的例子需要 3 个流：

```
\generate{\file{p1.sty}{\from{s1.dtx}{foo,bar}}
          \file{p2.sty}{\from{s2.dtx}{baz}
                        \from{s3.dtx}{baz}}
          \file{p3.sty}{\from{s1.dtx}{zip}
                        \from{s3.dtx}{zip}}
         }
```

尽管在读取 s2.dtx 文件时并没有写入 p3.sty，但它必须保持打开状态，因为稍后 s3.dtx 的某些部分将会写入其中。

有时候通过一次读取所有源文件来创建文件是不可能的。考虑以下例子：

```
\generate{\file{p1.sty}{\from{s1.dtx}{head}
                        \from{s2.dtx}{foo}
                        \from{s1.dtx}{tail}}
          \file{s1.drv}{\from{s1.dtx}{driver}}
         }
```

要生成 p1.sty 文件，必须对 s1.dtx 进行两次读取：第一次读取时带有选项 head，然后读取文件 s2.dtx，然后再次读取 s1.dtx，这次带有选项 tail。DocStrip 可以正确处理这种情况：如果在一个 \file 声明内有多个相同输入文件的 \from，那么该文件将被多次读取。

如果一个 \file 声明中 \from 的顺序与之前 \file 确定的输入文件顺序不匹配，DocStrip 将会报错并中止。然后，你可以阅读下一节，或者放弃并将该文件放入单独的 \generate 中（但那样源文件将再次被读取）。

对于急切的人。 尝试以下算法：找到从最多源文件生成的文件，以此文件和其源文件按正确顺序编写 \generate 子句。取出其他需要生成的文件，并检查它们是否不违反第一个文件源文件的顺序。如果这样不行，请阅读下一节。

对于数学家。 “文件 A 必须在文件 B 之前读取”是所有源文件集合上的偏序关系。每个 \from 子句都向这个顺序添加了一个链。你需要做的是执行拓扑排序，即将偏序扩展为线性排序。完成后，只需按照首次出现在子句中的顺序列出你的源文件在 \generate 中，使其顺序与线性顺序相匹配。如果无法实现此目标，请阅读下一段。（也许未来的 DocStrip 版本将自动执行此排序，因此所有这些问题都将消失。）

对于那些必须了解所有内容的人。 有一种特殊情况，无法实现源文件的正确读取顺序。假设你需要生成两个文件，第一个文件来自 s1.dtx 和 s3.dtx（按照那个顺序），第二个文件来自 s2.dtx 和 s3.dtx。无论如何指定，文件将会按照 s1 s3 s2 或 s2 s3 s1 的顺序读取。解决方法的关键是神奇的宏 \needed，它将一个文件标记为需要输入但不将任何输出从它导向当前的 \file。在我们的例子中，正确的规范是：

```
\generate{\file{p1.sty}{\from{s1.dtx}{foo}
                        \needed{s2.dtx}
                        \from{s3.dtx}{bar}}
          \file{p2.sty}{\from{s2.dtx}{zip}
                        \from{s3.dtx}{zap}}
         }
```

\askforoverwritetrue
\askforoverwritefalse

这些宏指定了如果要生成的文件已经存在时应该发生什么。如果 \askforoverwritetrue 处于激活状态（默认情况下），会询问用户是否应该覆盖文件。但是如果已经使用了 \askforoverwritefalse，则现有文件将会被静默地覆盖。这些开关是局部的，可以在文件的任何地方发出，甚至在 \generate 子句内部（但在 \file 之间）。

\askonceonly

你可能不想在批处理文件中设置 \askforoverwritefalse，因为这意味着始终可以悄悄地覆盖其他人的文件。然而对于大型安装（如基础 LaTeX 发行版），单独询问用户数百个文件是否覆盖并不是很有帮助。因此批处理文件可以指定 \askonceonly。这意味着在批处理文件第一次询问用户问题后，用户可以选择更改行为，以便将“是”自动应用于所有未来的问题。这适用于 DocStrip 命令 \Ask 的任何使用，包括但不限于由 \askforoverwritetrue 控制的文件覆盖问题。

\preamble
\endpreamble
\postamble
\endpostamble

可以向 DocStrip 程序的输出添加多行信息。要添加到输出文件开头的信息应列在 \preamble 和 \endpreamble 命令之间；要添加到输出文件末尾的行应列在 \postamble 和 \endpostamble 命令之间。对于 DocStrip 发现的所有前言和后言内容，都会写入输出文件，但前面会加上 \MetaPrefix 的值（默认是两个% 字符）。如果在这些行中包含 ^^J 字符，那么与其在同一行上的所

有内容都会被写入输出文件的新行中。这个“特性”可以用来向剥离后的文件中添加 \typeout 或 \message。

\declarepreamble
\declarepostamble
\usepreamble
\usepostamble
\nopreamble
\nopostamble

有时候，对于一个更大的包中的不同文件，希望拥有不同的前言是很理想的（例如，因为其中一些是可定制的配置文件，需要标记为这样）。在这种情况下，可以使用 \declarepreamble\somename，然后输入你的前言，用 \endpreamble 结束，稍后使用 \usepreamble\somename 切换到这个前言。如果不想使用任何前言，可以使用 \nopreamble 命令。此命令相当于说 \usepreamble\empty。同样的机制也适用于后言，\use... 声明是局部的，可以出现在 \generate 内部。

命令 \preamble 和 \postamble 定义并激活名为 \defaultpreamble 和 \defaultpostamble 的前（后）言。

\batchinput

批处理文件命令可以放入多个批处理文件中，然后从主批处理文件中执行这些文件。例如，如果一个分发包含几个不同的部分，这是很有用的。你可以为每个部分编写单独的批处理文件，并且还有一个主文件，简单地调用部分的批处理文件。为此，可以使用命令 \batchinput{⟨*file*⟩} 调用主文件中的单独批处理文件。不要使用 \input 来实现这个目的，这个命令只应该用于调用前面解释过的 DocStrip 程序，并且在用于其他目的时会被忽略。

\ifToplevel

当批处理文件被嵌套时，你可能希望在较低级别的批处理文件中抑制某些命令，比如终端消息。为此，可以使用 \ifToplevel 命令，它仅在当前批处理文件是最外层文件时执行其参数。确保将参数的左花括号放在与命令本身相同的行上，否则 DocStrip 程序会感到困惑。

\showprogress
\keepsilent

当在 DocStrip 中包含 stats 选项时，它可以在处理输入文件的每一行时向终端写入消息。这条消息由一个单个字符组成，表示特定行的类型。我们使用以下字符：

- % 每当输入行是注释时，在终端上写入 %-字符。
- . 每当遇到一个代码行时，在终端上写入一个 .-字符。
- / 当输入文件中出现一系列空行时，至多保留其中的一行。DocStrip 程序使用 /-字符表示删除的空行。
- < 当在输入中发现一个 ‘guard line’ 并且它开始一个可选包含的代码块时，在终端上通过显示 <-字符来表示，并伴随着 guard 的布尔表达式。
- > 条件包含的代码块结束时，通过显示 >-字符来表示。

当包含 stats 选项时，默认情况下会打开此功能，否则会关闭。可以使用命令 \showprogress 和 \keepsilent 来切换此功能。

4.2.1 支持旧接口

\generateFile 以下是指定要生成的文件的旧语法。它只允许指定一个输出文件。

\generateFile{⟨*output*⟩}{⟨*ask*⟩}{[\from{⟨*input*⟩}{⟨*optionlist*⟩}]*}

其中 ⟨*output*⟩、⟨*input*⟩ 和 ⟨*optionlist*⟩ 的含义与 \generate 相同。通过 ⟨*ask*⟩，你可以指示 TEX 是静默地覆盖先前存在的文件（f），还是发出警告并询问是否应该覆盖现有文件（t）（它会覆盖 \askforoverwrite 设置）。

\include \processFile 早期版本的 DocStrip 程序支持一种不同类型的命令告诉 TEX 要做什么。这个命令比 \generateFile 功能较弱；当 ⟨*output*⟩ 是由一个 ⟨*input*⟩ 创建时可以使用。语法如下：

\include{⟨*optionlist*⟩}

\processFile{⟨*name*⟩}{⟨*inext*⟩}{⟨*outext*⟩}{⟨*ask*⟩}

这个命令基于文件名由两部分构成的环境，即名称和扩展名，用点分隔。此命令的语法假定 ⟨*input*⟩ 和 ⟨*output*⟩ 共享相同的名称，只在扩展名上有所不同。为了向后兼容旧版 DocStrip，保留了此命令，但不鼓励使用。

5 代码的条件包含

当你使用 DocStrip 程序剥离 TEX 宏文件中的注释时，你有可能从一个文档文件中生成多个剥离后的宏文件。这是通过对可选代码的支持实现的。在文档文件中，可选代码是通过一个 'guard' 来标记的。

guard 是一个布尔表达式，它被包含在 < 和 > 中。它还必须紧跟在行首的 % 后面。例如：

```
...
%<bool>\TeX code
...
```

在这个例子中，如果在 \generateFile 命令的 ⟨*optionlist*⟩ 中存在选项 bool，那么这行代码将被包含在 ⟨*output*⟩ 中。

布尔表达式的语法是：

⟨*Expression*⟩ ::= ⟨*Secondary*⟩ [{|, ,} ⟨*Secondary*⟩]*

⟨*Secondary*⟩ ::= ⟨*Primary*⟩ [& ⟨*Primary*⟩]*

⟨*Primary*⟩ ::= ⟨*Terminal*⟩ | !⟨*Primary*⟩ | (⟨*Expression*⟩)

| 代表析取，& 代表合取，! 代表否定。而 ⟨*Terminal*⟩ 是任意的字母序列当且仅当在必须包含的选项列表中评估为 ⟨*true*⟩（[3]）。

[3] iff 代表“当且仅当”

两种类型的可选代码受到支持：一种是可以放在一行文本上的可选代码，就像上面的例子一样；另一种是可以有可选代码块。

为了区分这两种可选代码，引入了 'guard 修饰符'。它是紧跟在 guard 的 `<` 后面的一个字符。它可以是 `*` 表示代码块的开始，也可以是 `/` 表示代码块的结束[4]。代码块的开始和结束 guard 必须单独占据一行。

当一个代码块不被包含时，该块内出现的任何 guard 都不会被评估。

6 内部函数和变量

LaTeX 开发的一个重要考虑因素是分离公共函数和内部函数。对于一个模块私有的函数和变量不应该被任何其他模块使用或修改。由于 TeX 没有任何正式的命名空间系统，这需要一种约定来指示哪些函数在代码级别的模块中是公共的，哪些是私有的。

使用 DocStrip 可以使用 "双部分" 系统来指示内部函数。在 `.dtx` 文件中，内部函数可以用 `@@` 代替模块名称，例如：

```
\cs_new_protected:Npn \@@_some_function:nn #1#2
  {
    % Some code here
  }
\tl_new:N \l_@@_internal_tl
```

要使用 DocStrip 提取代码，原始的 'guard' 机制被扩展，引入了语法 `%<@@=`⟨*module*⟩`>`。当提取代码时，⟨*module*⟩ 名称会替换 `@@`，这样

```
%<*package>
%<@@=foo>
\cs_new_protected:Npn \__foo_some_function:nn #1#2
   {
     % Some code here
   }
\tl_new:N \l__foo_internal_tl
%</package>
```

这段代码提取出来的形式如下：

```
\cs_new_protected:Npn \__foo_some_function:nn #1#2
   {
     % Some code here
```

[4]为了与早期版本的 DocStrip 兼容，也支持 `+` 和 `-` 作为 'guard 修饰符'。但是，与先前行为相比，与一个 `+` 修饰的 guard 对应的行不会包含在求值为假的 guard 块内，这里存在不兼容。

```
  }
\tl_new:N \l__foo_internal_tl
```

这里的 `__` 表示这些函数和变量是 `foo` 模块内部的。

使用 `@@@@` 可以在输出中得到 `@@`（使用 `@@@@@` 可得到 `@@@`）。对于较长的代码片段，可以通过给出一个空的模块名称来完全禁止替换，即使用语法 `%<@@=>`。

这个替换算法的具体步骤如下：

1. 首先，将 `@@@@` 视为一个特殊情况（通过使用一个临时伪装）。
2. 然后将所有的 `__@@` 更改为 `__`⟨*module*⟩。
3. 然后将所有剩余的 `_@@` 更改为 `__`⟨*module*⟩。
4. 然后将所有剩余的 `@@` 更改为 `__`⟨*module*⟩。
5. 最后，通过将每个“伪装的 `@@@@`”更改为 `@@` 来进行整理。

因此，替换意味着 `@@` 被 ⟨*module*⟩ 名称替换，并且在 `@@` 前的 0、1 或 2 个下划线被替换为精确的 2 个下划线（同时保留任何更多的下划线）。

7 其他语言

由于 TEX 是一个开放的系统，一些 TEX 包含有非 TEX 文件。一些作者使用 DocStrip 生成 PostScript 头文件、shell 脚本或其他语言中的程序。对于他们来说，DocStrip 的注释去除可能会引起一些问题。本节描述了如何有效地使用 DocStrip 生成非 TEX 文件。

7.1 添加到每个文件中的内容

在生成“其他”语言的文件时的第一个问题是 DocStrip 会向每个生成的文件的开头和结尾添加一些内容，这些内容可能与该语言的语法不匹配。所以我们会仔细研究到底具体添加了什么内容。

放在文件开头的整个文本都保存在由 `\declarepreamble` 定义的宏中。每一行输入到 `\declarepreamble` 的内容都会以 `\MetaPrefix` 的当前值作为前缀。标准的 DocStrip 头部会被插入到你的文本之前，并且宏 `\inFileName`、`\outFileName` 和 `\ReferenceLines` 被用作稍后填充信息的占位符（具体用于每个输出文件）。不要尝试重新定义这些宏。例如：

```
\declarepreamble\foo
```

```
----------------------------
Package FOO for use with TeX
\endpreamble
```

宏 \foo 就被定义为：

```
%%^^J
%% This is file `\outFileName ',^^J
%% generated with the docstrip utility.^^J
\ReferenceLines^^J
%% ----------------------------^^J
%% Package FOO for use with TeX.
```

你可以自由地对其进行操作甚至从头开始定义。要将前文嵌入到 Adobe 结构化注释中，只需使用 \edef：

```
\edef\foo{\perCent!PS-Adobe-3.0^^J%
          \DoubleperCent\space Title: \outFileName^^J%
          \foo^^J%
          \DoubleperCent\space EndComments}
```

然后使用 \usepreamble\foo 来选择你的新前文。关于后文的内容也适用相同的方法。

你也可以阻止 DocStrip 向文件中添加任何内容，并直接在代码中添加任何特定于语言的调用：

```
\generate{\usepreamble\empty
          \usepostamble\empty
          \file{foo.ps}{\from{mypackage.dtx}{ps}}}
```

或者使用 \nopreamble 和 \nopostamble。

7.2 Meta 注释（元注释）

你可以通过重新定义 \MetaPrefix 来更改用于将元注释放入输出文件中的前缀。它的默认值是 \DoubleperCent。前文使用了 \MetaPrefix 在 \declarepreamble 时的当前值，而源文件中的元注释使用了在 \generate 时的当前值。请注意，这意味着你不能同时使用不同的 \MetaPrefix 生成两个文件。

7.3 Verbatim 模式（逐字模式）

如果你的编程语言使用了某种结构可能会与 DocStrip 产生严重干扰（例如第一列中的百分号），你可能需要一种方式来阻止它被剥离。为此，DocStrip 提供了“逐字模式”。

形式为 %<< ⟨*END-TAG*⟩ 的“守卫表达式”标记了一个将逐字复制的部分，直到包含只有一个百分号且位于第一列，后跟 ⟨*END-TAG*⟩ 的行。你可以选择任何你想要的 ⟨*END-TAG*⟩，但请注意这里计算空格。例如：

```
%<*myblock>
some stupid()
   #computer<program>
%<<COMMENT
% These two lines are copied verbatim (including percents
%% even if \MetaPrefix is something different than %%).
%COMMENT
   using*strange@programming<language>
%</myblock>
```

输出为（当使用定义了 myblock 时）：

```
some stupid()
   #computer<program>
% These two lines are copied verbatim (including percents
%% even if \MetaPrefix is something different than %%).
   using*strange@programming<language>
```

8 生成文档

我们提供了一个简短的驱动文件，可以通过 DocStrip 程序使用条件 ‘driver’ 进行提取。为了允许在 IniTEX 时使用 docstrip.dtx 作为程序（例如，去除自己的注释），我们需要添加一些原始代码。通过这种额外的检查，仍然可以使用 LATEX 2ε 处理此文件以排版文档。 如果 \documentclass 未定义，例如在 IniTEX 或 plain TEX 中进行格式化时，我们会绕过驱动文件。

我们使用一些技巧来避免在 \ifx 结构未完成时发出 \end{document}。如果下面的条件为真，则会实时构建一个 \fi，完成了 \ifx，真正的 \fi 将永远不会被看到，因为它位于 \end{document} 之后。另一方面，如果条件为假，TEX 将跳过 \csname fi\endcsname，不知道它可能代表 \fi，驱动文件将被跳过，然后才完成条件。

额外的保护 gobble 防止 DocStrip 将这些技巧提取到真实的驱动文件中。否则，我们会处理以下行，从而生成文档格式。