etoolbox 宏包

面向类和包作者的 e-TeX 工具箱

Philipp Lehman, Joseph Wright
joseph.wright@morningstar2.co.uk
张泓知 翻译

Version v2.5k
2020/10/05
翻译日期 2023 年 12 月 8 日

Contents

1 介绍

1.1 关于 etoolbox

etoolbox 宏包是一个编程工具箱，主要面向 LaTeX 类和宏包作者。它提供了一些 e-TeX 提供的新原语的 LaTeX 前端，以及一些与 e-TeX 无关但与本宏包的特性相匹配的通用工具。

1.2 许可证

2 用户命令

本节中的工具旨在面向常规用户以及类和宏包作者。

2.1 定义

\newrobustcmd{⟨*command*⟩}[⟨*arguments*⟩][⟨*optarg default*⟩]{⟨*replacement text*⟩}
\newrobustcmd*{⟨*command*⟩}[⟨*arguments*⟩][⟨*optarg default*⟩]{⟨*replacement text*⟩}

此命令的语法和行为类似于 `\newcommand`，但新定义的 ⟨*command*⟩ 将是强健的 (robust)。此命令的行为与 LaTeX 核心的 `\DeclareRobustCommand` 命令不同之处在于，如果 ⟨*command*⟩ 已经定义，它将发出错误消息而不仅仅是信息性消息。由于它使用 e-TeX 的低级保护机制而不是相应的更高级别的 LaTeX 工具，因此不需要额外的宏来实现‘强健性 (robustness) ’。

\renewrobustcmd{⟨*command*⟩}[⟨*arguments*⟩][⟨*optarg default*⟩]{⟨*replacement text*⟩}
\renewrobustcmd*{⟨*command*⟩}[⟨*arguments*⟩][⟨*optarg default*⟩]{⟨*replacement text*⟩}

此命令的语法和行为类似于 `\renewcommand`，但重新定义的 ⟨*command*⟩ 将是强健的 (robust)。

\providerobustcmd{⟨*command*⟩}[⟨*arguments*⟩][⟨*optarg default*⟩]{⟨*replacement text*⟩}
\providerobustcmd*{⟨*command*⟩}[⟨*arguments*⟩][⟨*optarg default*⟩]{⟨*replacement text*⟩}

此命令的语法和行为类似于 `\providecommand`，但新定义的 ⟨*command*⟩ 将是强健的 (robust)。请注意，此命令只会为未定义的 ⟨*command*⟩ 提供强健 (robust) 定义。它不会使已经定义的 ⟨*command*⟩ 变得强健 (robust)。

[1] http://www.latex-project.org/lppl/

2.2 补丁 (Patching)

\robustify{⟨*command*⟩}

重新定义了一个由 \newcommand 定义的 ⟨*command*⟩，使其成为强健 (robust) 命令，而不更改其参数、前缀或替换文本。如果 ⟨*command*⟩ 是用 \DeclareRobustCommand 定义的，则会自动检测到，并且 LaTeX 的高级保护机制将被相应的低级 e-TeX 功能替换。

2.3 保护

\protecting{⟨*code*⟩}

此命令将 LaTeX 的保护机制（通常需要在每个易损命令前加上 \protect）应用于任意 ⟨*code*⟩ 的整个代码块。其行为取决于当前的 \protect 状态。请注意，即使是单个记号，⟨*code*⟩ 周围的大括号也是必需的。

2.4 长度与计数器分配

此部分的工具是 \setcounter 和 \setlength 的替代品，支持算术表达式。

\defcounter{⟨*counter*⟩}{⟨*integer expression*⟩}

将值分配给之前用 \newcounter 初始化的 LaTeX ⟨*counter*⟩。此命令在概念和语法上类似于 \setcounter，但有两个主要区别。1）第二个参数可以是一个 ⟨*integer expression*⟩，它将使用 \numexpr 处理。⟨*integer expression*⟩ 可以是此上下文中有效的任意代码。分配给 ⟨*counter*⟩ 的值将是计算结果。2）与 \setcounter 相反，默认情况下，此赋值是局部的，但 \defcounter 可以使用 \global 进行前缀。\setcounter 的功能等效于 \global\defcounter。

\deflength{⟨*length*⟩}{⟨*glue expression*⟩}

将值分配给之前用 \newlength 初始化的 ⟨*length*⟩ 寄存器。此命令在概念和语法上类似于 \setlength，但第二个参数可以是一个 ⟨*glue expression*⟩，它将使用 \glueexpr 处理。⟨*glue expression*⟩ 可以是此上下文中有效的任意代码。分配给 ⟨*length*⟩ 寄存器的值将是计算结果。此赋值默认情况下是局部的，但 \deflength 可以使用 \global 进行前缀。此命令可以作为 \setlength 的替代。

2.5 附加文档钩子

LaTeX 提供了两个钩子，可以推迟代码的执行，要么推迟到文档正文的开头，要么推迟到文档的末尾。任何 `\AtBeginDocument` 代码都会在主 `aux` 文件首次读取后，在文档正文的开头执行。任何 `\AtEndDocument` 代码都会在文档末尾执行，在主 `aux` 文件第二次读取之前执行。这里介绍的钩子在概念上类似，但将其 ⟨*code*⟩ 参数的执行推迟到略微不同的位置。⟨*code*⟩ 可以是任意 TeX 代码。⟨*code*⟩ 参数中的参数字符是允许的，不需要加倍。

`\AfterPreamble`{⟨*code*⟩}

此钩子是 `\AtBeginDocument` 的变体，可在导言部分和文档正文中使用。在导言部分使用时，其行为与 `\AtBeginDocument` 完全相同。在文档正文中使用时，它会立即执行其 ⟨*code*⟩ 参数。`\AtBeginDocument` 在这种情况下会发出错误。此钩子非常有用，用于推迟需要写入主 `aux` 文件的代码。

`\AtEndPreamble`{⟨*code*⟩}

此钩子与 `\AtBeginDocument` 不同，因为 ⟨*code*⟩ 在导言部分的最后执行，在主 `aux` 文件（在前一个 LaTeX pass 上写入）被读取之前，也在任何 `\AtBeginDocument` 代码之前执行。请注意，此时不可能向 `aux` 文件写入。

`\AfterEndPreamble`{⟨*code*⟩}

此钩子与 `\AtBeginDocument` 不同，因为 ⟨*code*⟩ 在 `\begin{document}` 的最后执行，而且在任何 `\AtBeginDocument` 代码之后执行。请注意，在此钩子执行时，使用 `\@onlypreamble` 限定其作用域的命令将不再可用。

`\AfterEndDocument`{⟨*code*⟩}

此钩子与 `\AtEndDocument` 不同，因为 ⟨*code*⟩ 在文档的最后执行，在当前 LaTeX pass 上写入的主 `aux` 文件被读取之后，并在任何 `\AtEndDocument` 代码之后执行。

从某种意义上说，`\AtBeginDocument` 代码既不是导言部分的一部分，也不是文档正文的一部分，因为它在排版之前执行初始化序列的中间位置。有时，将代码移至导言部分的末尾是有必要的，因为此时已经加载了所有请求的包。然而，如果代码需要在 `aux` 文件中执行，`\AtBeginDocument` 代码执行太晚了。相反，`\AtEndPreamble` 代码是导言部分的一部分；`\AfterEndPreamble` 代码是文档正文的一部分，并可能包含要在文档开头排版的可打印文本。总而言之，在 `\begin{document}` 中，LaTeX 将执行以下任务：

- 执行任何 \AtEndPreamble 代码
- 开始文档正文的初始化（页面布局、默认字体等）
- 加载前一个 LaTeX pass 上写入的主 aux 文件
- 在当前 pass 上打开主 aux 文件进行写入
- 继续文档正文的初始化
- 执行任何 \AtBeginDocument 代码
- 完成文档正文的初始化
- 禁用所有 \@onlypreamble 命令
- 执行任何 \AfterEndPreamble 代码

在 \end{document} 中，LaTeX 将执行以下任务：

- 执行任何 \AtEndDocument 代码
- 执行最终的 \clearpage 操作
- 关闭写入的主 aux 文件
- 加载当前 pass 上写入的主 aux 文件
- 执行最终的测试和警告，如果适用的话
- 执行任何 \AfterEndDocument 代码

任何 \AtEndDocument 代码都可以被视为文档正文的一部分，因为在其执行时，仍然可以执行排版任务并写入主 aux 文件。\AfterEndDocument 代码不是文档正文的一部分。此钩子对于在 LaTeX pass 的最后评估 aux 文件中的数据非常有用。

2.6 环境钩子

本节工具提供了针对环境的钩子。请注意，它们不会修改⟨*environment*⟩的定义。它们挂钩到 \begin 和 \end 命令上。重新定义 ⟨*environment*⟩ 不会清除相应的钩子。参数字符在 ⟨*code*⟩ 参数中是允许的，无需加倍。

\AtBeginEnvironment{⟨*environment*⟩}{⟨*code*⟩}

将任意 ⟨*code*⟩ 附加到由 \begin 命令在给定 ⟨*environment*⟩ 的开头执行的钩子中，在 \⟨*environment*⟩ 之前，位于由 \begin 打开的组内。

\AtEndEnvironment{⟨*environment*⟩}{⟨*code*⟩}

将任意 ⟨*code*⟩ 附加到由 \end 命令在给定 ⟨*environment*⟩ 的末尾执行的钩子中，在 \end⟨*environment*⟩ 之前，位于由 \begin 打开的组内。

\BeforeBeginEnvironment{⟨*environment*⟩}{⟨*code*⟩}

将任意 ⟨*code*⟩ 附加到由 \begin 命令在非常早期执行的钩子中，在持有环境的组被打开之前。

\AfterEndEnvironment{⟨*environment*⟩}{⟨*code*⟩}

将任意 ⟨*code*⟩ 附加到由 \end 命令在非常晚期执行的钩子中，在持有环境的组被关闭后执行。

3 作者命令

本节工具面向类和包的作者。

3.1 定义

3.1.1 宏定义

本节工具是简单但经常需要的快捷方式，扩展了 LaTeX 核心中 \@namedef 和 \@nameuse 宏的作用范围。

\csdef{⟨*csname*⟩}⟨*arguments*⟩{⟨*replacement text*⟩}

类似于 TeX 原语 \def，但它以控制序列名称作为第一个参数。此命令是强健的 (robust)，对应于 \@namedef。

\csgdef{⟨*csname*⟩}⟨*arguments*⟩{⟨*replacement text*⟩}

类似于 TeX 原语 \gdef，但它以控制序列名称作为第一个参数。此命令是强健的 (robust)。

\csedef{⟨*csname*⟩}⟨*arguments*⟩{⟨*replacement text*⟩}

类似于 TeX 原语 \edef，但它以控制序列名称作为第一个参数。此命令是强健的 (robust)。

\csxdef{⟨*csname*⟩}⟨*arguments*⟩{⟨*replacement text*⟩}

类似于 TeX 原语 \xdef，但它以控制序列名称作为第一个参数。此命令是强健的 (robust)。

\protected@csedef{⟨*csname*⟩}⟨*arguments*⟩{⟨*replacement text*⟩}

类似于 \csedef，但临时启用了 LaTeX 的保护机制。换句话说，此命令类似于 LaTeX 核心命令 \protected@edef，但它以控制序列名称作为第一个参数。此命令是强健的 (robust)。

\protected@csxdef{⟨*csname*⟩}⟨*arguments*⟩{⟨*replacement text*⟩}

类似于 \csxdef，但临时启用了 LaTeX 的保护机制。换句话说，此命令类似于 LaTeX 核心命令 \protected@xdef，但它以控制序列名称作为第一个参数。此命令是强健的 (robust)。

\cslet{⟨*csname*⟩}{⟨*command*⟩}

类似于 TeX 原语 \let，但第一个参数是一个控制序列名称。如果 ⟨*command*⟩ 未定义，在分配后 ⟨*csname*⟩ 也将被未定义。此命令是强健的 (robust)，可带有 \global 前缀。

\letcs{⟨*command*⟩}{⟨*csname*⟩}

类似于 TeX 原语 \let，但第二个参数是一个控制序列名称。如果 ⟨*csname*⟩ 未定义，在分配后 ⟨*command*⟩ 也将被未定义。此命令是强健的 (robust)，可带有 \global 前缀。

\csletcs{⟨*csname*⟩}{⟨*csname*⟩}

类似于 TeX 原语 \let，但两个参数都是控制序列名称。如果第二个 ⟨*csname*⟩ 未定义，在分配后第一个 ⟨*csname*⟩ 也将被未定义。此命令是强健的 (robust)，可带有 \global 前缀。

\csuse{⟨*csname*⟩}

以控制序列名称为参数，并形成控制序列记号。此命令不同于 LaTeX 核心中的 \@nameuse 宏，如果控制序列未定义，它会展开为空字符串。

\undef⟨*command*⟩

清除 ⟨*command*⟩，使得 e-TeX 的 \ifdefined 和 \ifcsname 测试将其视为未定义。此命令是强健的 (robust)，可带有 \global 前缀。

\gundef⟨*command*⟩

类似于 \undef，但是作用于全局。

\csundef{⟨*csname*⟩}

类似于 \undef，但它以控制序列名称作为参数。此命令是强健的 (robust)，可带有 \global 前缀。

\csgundef{⟨*csname*⟩}

类似于 \csundef，但是作用于全局。

\csmeaning{⟨*csname*⟩}

类似于 TeX 原语 \meaning，但以控制序列名称作为参数。如果控制序列未定义，此命令不会隐式地将其赋予 \relax 的含义。

\csshow{⟨*csname*⟩}

类似于 TeX 原语 \show，但以控制序列名称作为参数。如果控制序列未定义，此命令不会隐式地将其赋予 \relax 的含义。此命令是强健的 (robust)。

3.1.2 算术定义

本节中的工具允许使用宏进行计算，而不是长度寄存器和计数器。

\numdef⟨*command*⟩{⟨*integer expression*⟩}

类似于 \edef，不同之处在于 ⟨*integer expression*⟩ 使用 \numexpr 处理。⟨*integer expression*⟩ 可以是在此上下文中有效的任意代码。分配给 ⟨*command*⟩ 的替换文本将是该计算的结果。如果 ⟨*command*⟩ 未定义，则在处理 ⟨*integer expression*⟩ 之前将其初始化为 0.

\numgdef⟨*command*⟩{⟨*integer expression*⟩}

类似于 \numdef，不同之处在于分配是全局的。

\csnumdef{⟨*csname*⟩}{⟨*integer expression*⟩}

类似于 \numdef，不同之处在于它以控制序列名称作为其第一个参数。

\csnumgdef{⟨*csname*⟩}{⟨*integer expression*⟩}

类似于 \numgdef，不同之处在于它以控制序列名称作为其第一个参数。

\dimdef⟨*command*⟩{⟨*dimen expression*⟩}

类似于 \edef ，不同之处在于 ⟨*dimen expression*⟩ 使用 \dimexpr 处理。⟨*dimen expression*⟩ 可以是在此上下文中有效的任意代码。分配给 ⟨*command*⟩ 的替换文本将是该计算的结果。如果 ⟨*command*⟩ 未定义，则在处理 ⟨*dimen expression*⟩ 之前将其初始化为 0pt。

\dimgdef⟨*command*⟩{⟨*dimen expression*⟩}

类似于 \dimdef，不同之处在于分配是全局的。

\csdimdef{⟨*csname*⟩}{⟨*dimen expression*⟩}

类似于 \dimdef，不同之处在于它以控制序列名称作为其第一个参数。

\csdimgdef{⟨*csname*⟩}{⟨*dimen expression*⟩}

类似于 \dimgdef，不同之处在于它以控制序列名称作为其第一个参数。

\gluedef⟨*command*⟩{⟨*glue expression*⟩}

类似于 \edef，不同之处在于 ⟨*glue expression*⟩ 使用 \glueexpr 处理。⟨*glue expression*⟩ 可以是在此上下文中有效的任意代码。分配给 ⟨*command*⟩ 的替换文本将是该计算的结果。如果 ⟨*command*⟩ 未定义，则在处理 ⟨*glue expression*⟩ 之前将其初始化为 0pt plus 0pt minus 0pt。

\gluegdef⟨*command*⟩{⟨*glue expression*⟩}

类似于 \gluedef，不同之处在于分配是全局的。

\csgluedef{⟨*csname*⟩}{⟨*glue expression*⟩}

类似于 \gluedef，不同之处在于它以控制序列名称作为其第一个参数。

\csgluegdef{⟨*csname*⟩}{⟨*glue expression*⟩}

类似于 \gluegdef，不同之处在于它以控制序列名称作为其第一个参数。

\mudef⟨*command*⟩{⟨*muglue expression*⟩}

类似于 \edef ，不同之处在于 ⟨*muglue expression*⟩ 使用 \muexpr 处理。⟨*muglue expression*⟩ 可以是在此上下文中有效的任意代码。分配给 ⟨*command*⟩ 的替换文本将是该计算的结果。如果 ⟨*command*⟩ 未定义，则在处理 ⟨*muglue expression*⟩ 之前将其初始化为 0mu。

\mugdef⟨*command*⟩{⟨*muglue expression*⟩}

类似于 \mudef，不同之处在于分配是全局的。

\csmudef{⟨*csname*⟩}{⟨*muglue expression*⟩}

类似于 \mudef，不同之处在于它以控制序列名称作为其第一个参数。

\csmugdef{⟨*csname*⟩}{⟨*muglue expression*⟩}

类似于 \mugdef，不同之处在于它以控制序列名称作为其第一个参数。

3.2 展开控制

本节中的工具对于控制在 \edef 或类似情境下的展开很有用。

\expandonce⟨*command*⟩

此命令展开一次 ⟨*command*⟩ 并阻止进一步展开替换文本。此命令可展开。

\csexpandonce{⟨*csname*⟩}

类似于 \expandonce，不同之处在于它以控制序列名称作为其参数。

3.3 钩子管理

本节中的工具旨在进行钩子管理。在此上下文中，⟨*hook*⟩ 是一个没有任何参数和前缀的普通宏，用于收集稍后要执行的代码。这些工具也可以用于通过将代码附加到其替换文本来修补简单的宏。对于更复杂的修补操作，请参见第 3.4 节。本节中的所有命令都将初始化 ⟨*hook*⟩，如果未定义的话。

3.3.1 向钩子追加内容

本节中的工具用于向钩子追加任意代码。

`\appto`⟨*hook*⟩`{`⟨*code*⟩`}`

此命令将任意 ⟨*code*⟩ 追加到 ⟨*hook*⟩。如果 ⟨*code*⟩ 包含任何参数字符，它们无需加倍。此命令是强健的 (robust)。

`\gappto`⟨*hook*⟩`{`⟨*code*⟩`}`

类似于 `\appto`，不同之处在于分配是全局的。此命令可用作 LaTeX 核心中的 `\g@addto@macro` 命令的替代品。

`\eappto`⟨*hook*⟩`{`⟨*code*⟩`}`

此命令将任意 ⟨*code*⟩ 追加到 ⟨*hook*⟩。在定义时展开 ⟨*code*⟩。仅展开新的 ⟨*code*⟩，不展开 ⟨*hook*⟩ 的当前替换文本。此命令是强健的 (robust)。

`\xappto`⟨*hook*⟩`{`⟨*code*⟩`}`

类似于 `\eappto`，不同之处在于分配是全局的。

`\protected@eappto`⟨*hook*⟩`{`⟨*code*⟩`}`

类似于 `\eappto`，不同之处在于暂时启用了 LaTeX 的保护机制。

`\protected@xappto`⟨*hook*⟩`{`⟨*code*⟩`}`

类似于 `\xappto`，不同之处在于暂时启用了 LaTeX 的保护机制。

`\csappto{`⟨*csname*⟩`}{`⟨*code*⟩`}`

类似于 `\appto`，不同之处在于它以控制序列名称作为其第一个参数。

`\csgappto{`⟨*csname*⟩`}{`⟨*code*⟩`}`

类似于 `\gappto`，不同之处在于它以控制序列名称作为其第一个参数。

`\cseappto{`⟨*csname*⟩`}{`⟨*code*⟩`}`

类似于 `\eappto`，不同之处在于它以控制序列名称作为其第一个参数。

\csxappto{⟨*csname*⟩}{⟨*code*⟩}

类似于 \xappto，不同之处在于它以控制序列名称作为其第一个参数。

\protected@cseappto⟨*hook*⟩{⟨*code*⟩}

类似于 \protected@eappto，不同之处在于它以控制序列名称作为其第一个参数。

\protected@csxappto⟨*hook*⟩{⟨*code*⟩}

类似于 \protected@xappto，不同之处在于它以控制序列名称作为其第一个参数。

3.3.2 在钩子前面插入内容

本节中的工具向钩子前面插入任意代码，即将代码插入到钩子的开头而不是添加到末尾。

\preto⟨*hook*⟩{⟨*code*⟩}

类似于 \appto，不同之处在于 ⟨*code*⟩ 被前置。

\gpreto⟨*hook*⟩{⟨*code*⟩}

类似于 \preto，不同之处在于分配是全局的。

\epreto⟨*hook*⟩{⟨*code*⟩}

类似于 \eappto，不同之处在于 ⟨*code*⟩ 被前置。

\xpreto⟨*hook*⟩{⟨*code*⟩}

类似于 \epreto，不同之处在于分配是全局的。

\protected@epreto⟨*hook*⟩{⟨*code*⟩}

类似于 \epreto，不同之处在于暂时启用了 LaTeX 的保护机制。

\protected@xpreto⟨*hook*⟩{⟨*code*⟩}

类似于 \xpreto，不同之处在于暂时启用了 LaTeX 的保护机制。

\cspreto{⟨*csname*⟩}{⟨*code*⟩}

类似于 \preto，不同之处在于它以控制序列名称作为其第一个参数。

\csgpreto{⟨*csname*⟩}{⟨*code*⟩}

类似于 \gpreto，不同之处在于它以控制序列名称作为其第一个参数。

\csepreto{⟨*csname*⟩}{⟨*code*⟩}

类似于 \epreto，不同之处在于它以控制序列名称作为其第一个参数。

\csxpreto{⟨*csname*⟩}{⟨*code*⟩}

类似于 \xpreto，不同之处在于它以控制序列名称作为其第一个参数。

\protected@csepreto⟨*hook*⟩{⟨*code*⟩}

类似于 \protected@epreto，不同之处在于它以控制序列名称作为其第一个参数。

\protected@csxpreto⟨*hook*⟩{⟨*code*⟩}

类似于 \protected@xpreto，不同之处在于它以控制序列名称作为其第一个参数。

3.4 补丁 (Patching)

本节中的工具用于钩取或修改现有代码。这里介绍的所有命令都保留了被修补命令的参数和前缀。请注意，\outer 命令可能无法被修补。另外，本节中的命令在修补失败时不会自动发出任何错误信息。相反，它们接受一个 ⟨*failure*⟩ 参数，应该提供合适的回退代码或错误信息。在导言区使用 \tracingpatches 可以让这些命令将调试信息写入传输文件。

\patchcmd[⟨*prefix*⟩]{⟨*command*⟩}{⟨*search*⟩}{⟨*replace*⟩}{⟨*success*⟩}{⟨*failure*⟩}

此命令提取 ⟨*command*⟩ 的替换文本，用 ⟨*replace*⟩ 替换 ⟨*search*⟩，然后重新组装 ⟨*command*⟩。匹配模式与类别码无关，在被修补的 ⟨*command*⟩ 的替换文本中匹配第一个 ⟨*search*⟩ 模式。请注意，修补过程涉及对 ⟨*command*⟩ 的替换文本进行解记号化，然后在修补后重新记号化为当前的类别码制度。@ 符号的类别码临时设置为 11。如果 ⟨*command*⟩ 的替换文本中包含任何具有非标准类别码的记号，必须在修补之前调整相应的类别码。如果要替换或插入的代码涉及到 ⟨*command*⟩ 的参数，则参数字符无需加倍。如果指定了可选的 ⟨*prefix*⟩，它将替换 ⟨*command*⟩ 的前缀。空的 ⟨*prefix*⟩ 参数会从 ⟨*command*⟩ 中删除所有前缀。此命令是局部的。此命令在修补之前隐式执行相当于 \ifpatchable 的测试。如果此测试成功，则命令应用修补并执行 ⟨*success*⟩。如果测试失败，则执行 ⟨*failure*⟩，但不修改原始的 ⟨*command*⟩。此命令是强健的 (robust)。

\ifpatchable{⟨*command*⟩}{⟨*search*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果可以使用 \patchcmd 修补 ⟨*command*⟩ 并在其替换文本中找到 ⟨*search*⟩ 模式，则执行 ⟨*true*⟩；否则执行 ⟨*false*⟩。此命令是强健的 (robust)。

\ifpatchable*{⟨*command*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于 \ifpatchable，但星号变体不需要搜索模式。使用此版本检查是否可以使用 \apptocmd 和 \pretocmd 修补命令。

\apptocmd{⟨*command*⟩}{⟨*code*⟩}{⟨*success*⟩}{⟨*failure*⟩}

此命令向 ⟨*command*⟩ 的替换文本追加 ⟨*code*⟩。如果 ⟨*command*⟩ 是一个无参数的宏，则其行为类似于第 3.3.1 节中的 \appto。与 \appto 不同，\apptocmd 还可以用于修补带参数的命令。在这种情况下，它将解记号化 ⟨*command*⟩ 的替换文本，应用修补，并在当前类别码制度下重新记号化。@ 符号的类别码临时设置为 11。⟨*code*⟩ 可能涉及 ⟨*command*⟩ 的参数。此命令是局部的。如果修补成功，则执行 ⟨*success*⟩。如果修补失败，则执行 ⟨*failure*⟩，但不修改原始的 ⟨*command*⟩。此命令是强健的 (robust)。

\pretocmd{⟨*command*⟩}{⟨*code*⟩}{⟨*success*⟩}{⟨*failure*⟩}

此命令与 \apptocmd 类似，但 ⟨*code*⟩ 被插入到 ⟨*command*⟩ 的替换文本开头。如果 ⟨*command*⟩ 是一个无参数的宏，则其行为类似于第 3.3.1 节中的 \preto。与 \preto 不同，\pretocmd 也可用于修补带参数的命令。在这种情况下，它将解记号化 ⟨*command*⟩ 的替换文本，应用修补，并在当前类别码制度下重新记号化。@ 符号的类别码临时设置为 11。⟨*code*⟩ 可能涉及 ⟨*command*⟩ 的参数。此命令是局部的。如果修补成功，则执行 ⟨*success*⟩。如果修补失败，则执行 ⟨*failure*⟩，但不修改原始的 ⟨*command*⟩。此命令是强健的 (robust)。

\tracingpatches 启用所有修补命令的跟踪，包括 \ifpatchable。调试信息将写入传输文件。如果无法确定为何不应用修补或 \ifpatchable 返回 ⟨*false*⟩，则此命令很有用。此命令必须在导言区使用。

3.5 布尔标志

该包提供了两个完全独立的布尔标志接口。第 3.5.1 节中的工具是 \newif 的 LaTeX 前端。第 3.5.2 节中的工具使用了不同的机制。

3.5.1 TeX 标志

由于本节中的工具在内部基于 `\newif`，因此它们可以用于测试和更改先前用 `\newif` 定义的标志的状态。它们也与 `ifthen` 包的布尔测试兼容，并可以作为查询 TeX 原语（如 `\ifmmode`）的 LaTeX 接口。使用 `\newif` 方法需要每个标志总共三个宏。

`\newbool{`⟨*name*⟩`}`

定义名为 ⟨*name*⟩ 的新布尔标志。如果已经定义了该标志，则此命令会发出错误。新定义的标志的初始状态为 `false`。此命令是强健的 (robust)。

`\providebool{`⟨*name*⟩`}`

定义名为 ⟨*name*⟩ 的新布尔标志，除非已经定义。此命令是强健的 (robust)。

`\booltrue{`⟨*name*⟩`}`

将布尔标志 ⟨*name*⟩ 设置为 `true`。此命令是强健的 (robust)，可以使用 `\global` 修饰。如果标志未定义，它将发出错误。

`\boolfalse{`⟨*name*⟩`}`

将布尔标志 ⟨*name*⟩ 设置为 `false`。此命令是强健的 (robust)，可以使用 `\global` 修饰。如果标志未定义，它将发出错误。

`\setbool{`⟨*name*⟩`}{`⟨*value*⟩`}`

将布尔标志 ⟨*name*⟩ 设置为 ⟨*value*⟩，可以是 `true` 或 `false`。此命令是强健的 (robust)，可以使用 `\global` 修饰。如果标志未定义，它将发出错误。

`\ifbool{`⟨*name*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

如果布尔标志 ⟨*name*⟩ 的状态为 `true`，则展开为 ⟨*true*⟩，否则为 ⟨*false*⟩。如果标志未定义，此命令将发出错误。此命令可用于执行基于普通 TeX 语法的任何布尔测试，即通常使用以下方式：

```
\iftest true\else false\fi
```

这包括所有使用 `\newif` 定义的标志，以及 TeX 原语，如 `\ifmmode`。在表达式中使用标志或原语时省略 `\if` 前缀。例如：

```
\ifmytest true\else false\fi
\ifmmode true\else false\fi
```

变为

```
\ifbool{mytest}{true}{false}
\ifbool{mmode}{true}{false}
```

\notbool{⟨*name*⟩}{⟨*not true*⟩}{⟨*not false*⟩}

类似于 `\ifbool`，但对测试进行否定。

3.5.2 LaTeX 标志

与第 3.5.1 节中的标志相比，本节中的工具每个标志仅需要一个宏。它们还使用单独的命名空间，以避免与常规宏产生名称冲突。

\newtoggle{⟨*name*⟩}

定义名为 ⟨*name*⟩ 的新布尔标志。如果已经定义了该标志，则此命令会发出错误。新定义的标志的初始状态为 `false`。此命令是强健的 (robust)。

\providetoggle{⟨*name*⟩}

定义名为 ⟨*name*⟩ 的新布尔标志，除非已经定义。此命令是强健的 (robust)。

\toggletrue{⟨*name*⟩}

将布尔标志 ⟨*name*⟩ 设置为 `true`。此命令是强健的 (robust)，可以使用 `\global` 修饰。如果标志未定义，它将发出错误。

\togglefalse{⟨*name*⟩}

将布尔标志 ⟨*name*⟩ 设置为 `false`。此命令是强健的 (robust)，可以使用 `\global` 修饰。如果标志未定义，它将发出错误。

\settoggle{⟨*name*⟩}{⟨*value*⟩}

将布尔标志 ⟨*name*⟩ 设置为 ⟨*value*⟩，可以是 `true` 或 `false`。此命令是强健的 (robust)，可以使用 `\global` 修饰。如果标志未定义，它将发出错误。

\iftoggle{⟨*name*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果布尔标志 ⟨*name*⟩ 的状态为 `true`，则展开为 ⟨*true*⟩，否则为 ⟨*false*⟩。如果标志未定义，此命令将发出错误。

\nottoggle{⟨*name*⟩}{⟨*not true*⟩}{⟨*not false*⟩}

类似于 `\iftoggle`，但对测试进行否定。

3.6 通用测试

3.6.1 宏测试

\ifdef{⟨*control sequence*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果 ⟨*control sequence*⟩ 已定义，则展开为 ⟨*true*⟩，否则为 ⟨*false*⟩。请注意，即使控制序列的含义是 `\relax`，它也将被视为已定义。此命令是对 e-TeX 原语 `\ifdefined` 的 LaTeX 包装。

\ifcsdef{⟨*csname*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于 `\ifdef`，但它以控制序列名称作为其第一个参数。此命令是对 e-TeX 原语 `\ifcsname` 的 LaTeX 包装。

\ifundef{⟨*control sequence*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果 ⟨*control sequence*⟩ 未定义，则展开为 ⟨*true*⟩，否则为 ⟨*false*⟩。除了反转测试的逻辑外，此命令还不同于 `\ifdef`，它还包括如果命令的含义是 `\relax` 则将其视为未定义。

\ifcsundef{⟨*csname*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于 `\ifundef`，但它以控制序列名称作为其第一个参数。此命令可用作 LaTeX 核心中的 `\@ifundefined` 测试的替代品。

\ifdefmacro{⟨*control sequence*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果 ⟨*control sequence*⟩ 已定义且是一个宏，则展开为 ⟨*true*⟩，否则为 ⟨*false*⟩。

\ifcsmacro{⟨*csname*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于 `\ifdefmacro`，但它以控制序列名称作为其第一个参数。

\ifdefparam{⟨*control sequence*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果 ⟨*control sequence*⟩ 已定义且是具有一个或多个参数的宏，则展开为 ⟨*true*⟩，否则为 ⟨*false*⟩。

\ifcsparam{⟨*csname*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于 `\ifdefparam`，但它以控制序列名称作为其第一个参数。

\ifdefprefix{⟨*control sequence*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果 ⟨*control sequence*⟩ 已定义且是以 `\long` 和/或 `\protected` 为前缀的宏，则展开为 ⟨*true*⟩，否则为 ⟨*false*⟩。请注意，`\outer` 宏无法进行测试。

\ifcsprefix{⟨*csname*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于 `\ifdefprefix`，但它以控制序列名称作为其第一个参数。

\ifdefprotected{⟨*control sequence*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果 ⟨*control sequence*⟩ 已定义且是以 `\protected` 为前缀的宏，则展开为 ⟨*true*⟩，否则为 ⟨*false*⟩。

\ifcsprotected{⟨*csname*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于 `\ifdefprotected`，但它以控制序列名称作为其第一个参数。

\ifdefltxprotect{⟨*control sequence*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果 ⟨*control sequence*⟩ 已定义且是一个 LaTeX 保护外壳，则执行 ⟨*true*⟩，否则为 ⟨*false*⟩。此命令是强健的 (robust)。它将检测通过 `\DeclareRobustCommand` 或类似技术定义的命令。

\ifcsltxprotect{⟨*csname*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于 `\ifdefltxprotect`，但它以控制序列名称作为其第一个参数。

\ifdefempty{⟨*control sequence*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果 ⟨*control sequence*⟩ 已定义且是一个没有参数且替换文本为空的宏，则展开为 ⟨*true*⟩，否则为 ⟨*false*⟩。与 `\ifx` 不同，此测试忽略了 ⟨*command*⟩ 的前缀。

\ifcsempty{⟨*csname*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于 \ifdefempty，但它以控制序列名称作为其第一个参数。

\ifdefvoid{⟨*control sequence*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果 ⟨*control sequence*⟩ 未定义，或其含义是 \relax，或是一个没有参数且替换文本为空的控制序列，则展开为 ⟨*true*⟩，否则为 ⟨*false*⟩。

\ifcsvoid{⟨*csname*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于 \ifdefvoid，但它以控制序列名称作为其第一个参数。

\ifdefequal{⟨*control sequence*⟩}{⟨*control sequence*⟩}{⟨*true*⟩}{⟨*false*⟩}

比较两个控制序列，如果它们在 \ifx 的意义上相等，则展开为 ⟨*true*⟩，否则为 ⟨*false*⟩。与 \ifx 不同，此测试还将两个控制序列都未定义或含义为 \relax 视为 ⟨*false*⟩。

\ifcsequal{⟨*csname*⟩}{⟨*csname*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于 \ifdefequal，但它以控制序列名称作为其参数。

\ifdefstring{⟨*command*⟩}{⟨*string*⟩}{⟨*true*⟩}{⟨*false*⟩}

将 ⟨*command*⟩ 的替换文本与 ⟨*string*⟩ 进行比较，如果它们相等，则执行 ⟨*true*⟩，否则执行 ⟨*false*⟩。在测试中，⟨*command*⟩ 和 ⟨*string*⟩ 都不会被展开，并且比较是不受类别码影响的。在 ⟨*string*⟩ 参数中的控制序列标记将被解标记并视为字符串。此命令是强健的 (robust)。请注意，它只考虑 ⟨*command*⟩ 的替换文本。例如，这个测试

```
\long\edef\mymacro#1#2{\string&}
\ifdefstring{\mymacro}{&}{true}{false}
```

将得到 ⟨*true*⟩。忽略了 \mymacro 的前缀和参数，以及替换文本中的类别码。

\ifcsstring{⟨*csname*⟩}{⟨*string*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于 \ifdefstring，但它以控制序列名称作为其第一个参数。

\ifdefstrequal{⟨*command*⟩}{⟨*command*⟩}{⟨*true*⟩}{⟨*false*⟩}

执行两个命令的不受类别码影响的字符串比较。此命令类似于 \ifdefstring，但要比较的两个参数都是宏。此命令是强健的 (robust)。

\ifcsstrequal{⟨*csname*⟩}{⟨*csname*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于 \ifdefstrequal，但它以控制序列名称作为其参数。

3.6.2 计数器和长度测试

\ifdefcounter{⟨*control sequence*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果⟨*control sequence*⟩是由\newcount分配的 TeX \count 寄存器，则扩展为⟨*true*⟩，否则为⟨*false*⟩。

\ifcscounter{⟨*csname*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于\ifdefcounter，但它以控制序列名称作为其第一个参数。

\ifltxcounter{⟨*name*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果⟨*name*⟩是由\newcounter分配的 LaTeX 计数器，则扩展为⟨*true*⟩，否则为⟨*false*⟩。

\ifdeflength{⟨*control sequence*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果⟨*control sequence*⟩是由\newskip或\newlength分配的 TeX \skip 寄存器，则扩展为 ⟨*true*⟩，否则为⟨*false*⟩。

\ifcslength{⟨*csname*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于\ifdeflength，但它以控制序列名称作为其第一个参数。

\ifdefdimen{⟨*control sequence*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果⟨*control sequence*⟩是由\newdimen分配的 TeX \dimen 寄存器，则扩展为⟨*true*⟩，否则为⟨*false*⟩。

\ifcsdimen{⟨*csname*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于\ifdefdimen，但它以控制序列名称作为其第一个参数。

3.6.3 字符串测试

\ifstrequal{⟨*string*⟩}{⟨*string*⟩}{⟨*true*⟩}{⟨*false*⟩}

比较两个字符串，如果它们相等则执行⟨*true*⟩，否则执行⟨*false*⟩。在测试中不会展开字符串，比较时不考虑类别码。任何⟨*string*⟩参数中的控制序列标记都将被解记号化并视为字符串。此命令是强健的 (robust)。

\ifstrempty{⟨*string*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果⟨*string*⟩为空，则扩展为⟨*true*⟩，否则扩展为⟨*false*⟩。在测试中不会展开⟨*string*⟩。

\ifblank{⟨*string*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果⟨*string*⟩为空白（即空或只包含空格），则扩展为⟨*true*⟩，否则扩展为⟨*false*⟩。在测试中不会展开 ⟨*string*⟩。

\notblank{⟨*string*⟩}{⟨*not true*⟩}{⟨*not false*⟩}

类似于\ifblank，但否定了测试。

3.6.4 算术测试

\ifnumcomp{⟨*integer expression*⟩}{⟨*relation*⟩}{⟨*integer expression*⟩}{⟨*true*⟩}{⟨*false*⟩}

根据 ⟨*relation*⟩ 比较两个整数表达式，根据结果展开为 ⟨*true*⟩ 或 ⟨*false*⟩。⟨*relation*⟩ 可以是 <、> 或 =。两个整数表达式将使用 \numexpr 处理。一个 ⟨*integer expression*⟩ 可以是在此上下文中有效的任意代码。所有算术表达式都可以包含空格。以下是一些例子：

```
\ifnumcomp{3}{>}{6}{true}{false}
\ifnumcomp{(7 + 5) / 2}{=}{6}{true}{false}
\ifnumcomp{(7+5) / 4}{>}{3*(12-10)}{true}{false}
\newcounter{countA}
\setcounter{countA}{6}
\newcounter{countB}
\setcounter{countB}{5}
\ifnumcomp{\value{countA} * \value{countB}/2}{=}{15}{true}{false}
\ifnumcomp{6/2}{=}{5/2}{true}{false}
```

从技术上讲，此命令是 LaTeX 对 TeX 原语 \ifnum 的包装，包含了 \numexpr。注意，\numexpr 将对所有整数表达式的结果进行四舍五入，即在比较之前，两个表达式都将被处理并四舍五入。在上述示例的最后一行中，第二个表达式的结果是 2.5，四舍五入为 3，因此 \ifnumcomp 将展开为 ⟨*true*⟩。

\ifnumequal{⟨*integer expression*⟩}{⟨*integer expression*⟩}{⟨*true*⟩}{⟨*false*⟩}

\ifnumcomp{...}{=}{...}{...}{...} 的替代语法。

\ifnumgreater{⟨*integer expression*⟩}{⟨*integer expression*⟩}{⟨*true*⟩}{⟨*false*⟩}

\ifnumcomp{...}{>}{...}{...}{...} 的替代语法。

\ifnumless{⟨*integer expression*⟩}{⟨*integer expression*⟩}{⟨*true*⟩}{⟨*false*⟩}

\ifnumcomp{...}{<}{...}{...}{...} 的替代语法。

\ifnumodd{⟨*integer expression*⟩}{⟨*true*⟩}{⟨*false*⟩}

对整数表达式进行评估，如果结果为奇数则展开为 ⟨*true*⟩，否则展开为 ⟨*false*⟩。从技术上讲，此命令是 LaTeX 对 TeX 原语 \ifodd 的包装，包含了 \numexpr。

\ifdimcomp{⟨*dimen expression*⟩}{⟨*relation*⟩}{⟨*dimen expression*⟩}{⟨*true*⟩}{⟨*false*⟩}

根据 ⟨*relation*⟩ 比较两个尺寸表达式，并根据结果展开为 ⟨*true*⟩ 或 ⟨*false*⟩。⟨*relation*⟩ 可以是 <、> 或 =。两个尺寸表达式都将使用 \dimexpr 处理。一个 ⟨*dimen expression*⟩ 可以是在此上下文中有效的任意代码。所有算术表达式都可以包含空格。以下是一些例子：

```
\ifdimcomp{1cm}{=}{28.45274pt}{true}{false}
\ifdimcomp{(7pt + 5pt) / 2}{<}{2pt}{true}{false}
\ifdimcomp{(3.725pt + 0.025pt) * 2}{<}{7pt}{true}{false}
\newlength{\lengthA}
\setlength{\lengthA}{7.25pt}
\newlength{\lengthB}
\setlength{\lengthB}{4.75pt}
\ifdimcomp{(\lengthA + \lengthB) / 2}{>}{2.75pt * 2}{true}{false}
\ifdimcomp{(\lengthA + \lengthB) / 2}{>}{25pt / 6}{true}{false}
```

从技术上讲，此命令是 LaTeX 对 TeX 原语 \ifdim 的包装，包含了 \dimexpr。由于 \ifdimcomp 和 \ifnumcomp 都是可展开的，它们也可以嵌套使用：

```
\ifnumcomp{\ifdimcomp{5pt+5pt}{=}{10pt}{1}{0}}{>}{0}{true}{false}
```

\ifdimequal{⟨*dimen expression*⟩}{⟨*dimen expression*⟩}{⟨*true*⟩}{⟨*false*⟩}

\ifdimcomp{...}{=}{...}{...}{...} 的替代语法。

\ifdimgreater{⟨*dimen expression*⟩}{⟨*dimen expression*⟩}{⟨*true*⟩}{⟨*false*⟩}

\ifdimcomp{...}{>}{...}{...}{...} 的替代语法。

`\ifdimless{`⟨*dimen expression*⟩`}{`⟨*dimen expression*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

`\ifdimcomp{...}{<}{...}{...}{...}` 的替代语法。

3.6.5 布尔表达式

本节中的命令是 `ifthen` 包提供的 `\ifthenelse` 命令的替代品。它们具有相同的目的，但在语法、概念和实现上有所不同。与 `\ifthenelse` 不同，它们本身不提供任何测试，而是作为其他测试的前端。任何满足特定语法要求的测试都可以用于布尔表达式。

`\ifboolexpr{`⟨*expression*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

对 ⟨*expression*⟩ 进行评估，如果为真则执行 ⟨*true*⟩，否则执行 ⟨*false*⟩。从左到右顺序评估 ⟨*expression*⟩。以下元素在 ⟨*expression*⟩ 中可用，并在下面更详细地讨论：测试运算符 `togl`、`bool`、`test`；逻辑运算符 `not`、`and`、`or`；子表达式定界符 `(...)`。可以自由使用空格、制表符和换行符来可视化排列 ⟨*expression*⟩。在 ⟨*expression*⟩ 中不允许空白行。此命令是强健的 (robust)。

`\ifboolexpe{`⟨*expression*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

`\ifboolexpr` 的可展开版本，在仅限展开的上下文中（例如 `\edef` 或 `\write` 操作）可以处理。请注意，⟨*expression*⟩ 中使用的所有测试必须是可展开的，即使 `\ifboolexpe` 不位于仅展开的上下文中。

`\whileboolexpr{`⟨*expression*⟩`}{`⟨*code*⟩`}`

类似于 `\ifboolexpr` 对 ⟨*expression*⟩ 进行评估，并在表达式为真时重复执行 ⟨*code*⟩。⟨*code*⟩ 可以是任何有效的 TeX 或 LaTeX 代码。此命令是强健的 (robust)。

`\unlessboolexpr{`⟨*expression*⟩`}{`⟨*code*⟩`}`

类似于 `\whileboolexpr`，但否定了 ⟨*expression*⟩，即只有在表达式为真时才会重复执行 ⟨*code*⟩。此命令是强健的 (robust)。

以下测试运算符可用于 ⟨*expression*⟩：

`togl` 使用 `togl` 运算符测试使用 `\newtoggle` 定义的标志的状态。例如：

```
\iftoggle{mytoggle}{true}{false}
```

变为

```
\ifboolexpr{ togl {mytoggle} }{true}{false}
```

`togl` 运算符可以与 `\ifboolexpr` 和 `\ifboolexpe` 都一起使用。

`bool` 使用 `bool` 运算符基于纯 TeX 语法执行布尔测试，即通常使用如下方式的任何测试：

```
\iftest true\else false\fi
```

这包括所有使用 `\newif` 定义的标志以及诸如 `\ifmmode` 等 TeX 原语。在表达式中使用标志或原语时，忽略 `\if` 前缀。例如：

```
\ifmmode true\else false\fi
\ifmytest true\else false\fi
```

变为

```
\ifboolexpr{ bool {mmode} }{true}{false}
\ifboolexpr{ bool {mytest} }{true}{false}
```

这也适用于使用 `\newbool` 定义的标志（见 § 3.5.1）。在这种情况下，

```
\ifbool{mybool}{true}{false}
```

变为

```
\ifboolexpr{ bool {mybool} }{true}{false}
```

`bool` 运算符可以与 `\ifboolexpr` 和 `\ifboolexpe` 都一起使用。

`test` 使用 `test` 运算符基于 LaTeX 语法执行测试，即通常使用如下方式的任何测试：

```
\iftest{true}{false}
```

这适用于基于 LaTeX 语法的所有宏，即宏必须接受 ⟨*true*⟩ 和 ⟨*false*⟩ 参数，并且这些参数必须是最终参数。例如：

```
\ifdef{\somemacro}{true}{false}
\ifdimless{\textwidth}{365pt}{true}{false}
  \ifnumcomp{\value{somecounter}}{>}{3}{true}{false}
```

在 ⟨*expression*⟩ 中使用这些测试时，它们的 ⟨*true*⟩ 和 ⟨*false*⟩ 参数被省略。例如：

```
\ifcsdef{mymacro}{true}{false}
```

变为

```
\ifboolexpr{ test {\ifcsdef{mymacro}} }{true}{false}
```

和

```
\ifnumcomp{\value{mycounter}}{>}{3}{true}{false}
```

变为

```
\ifboolexpr{
  test {\ifnumcomp{\value{mycounter}}{>}{3}}
}
{true}
{false}
```

`test` 运算符可以无限制地与 `\ifboolexpr` 一起使用。它也可以与 `\ifboolexpe` 一起使用，前提是测试是可展开的。一些在 § 3.6 中的通用测试是强健的 (robust)。，即使 `\ifboolexpe` 不位于仅展开的上下文中，也不能与之一起使用。如果测试不可展开，请使用 `\ifboolexpr`。

由于 `\ifboolexpr` 和 `\ifboolexpe` 需要处理开销，通常在单个测试中使用它们没有好处。§ 3.6 中的独立测试比 `test` 更有效率，§ 3.5.1 中的 `\ifbool` 比 `bool` 更有效率，§ 3.5.2 中的 `\iftoggle` 比 `togl` 更有效率。`\ifboolexpr` 和 `\ifboolexpe` 的目的在于它们支持逻辑运算符和子表达式。以下逻辑运算符可用于 ⟨*expression*⟩：

not `not` 运算符否定紧接其后的元素的真值。您可以在 `togl`、`bool`、`test` 和子表达式前加上 `not`。例如：

```
\ifboolexpr{
  not bool {mybool}
}
{true}
{false}
```

如果 mybool 为 false，则结果为 ⟨*true*⟩；如果 mybool 为 true，则结果为 ⟨*false*⟩。并且

```
\ifboolexpr{
  not ( bool {boolA} or bool {boolB} )
}
{true}
{false}
```

如果 boolA 和 boolB 都为 false，则结果为 ⟨*true*⟩。

and

and 运算符表示合取（a 和 b 都为真）。如果 and 连接的所有元素都为真，则 ⟨*expression*⟩ 为真。例如：

```
\ifboolexpr{
  bool {boolA} and bool {boolB}
}
{true}
{false}
```

如果两个 bool 测试都为真，则结果为 ⟨*true*⟩。不提供 nand 运算符（否定的合取，即不是两者都为真），但可以通过在否定子表达式中使用 and 来表达。例如：

```
bool {boolA} nand bool {boolB}
```

可以写为

```
not ( bool {boolA} and bool {boolB} )
```

or

or 运算符表示非排他的析取（a 或 b 或两者都为真）。如果至少有一个使用 or 连接的元素为真，则 ⟨*expression*⟩ 为真。例如：

```
\ifboolexpr{
  togl {toglA} or togl {toglB}
}
{true}
{false}
```

如果至少一个或两个测试为真，则结果为 ⟨*true*⟩。不提供 nor 运算符（否定的非排他的析取，即两者都不为真），但可以通过在否定子表达式中使用 or 来表达。例如：

```
bool {boolA} nor bool {boolB}
```

可以写为

```
not ( bool {boolA} or bool {boolB} )
```

`(...)` 括号用于限定 ⟨*expression*⟩ 中的子表达式。子表达式会被计算，其计算结果会被视为封闭表达式中的单个真值。子表达式可以嵌套。例如，表达式：

```
( bool {boolA} or bool {boolB} )
and
( bool {boolC} or bool {boolD} )
```

如果两个子表达式都为真，则结果为真；即如果 `boolA`/`boolB` 中至少一个为真，且 `boolC`/`boolD` 中至少一个为真，则结果为真。如果所有元素都使用 `and` 或 `or` 连接，则通常不需要子表达式。例如，表达式

```
bool {boolA} and bool {boolB} and {boolC} and bool {boolD}
bool {boolA} or bool {boolB} or {boolC} or bool {boolD}
```

将产生预期的结果：第一个表达式如果所有元素都为真，则结果为真；第二个表达式如果至少一个元素为真，则结果为真。但是，当组合 `and` 和 `or` 时，建议始终将元素分组到子表达式中，以避免可能因形式布尔表达式的语义与自然语言的语义之间的差异而产生的误解。例如，以下表达式

```
bool {coffee} and bool {milk} or bool {sugar}
```

如果 `sugar` 为真，则始终为真，因为 `or` 运算符将 `and` 评估结果作为输入。与英语中的表达含义相反，它不是按照以下方式处理：

```
bool {coffee} and ( bool {milk} or bool {sugar} )
```

但是严格地从左到右进行评估：

```
( bool {coffee} and bool {milk} ) or bool {sugar}
```

这可能不是您想要的顺序。

3.7 列表处理

3.7.1 用户输入

本节工具主要用于处理用户输入。在构建供包内部使用的列表时，可能更倾向于使用第 3.7.2 节中的工具，因为它们允许测试元素是否在列表中。

\DeclareListParser{⟨*command*⟩}{⟨*separator*⟩}

此命令定义一个列表解析器，类似于下面的 \docsvlist 命令，其定义如下：

```
\DeclareListParser{\docsvlist}{,}
```

请注意，列表解析器对 ⟨*separator*⟩ 的分类代码很敏感。

\DeclareListParser*{⟨*command*⟩}{⟨*separator*⟩}

\DeclareListParser 的星号变体定义了类似于下面的 \forcsvlist 命令的列表解析器，其定义如下：

```
\DeclareListParser*{\forcsvlist}{,}
```

\docsvlist{⟨*item, item, ...*⟩}

此命令循环处理逗号分隔的项目列表，并对列表中的每个项目执行辅助命令 \do，将项目作为参数传递。与 LaTeX 核心中的 \@for 循环不同，\docsvlist 是可展开的。通过适当定义 \do，列表可以在 \edef 或可比较的上下文中处理。您可以在 \do 的替换文本末尾使用 \listbreak 来停止处理并丢弃列表中的剩余项目。列表分隔符后的空格将被忽略。如果项目包含逗号或以空格开头，则必须用花括号包装。在处理列表时，花括号将被移除。以下是使用示例，将逗号分隔的列表打印为 itemize 环境：

```
\begin{itemize}
  \renewcommand*{\do}[1]{\item #1}
        \docsvlist{item1, item2, {item3a, item3b}, item4}
\end{itemize}
```

还有另一个示例：

```
\renewcommand*{\do}[1]{* #1\MessageBreak}
\PackageInfo{mypackage}{%
  Example list:\MessageBreak
  \docsvlist{item1, item2, {item3a, item3b}, item4}}
```

在此示例中，列表将作为信息消息写入日志文件。列表处理发生在 \write 操作期间。

\forcsvlist{⟨*handler*⟩}{⟨*item, item, ...*⟩}

此命令与 \docsvlist 类似，但 \do 被在调用时指定的 ⟨*handler*⟩ 替换。⟨*handler*⟩ 也可以是一系列命令，只要最后给出的命令将项目作为尾随参数。例如，以下代码将把逗号分隔的项目列表转换为名为 \mylist 的内部列表：

```
\forcsvlist{\listadd\mylist}{item1, item2, item3}
```

3.7.2 内部列表

本节中的工具用于处理内部数据列表。在这个上下文中，'内部列表' 是指一个没有任何参数和前缀的普通宏，用于收集数据。这些列表使用特殊字符作为内部列表分隔符。[2] 在处理用户以列表格式输入的内容时，请参考第 3.7.1 节中的工具。

\listadd{⟨*listmacro*⟩}{⟨*item*⟩}

这个命令将一个 ⟨*item*⟩ 添加到 ⟨*listmacro*⟩ 中。空的 ⟨*item*⟩ 不会添加到列表中。

\listgadd{⟨*listmacro*⟩}{⟨*item*⟩}

类似于 \listadd，但是执行的是全局赋值。

\listeadd{⟨*listmacro*⟩}{⟨*item*⟩}

类似于 \listadd，但是 ⟨*item*⟩ 在定义时被展开。只有新的 ⟨*item*⟩ 会被展开，⟨*listmacro*⟩ 不会被展开。如果展开后的 ⟨*item*⟩ 是空的，那么它不会被添加到列表中。

\listxadd{⟨*listmacro*⟩}{⟨*item*⟩}

类似于 \listeadd，但是执行的是全局赋值。

\listcsadd{⟨*listcsname*⟩}{⟨*item*⟩}

类似于 \listadd，但是第一个参数是控制序列的名称。

\listcsgadd{⟨*listcsname*⟩}{⟨*item*⟩}

类似于 \listcsadd，但是执行的是全局赋值。

[2] 这个特殊字符是具有类别码 3 的 |。请注意，你不能通过 \listname 来排版列表。使用 \show 来检查列表。

\listcseadd{⟨*listcsname*⟩}{⟨*item*⟩}

类似于 \listeadd，但是第一个参数是控制序列的名称。

\listcsxadd{⟨*listcsname*⟩}{⟨*item*⟩}

类似于 \listcseadd，但是执行的是全局赋值。

\listremove{⟨*listmacro*⟩}{⟨*item*⟩}

这个命令从 ⟨*listmacro*⟩ 中移除一个 ⟨*item*⟩。空的 ⟨*item*⟩ 会被忽略。

\listgremove{⟨*listmacro*⟩}{⟨*item*⟩}

类似于 \listremove，但是执行的是全局赋值。

\listcsremove{⟨*listcsname*⟩}{⟨*item*⟩}

类似于 \listremove，但是第一个参数是控制序列的名称。

\listcsgremove{⟨*listcsname*⟩}{⟨*item*⟩}

类似于 \listcsremove，但是执行的是全局赋值。

\dolistloop{⟨*listmacro*⟩}

这个命令遍历 ⟨*listmacro*⟩ 中的所有项目，并为列表中的每个项目执行辅助命令 \do，将项目作为参数传递。列表循环本身是可展开的。你可以在 \do 的替换文本末尾使用 \listbreak 来停止处理，并丢弃列表中剩余的项目。以下是一个用法示例，将名为 \mylist 的内部列表打印为 itemize 环境：

```
\begin{itemize}
  \renewcommand*{\do}[1]{\item #1}
        \dolistloop{\mylist}
\end{itemize}
```

\dolistcsloop{⟨*listcsname*⟩}

类似于 \dolistloop，但是第一个参数是控制序列的名称。

\forlistloop{⟨*handler*⟩}{⟨*listmacro*⟩}

这个命令类似于 \dolistloop，但是在调用时，\do 被指定的 ⟨*handler*⟩ 替代。⟨*handler*⟩ 也可以是一系列命令，只要最后给出的命令将项目作为尾随参数。例如，以下代码将在内部列表 \mylist 中的所有项目前加上 \item，在处理列表时计数项目，并在末尾附加项目计数：

```
\newcounter{itemcount}
\begin{itemize}
  \forlistloop{\stepcounter{itemcount}\item}{\mylist}
  \item Total: \number\value{itemcount} items
\end{itemize}
```

\forlistcsloop{⟨*handler*⟩}{⟨*listcsname*⟩}

类似于 \forlistloop，但是第二个参数是控制序列的名称。

\ifinlist{⟨*item*⟩}{⟨*listmacro*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果 ⟨*item*⟩ 包含在 ⟨*listmacro*⟩ 中，则执行 ⟨*true*⟩，否则执行 ⟨*false*⟩。请注意，此测试使用基于 TEX 的参数扫描器进行模式匹配，以检查搜索字符串是否包含在列表中。这意味着它通常比遍历列表中的所有项目更快，但也意味着项目不能包含花括号，否则会有效地隐藏它们。换句话说，在处理纯字符串列表而不是可打印数据时，这个宏最有用。在处理可打印文本时，最好使用 \dolistloop 来检查一个项目是否在列表中，如下所示：

```
\renewcommand*{\do}[1]{%
  \ifstrequal{#1}{item}
  {item found!\listbreak}
  {}}
\dolistloop{\mylist}
```

\xifinlist{⟨*item*⟩}{⟨*listmacro*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于 \ifinlist，但是在测试之前展开了 ⟨*item*⟩。

\ifinlistcs{⟨*item*⟩}{⟨*listcsname*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于 \ifinlist，但是第二个参数是控制序列的名称。

\xifinlistcs{⟨*item*⟩}{⟨*listcsname*⟩}{⟨*true*⟩}{⟨*false*⟩}

类似于 \xifinlist，但是第二个参数是控制序列的名称。

3.8 杂项工具

\rmntonum{⟨*numeral*⟩}

TeX 原语 \romannumeral 可以将整数转换为罗马数字，但是 TeX 或 LaTeX 没有提供相反的命令。\rmntonum 填补了这个空白。它以罗马数字作为参数，并将其转换为相应的整数。由于它是可展开的，因此也可以用于计数器赋值或算术测试：

```
\rmntonum{mcmxcv}
\setcounter{counter}{\rmntonum{CXVI}}
\ifnumless{\rmntonum{mcmxcviii}}{2000}{true}{false}
```

⟨*numeral*⟩ 参数必须是一个文字字符串。在解析之前，它将被去除记号化。罗马数字的解析不区分大小写，并忽略参数中的空格。如果参数中存在无效的记号，\rmntonum 将展开为 -1；空参数将产生空字符串。请注意，\rmntonum 不会检查罗马数字的形式有效性。例如，V 和 VX 都会产生 5，IC 会产生 99，等等。

\ifrmnum{⟨*string*⟩}{⟨*true*⟩}{⟨*false*⟩}

如果 ⟨*string*⟩ 是一个罗马数字，则展开为 ⟨*true*⟩，否则展开为 ⟨*false*⟩。在执行测试之前，⟨*string*⟩ 将被去除记号化。这个测试不区分大小写，并忽略 ⟨*string*⟩ 中的空格。请注意，\ifrmnum 不会检查罗马数字的形式有效性。例如，V 和 VXV 都会展开为 ⟨*true*⟩。严格来说，\ifrmnum 的作用是解析 ⟨*string*⟩ 以确定它是否由可以构成有效罗马数字的字符组成，但它不会检查它们是否真的是有效的罗马数字。

4 报告问题

etoolbox 宏包的开发代码托管在 GitHub 上：https://github.com/josephwright/etoolbox，这是记录该包任何问题的最佳位置。

5 修订历史

本修订历史记录了与此软件包用户相关的变更列表。不包括更多技术性质的更改，这些更改不影响用户界面或软件包的行为。如果修订历史中的条目说明某个功能已经“改

进”或“扩展”，这表示是一种语法向后兼容的修改，比如向现有命令添加可选参数。如果修订历史中的条目说明某个功能已被“修改”，那需要注意，这表示一种可能需要在一些，希望是少数情况下，修改现有文档的修改。右侧的数字表示本手册的相关部分。

2.5k 2020-10-05

内部更新

2.5j 2020-08-24

跟踪 LaTeX 2_ε 核心变更

2.5i 2020-07-13

跟踪 LaTeX 2_ε 核心变更

2.5h 2019-09-21

添加缺失的 `\gundef`

2.5g 2019-09-09

更新对 LaTeX 核心变更前 `\begin` 和 `\end` 的修补

2.5f 2018-08-18

修复 `\ifdefempty`、`\ifcsempty`、`\ifdefvoid` 和 `\ifcsvoid` 用于扩展为空格符的宏时出现的问题

2.5e 2018-02-11

在 `\DeclareListParser` 中对空列表分隔符进行更多工作

2.5d 2018-02-10

允许在 `\DeclareListParser` 中使用空列表分隔符

2.5c 2018-02-06

修复了由 v2.5b 引入的 `\forcsvlist` 的问题

2.5b 2018-02-04

在某些内部步骤中保留大括号

对列表处理器进行内部性能改进

2.5a 2018-02-03

对列表处理器进行内部性能改进

2.5 2017-11-22

添加了 `\csgundef` 3.1.1

添加了 `\gundef` 3.1.1

允许扫描包含换行符的宏

2.4 2017-01-02

将 `\listdel` 重命名为 `\listremove`（名称冲突） 3.7.2

将 `\listgdel` 重命名为 `\listgremove`（名称冲突） 3.7.2

2.3 2016-12-26

添加了 `\listdel` 3.7.2

添加了 `\listgdel` 3.7.2

2.2b 2016-12-01

修复了对某些类型的 LaTeX 强健 (robust) 命令的 `\ifdefltxprotect` 的问题

在 `\robustify` 处理后删除了冗余的宏

2.2a 2015-08-02

修复了 `\ifblank/\notblank` 中的强健性 (robustness) 错误

2.2 2015-05-04

添加了 `\csmeaning` 3.1.1

2.1d 2015-03-19

修复了与 `bm` 和某些类相关的问题

2.1c 2015-03-15

修复了 `\ifpatchable` 中的空格问题

修复了 `\patchcmd` 中的空格问题

修复了 `\robustify` 中的空格问题

2.1b 2015-03-10

对文档进行了轻微修订

2.1a 2015-03-10

新的维护者：Joseph Wright

在较新的 LaTeX 核心发布中跳过加载 `etex` 包

2.1 2011-01-03

2.0a 2010-09-12

2.0 2010-08-21

添加了 \forlistcsloop 3.7.2

允许在宏测试中测试 \par 3.6.1

修复了一些错误

1.9 2010-04-10

改进了 \letcs 3.1.1

改进了 \csletcs 3.1.1

改进了 \listeadd 3.7.2

改进了 \listxadd 3.7.2

添加了 \notblank 3.6.3

添加了 \ifnumodd 3.6.4

添加了 \ifboolexpr 3.6.5

添加了 \ifboolexpe 3.6.5

添加了 \whileboolexpr 3.6.5

添加了 \unlessboolexpr 3.6.5

1.8 2009-08-06

改进了 \deflength 2.4

添加了 \ifnumcomp 3.6.4

添加了 \ifnumequal 3.6.4

添加了 \ifnumgreater 3.6.4

添加了 \ifnumless 3.6.4

添加了 \ifdimcomp 3.6.4

添加了 \ifdimequal 3.6.4

添加了 \ifdimgreater 3.6.4

添加了 \ifdimless 3.6.4

1.7 2008-06-28

将 \AfterBeginDocument 重命名为 \AfterEndPreamble（名称冲突） 2.5

解决了与 hyperref 的冲突

稍微重新排列了手册内容

1.6 2008-06-22

改进了 \robustify 2.2

改进了 \patchcmd 和 \ifpatchable 3.4

修改并改进了 \apptocmd 3.4

修改并改进了 \pretocmd 3.4

添加了 \ifpatchable* 3.4

添加了 \tracingpatches 3.4

添加了 \AfterBeginDocument 2.5

添加了 \ifdefmacro 3.6.1

添加了 \ifcsmacro 3.6.1

添加了 \ifdefprefix 3.6.1

添加了 \ifcsprefix 3.6.1

添加了 \ifdefparam 3.6.1

添加了 \ifcsparam 3.6.1

添加了 \ifdefprotected 3.6.1

添加了 \ifcsprotected 3.6.1

添加了 \ifdefltxprotect 3.6.1

添加了 \ifcsltxprotect 3.6.1

添加了 \ifdefempty 3.6.1

添加了 \ifcsempty 3.6.1

改进了 \ifdefvoid 3.6.1

改进了 \ifcsvoid 3.6.1

添加了 \ifstrempty 3.6.3

添加了 \setbool 3.5.1

添加了 \settoggle 3.5.2

1.5 2008-04-26

添加了 \defcounter 2.4

添加了 \deflength 2.4

添加了 \ifdefstring 3.6.1

添加了 \ifcsstring 3.6.1

改进了 \rmntonum 3.8

添加了 \ifrmnum 3.8

添加了扩展的 PDF 书签到本手册

稍微重新排列了手册内容

1.4 2008-01-24

解决了与 tex4ht 的冲突

1.3 2007-10-08

将 elatex 包重命名为 etoolbox 1

将 \newswitch 重命名为 \newtoggle（名称冲突） 3.5.2

将 \provideswitch 重命名为 \providetoggle（保持一致性） 3.5.2

将 \switchtrue 重命名为 \toggletrue（保持一致性） 3.5.2

将 \switchfalse 重命名为 \togglefalse（保持一致性） 3.5.2

将 \ifswitch 重命名为 \iftoggle（保持一致性） 3.5.2

将 \notswitch 重命名为 \nottoggle（保持一致性） 3.5.2

添加了 \AtEndPreamble 2.5

添加了 \AfterEndDocument 2.5

添加了 \AfterPreamble 2.5

添加了 \undef 3.1.1

添加了 \csundef 3.1.1

添加了 \ifdefvoid 3.6.1

添加了 \ifcsvoid 3.6.1

添加了 \ifdefequal 3.6.1

1.2 2007-07-13

添加了 \provideswitch 3.5.2

添加了 \switchtrue 3.5.2

添加了 \switchfalse 3.5.2

添加了 \ifswitch 3.5.2

添加了 \notswitch 3.5.2

添加了 \DeclareListParser 3.7.1

添加了 \docsvlist 3.7.1

添加了 \rmntonum 3.8

1.1 2007-05-28

添加了 \protected@csedef 3.1.1

添加了 \protected@csxdef 3.1.1

添加了 \gluedef 3.1.2

添加了 \gluegdef 3.1.2

添加了 \csgluedef 3.1.2

添加了 \csgluegdef 3.1.2

添加了 \mudef 3.1.2

添加了 \mugdef 3.1.2

添加了 \csmudef 3.1.2

添加了 \csmugdef 3.1.2

添加了 \protected@eappto 3.3.1

添加了 \protected@xappto 3.3.1

添加了 \protected@cseappto 3.3.1

添加了 \protected@csxappto 3.3.1

添加了 \protected@epreto 3.3.2

添加了 \protected@xpreto 3.3.2

添加了 \protected@csepreto 3.3.2

添加了 \protected@csxpreto 3.3.2

修复了 `\newrobustcmd` 中的 bug 2.1

修复了 `\renewrobustcmd` 中的 bug 2.1

修复了 `\providerobustcmd` 中的 bug 2.1

1.0 2007-05-07

初始公开版本