LaTeX for package and class authors current version

中译本

2024-05-24

译者：我家有庭树

翻译时间：2024-07-28

[1]本文件可以在 LaTeX 公共许可的条件下发布和/或修改，无论是本许可的 1.3c 版本，还是（根据您的选择）任何以后的版本。有关详细详细信息，请参阅源代码 `clsguide.tex`。

目录

1 引言

LaTeX 2ε发布于 1994 年，并为 LaTeX 添加了一些当时的新概念。对于宏包和类作者，这些概念在文档 clsguide-historic 中进行了描述，该文档基本保持不变。从那时起，LaTeX 团队致力于许多想法，首先是用于 LaTeX 的编程语言（L3 编程层），然后是基于该语言构建的一系列工具。本文描述了由 LaTeX 内核为包和类开发人员提供的当前稳定工具集。我们假设文档作者熟悉一般的 LaTeX 使用，并且这里的材料应与 userguide 一起阅读，后者为常规的的 LaTeX 用户提供了关于创建命令的最新方法的信息。

2 编写类和宏包

本章节介绍了一些编写 LaTeX 包和类的一般要点。

2.1 是类还是包

当决定要在文档中添加一些新的 LaTeX 命令时，首先要决定应该编写文档类还是宏包，经验法则为：

> 如果命令可以用于任意文档类，就将其作为宏包；如果不是，就作为文档类。

文档类主要有两种类型：一种是如（article）文章、（report）报告或（letter）信件这样的独立类；另一种是其他类的扩展或变体---例如，建立在 article 文档类之上的 proc 文档类。

因此，一家公司可能有一个本地的 ownlet 文档类，用于打印带有自己抬头信纸的信件。这样的类建立在现有的 letter 类之上，因为它不能与任何其他文档类一起使用，所以应该使用 ownlet.cls 而不是 ownlet.sty。

相比之下，graphics 包提供了用来将图像包含到 LaTeX 文档中的命令。由于这些命令可以用于任何文档类，所以有 graphics.sty，而不是 graphics.cls。

2.2 使用 'docstrip' 和 'doc'

如果想要编写一个大型的用于 LaTeX 的类或宏包，你应该考虑使用 LaTeX 提供的 doc 软件。使用此软件编写的 LaTeX 类和包可以通过两种方式进行处理：可以通过 LaTeX 运行，以生成文档；或者可以使用 docstrip 进行处理，以生成 .cls 或 .sty 文件。

doc 软件可以自动生成定义索引、命令使用索引和更改日志列表。它对于维护和记录大型 TeX 源代码非常有用。

LaTeX 内核本身和标准类等的文档来源是 doc 文档；它们在发行版中的 .dtx 文件中。实际上，您可以通过在 source2e.tex 上运行 LaTeX，将内核的源代码排版为一个长文档，并包含索引。排版这些文档使用类文件 ltxdoc.cls。

有关 doc 和 docstrip 的更多信息，请参考文件 docstrip.dtx、doc.dtx 和书籍 *The LaTeX Companion*。有关其使用的示例，请查看 .dtx 文件。

2.3 标准文档类中的策略

我们收到的许多关于标准类的问题报告并不是关于错误的，而是或多或少礼貌地建议这些类中所体现的设计决策“并不理想”，并要求我们对其进行修改。

我们不应对这些文件进行此类更改的原因有几个。

- 尽管可能存在误解，但当前的行为显然是这些类设计时的意图。
- 改变“标准类”的这些方面并不是一个好做法，因为许多人依赖于它们。

因此，我们决定甚至不考虑进行此类修改，也不花时间来证明这一决定。这并不意味着我们不同意这些类的设计存在许多缺陷，但我们有许多优先级更高的任务，而不是不断解释为什么 LaTeX 的标准类不能被更改。

当然，我们欢迎更好的类的产生，或可以用来增强这些类的包。因此，当您考虑到这种缺陷时，我们希望您的第一反应是“我能做些什么来改善这一点？”

2.4 命令名称

在下一章介绍的 L3 编程层之前，LaTeX 具有三种类型的命令。

作者命令，如 \section，\emph，\times，这些命令大多为简称，且均为小写。

类和宏包编写命令，这些命令大多为长的、混合大小写的名称，例如：

```
\InputIfFileExists  \RequirePackage  \PassOptionsToClass
```

最后还有在 LaTeX 实现中使用的内部命令，例如 `\@tempchta`，`\@ifnextchar`，`\@eha`。这些命令基本上都包含一个 `@` 符号，这意味着它们没法使用在文档中，只能用在类和宏包文件中。

不幸的是，由于历史原因，这些命令之间的区别往往比较模糊。例如，`\hbox` 是一个内部命令，本来只应该在 LaTeX 内核中使用。`\m@ne` 表示常数-1，而它本应该叫 `\MinusOne`。

然而，这条经验法则依然适用：如果一个命令中包含 `@`，那么它就不是支持的 LaTeX 语言的一部分，并且其行为可能在未来的版本中改变。如果一个命令是混合大小写的，或者在 *LaTeX: A Document Preparation System* 被描述，你可以依赖 LaTeX 未来版本中支持该命令。

2.5 编程支持

正如在引言中提到的，现在的 LaTeX 内核加载了专门的编程支持，称为 L3 编程层（L3 programming layer），也常被称为 expl3。L3 编程层所采用的一般方法的详细信息在文档 `exp13` 中给出，而所有当前代码接口的参考可在 `interface3` 中找到。该层包含两种类型的命令：一组文档化的命令构成了 API，以及大量的私有内部命令。后者均以两个下划线开头，不应在定义它们的代码模块之外使用。这种更结构化的方法意味着使用 L3 编程层不会受到上述提到的“`@` 命令”的某种流动情况的影响。

我们在这里不详细介绍使用 L3 编程层的细节。有关该方法的介绍可在 https://www.alanshawn.com/latex3-tutorial/ 中找到。

2.6 盒子（Box）命令与颜色

即使您不打算在文档中使用颜色，通过了解本章中的要点，也可以确保您编写的类和宏包和这些颜色宏包兼容。这也有利于希望使用颜色的用户使用您的类和宏包。

确保“颜色安全”的一种简单的方式就是始终使用 LaTeX 盒子命令，而不是 TeX 原始语法，使用 `\sbox` 而不是 `\setbox`，`\mbox` 而不是 `\hbox` 和 `\parbox`，或 `minipage` 环境而不是 `\vbox`。LaTeX 盒子命令拥有新的选项，这意味着它们和 TeX 原始语法一样强大。

作为可能出错的一个例子，考虑在 `{\ttfamily <text>}` 中，字体在 `}` 之前恢复，而在类似的构造 `{\color{green} <text>}` 中，颜色在最后的 `}` 之后恢复。通常，这种区别并不重要；但考虑一个原始的 TeX 分配，例如：

```
\setbox0=\hbox{\color{green} <text>}
```

现在颜色恢复发生在 } 之后，因此不会存储在盒子中。这可能产生的坏影响取决于颜色是如何实现的：他可能导致文档其余部分的颜色错误，也可能在打印文时导致 dvi 驱动出错。

另一个值得关注的命令是 \normalcolor，这个命令通常只是 \relax（什么都不做），但是你可以像使用 \normalfont 一样使用该命令，将页面的某些区域（例如标题和章节）设置为“主文档颜色”。

2.7 常规样式

LaTeX 提供了许多命令，旨在帮助你生成良好结构的类和宏包文件，这些文件具有良好的稳健性和可移植性。本章节概述了一些命令的使用方法。

2.7.1 加载其他文件

LaTeX 提供了以下命令用于在类和宏包文件中使用其它的类和宏包：

```
\LoadClass          \LoadClassWithOptions
\RequirePackage     \RequirePackageWithOptions
```

我们强烈推荐您使用它们，而不是使用原始 \input 命令，原因有很多。

使用 \input <filenames> 命令加载的文件不会被 \listfiles 列出。

如果一个宏包始终使用 \RequirePackage 或 \usepackage 加载，那么即使请求多次加载，也只会被加载一次。相反，如果使用 \input 加载，则可能加载多次，这种额外的加载会浪费时间和内存，并可能产生奇怪的结果。

如果一个包提供选项处理，那么如果使用 input 加载可能会再一次产生奇怪的结果。

如果包 foo.sty 使用 \input baz.sty 加载包 \baz.sty，将会收到一个警告：

```
LaTeX Warning: You have requested package ‘foo’,
               but the package provides ‘baz’.
```

因此，基于以上原因，使用 \input 加载包并非一个好主意。例如，article.sty 仅包含以下几行：

```
\NeedsTeXFormat{LaTeX2e}
\@obsoletefile{article.cls}{article.sty}
\LoadClass{article}
```

您可能希望做同样的事情，或者如果您认为这样做是安全的，您可以决定直接删除 myclass.sty。

2.7.2 使其稳健

我们认为在编写宏包和类时，尽可能使用 LaTeX 命令是一种良好的实践。

因此，我们推荐使用 \newcommand，renewcommand 或 \providecommand 而不是 \def... 去编程并且定义文档接口 \NewDocumentCommand 等。(这些命令的细节参见 userguide)。

当你定义一个新环境时，请使用 \NewDocumentEnvironment 等（或在简单情况下使用 \newenvironment 等），而不是使用 \def\foo{...} 和 \def\endfoo{...} 等。

如果需要设置或改变 ⟨*dimen*⟩ 和 ⟨*skip*⟩ 的值，使用 \setlength 命令。

为了操作盒子，请使用 LaTeX 命令如 \sbox，\mbox 和 \parbox 而不是 \setbox，\hbox 和 \vbox。

使用 \PackageError,\PackageWarning 或者 \PackageInfo 而不是 \@latexerr,\@warning 或者 \wlong。

这种实践的优点在于，您的代码更具可读性，并且对其他有经验的 LaTeX 程序员更易于访问。

2.7.3 使其可移植

令您的文件尽可能具有可移植性是非常重要的。为了确保这一点，文件的名称不应与标准 LaTeX 分发中的文件相同，尽管其内容可能与这些文件的其中之一相似。坚持使用仅包含 ASCII 范围的文件名始终具有较低的风险；虽然 LaTeX 原生支持 UTF-8，但不能肯定所有工具都能如此。因此，出于同样的原因，避免在文件名中使用空格。

如果本地类或宏包具有共同的前缀也是非常有用的，例如 University of Nowhere 类可以以 unw 开始。这可以避免所有大学学位类都叫做 thesis.cls。

如果你依赖于 LaTeX 核心或某个宏包的特性，请您指定需要的发布日期。例如，宏包错误命令在 2022 年 6 月份的发布中被引入，如果你使用了他们，应该写：

```
\NeedsTeXFormat{LaTeX2e}[2022-06-01]
```

2.7.4 有用的钩子

有时，包需要安排在前言开始或结束时、文档结束时或每次使用环境时执行代码。这可以通过使用钩子来实现。作为一个文档作者，你可能会熟悉 \AtBeginDocument，这是一个围绕更强大命令 \AddToHook 的包装器。LaTeX 核心提供了大量的专用钩子（用于预定义位置）和通用钩子（用于任意命令）：使用这些的接口在 ithooks 中描述。这也有些其他的应用于文件中的钩子，在 itfilehooks 中描述。

3 类或宏包的结构

类和宏包的轮廓结构是：

Identification（标识） 文件声明它是一个LaTeX 2$_\varepsilon$包或类，并给出简短的自我描述。

Preliminary declarations（初步声明） 在这里，文件声明一些命令，并且还可以加载其他文件。通常这些命令只是声明选项中使用的代码所需的命令。

Options（选项） 文件声明并处理其选项。

More declarations（更多声明） 这是文件进行大部分工作的地方：声明新变量、命令和字体；以及加载其他文件。

3.1 Identification（标识）

类或包文件首先要做的就是识别自己。包文件的识别方式如下：

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{<package>}[<date> <other information>]
```

例如：

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{latexsym}[1998-08-17 Standard LaTeX package]
```

类文件的识别方式如下：

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{<class-name>}[<date> <other information>]
```

例如：

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{article}[2022-06-01 Standard LaTeX class]
```

⟨*date*⟩ 应该以 'YYYY-MM-DD' 的格式给出，并且如果使用了可选参数，则必须存在（对于 `\NeedsTeXFormat` 命令也是如此）。任何偏离此语法的情况都会导致低级的 TeX 错误---这些命令期望有效的语法以加快包或类的日常使用，并且不考虑开发者可能犯的错误！

每当用户在其 `\documentclass` 或者 `\usepackage` 命令中指定日期时，都会检查此日期。例如，如果您写：

```
\documentclass{article}[2022-06-01]
```

那么不同地区的用户将会收到警告，提示他们的 article 副本已过时。

类的描述在类使用时显示，包的描述则放在日志文件中，这些描述也会通过 \listfiles 命令显示。语法 Standard LaTeX 不得在任何非标准的 LaTeX 分发文件识别中使用。

3.2 使用类和宏包

一个 LaTeX 包和类可以通过以下语法加载一个包：

```
\RequirePackage[<options>]{<package>}[<date>]
```

例如：

```
\RequirePackage{ifthen}[2022-06-01]
```

该命令语法与作者命令 \usepackage 相同，它允许包或类使用其他包提供的功能。例如，通过加载 ifthen 包，包的编写者可以使用该包提供的 “if...then...else" 命令。

一个 LaTeX 类可以通过以下语法加载一个类：

```
\LoadClass[<options>]{<class-name>}[<date>]
```

例如：

```
\LoadClass[twocolumn]{article}
```

该命令语法和作者命令 \documentclass 相同，它允许类基于另一个类的语法和外观。例如，通过加载 article 类，类编写者只需要更改他们不喜欢的 article 部分，而无需从头编写一个新类。

在您想要简单的加载一个类或包文件，并且使用当前类所具有的确切选项的常见情况下，可以使用如下命令。

```
\LoadClassWithOptions{<class-name>}[<date>]
\RequirePackageWithOptions{<package>}[<date>]
```

例如：

```
\LoadClassWithOptions{article}
\RequirePackageWithOptions{graphics}[1995/12/01]
```

3.3 声明选项

包和类可以声明选项，这些选项可以有作者指定；例如，article 类声明了 twocolumn 选项。请注意，选项的名称应包含在 LaTeX 名称中允许的字符，特别是不应该包含任何控制序列。

LaTeX 支持两种创建选项的方法，键值系统和 ‘‘简单文本" 方法。推荐使用键值系统，因为他在处理选项时比文本方法更加灵活。这两种选项方法在 LaTeX 源代码中使用相同的基本结构：首先生命选项，然后处理选项。两者也允许将选项传递给其他包或底层类。由于经典的简单文本方法在概念上更易于说明，因此在这里使用它来展示一般结构，有关键值方法的完整细节，可以参见第4.4节。

选项的声明如下：

```
\DeclareOption{<option>}{<code>}
```

例如，graphics 包的 dvips 选项（略微简化）实现如下：

```
\DeclareOption{dvips}{\input{dvips.def}}
```

这意味当作者写下 \usepackage[dvips]{graphics} 时，dvips.def 文件会被加载。另一个例子，a4paper 选项在 article 类中被声明，用于设置 \paperheight 和 \paperwidth 的长度。

```
\DeclareOption{a4paper}{%
   \setlength{\paperheight}{297mm}%
   \setlength{\paperwidth}{210mm}%
}
```

有时用户会请求一个类或包没有明确声明的选项。默认情况下，这会产生一个警告（对于类）或错误（对于包）；这种行为可以通过以下方式进行更改：

```
\DeclareOption*{<code>}
```

例如，为了使包 fred 对未知选项产生警告而不是错误，可以指定：

```
\DeclareOption*{%
   \PackageWarning{fred}{Unknown option `\CurrentOption'}%
}
```

因此，如果作者写下 \usepackage[foo]{fred}，将会得到一个警告：Package fred Warning: Unknown option ‘foo’.。另一个例子，当使用 ⟨*ENC*⟩ 选项时，fontenc 包尝试加载文件 <ENC>enc.def，可以写作：

```
\DeclareOption*{%
   \input{\CurrentOption enc.def}%
}
```

可以使用 \PassOptionsToClass 或 \PassOptionsToPackage 命令将参数传递给包或类（请注意，这是一种特殊操作，仅适用于选项名称）：请参见4.5。例如，要将某个未知选项传递给 article 类，可以使用：

```
\DeclareOption*{%
   \PassOptionsToClass{\CurrentOption}{article}%
}
```

如果要这么做，那么应确保在之后的某个时刻加载该类，否则选项将不会被处理。到目前为止，我们只解释了如何声明选项，而不是如何执行它们。要处理调用文件时使用的选型，应该使用：

```
\ProcessOptions\relax
```

这将为每一个被指定和声明的选项执行代码，具体参见6.4，例如，如果 jane 包文件包含以下语句，当作者写下 \usepackage[foo,bar]{jane} 他们将会看到信息 Saw foo 以及 What’s bar。

```
\DeclareOption{foo}{\typeout{Saw foo.}}
\DeclareOption{baz}{\typeout{Saw baz.}}
\DeclareOption*{\typeout{What’s \CurrentOption?}}
\ProcessOptions\relax
```

3.4 最小类文件

类或包的大部分工作在于定义新的命令或改变文档的外观，这是在包的主体中完成的，使用诸如 \newcommand，\setlength 的命令。

每个类文件必须包含四个内容：\normalsize 的定义，\textwidth 和 \textheight 的长度，页面编号规范。因此，一个最小的文档类文件[2]看起来像是这样：

[2]该类现在已经包含在标准发行版中，名为 minimal.cls

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{minimal}[2022-06-01 Standard LaTeX minimal class]
\renewcommand{\normalsize}{\fontsize{10pt}{12pt}\selectfont}
\setlength{\textwidth}{6.5in}
\setlength{\textheight}{8in}
\pagenumbering{arabic}       % needed even though this class will
                             % not show page numbers
```

然后，该类文件不支持脚注、边注、浮动体等，也不会提供任何 2 个字母的命令如 \rm。因此，大多数类所包含的内容都会超过最小内容。

3.5 示例：本地 latter 类

一个公司可能有自己的 latter 类，用于设置公司风格的信件。本节展示了这样一个类的简单实现，尽管一个真正的类需要更多的结构。

该类首先声明自己为 neplet.cls：

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{neplet}[2022-06-01 NonExistent Press letter class]
```

接下类传递选项给 letter 类，加载 a4paper 选项：

```
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{letter}}
\ProcessOptions\relax
\LoadClass[a4paper]{letter}
```

为了使用公司信件抬头，重定义了第一页的页面风格，下面是用于 letters 第一页的页面风格：

```
\renewcommand{\ps@firstpage}{%
   \renewcommand{\@oddhead}{<letterhead goes here>}%
   \renewcommand{\@oddfoot}{<letterfoot goes here>}%
}
```

3.6 示例：newslatter 类

一个简单的新闻通讯（newslatter）可以使用 LaTeX 进行排版，使用 article 类的变体。该类首先声明自己为 smplnews.cls：

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{smplnews}[2022-06-01 The Simple News newsletter class]

\newcommand{\headlinecolor}{\normalcolor}
```

它将大多数指定的选项传递给 article 类，除了被设置为 off 的 onecolummn 选项和将标题设置为绿色的 green 选项。

```
\DeclareOption{onecolumn}{\OptionNotUsed}
\DeclareOption{green}{\renewcommand{\headlinecolor}{\color{green}}}

\DeclareOption*{\PassOptionsToClass{\CurrentOption}{article}}

\ProcessOptions\relax
```

上面代码的含义是：

1. \DeclareOption{onecolumn}{\OptionNotUsed} 这行命令用来处理已声明的选项。\ProcessOptions 命令会解析文档类的所有选项，并根据已声明的选项执行相应的命令。\relax 是一个占位符，用于告诉\ProcessOptions 命令处理完所有选项后停止执行。
2. \DeclareOption*{\PassOptionsToClass{\CurrentOption}{article}} 这行命令声明了一个名为* 的选项。当文档类使用任何未声明的选项编译时，\PassOptionsToClass{\CurrentOption}{article} 命令会被执行。这个命令将未声明的选项传递给 article 类。
3. \ProcessOptions\relax 这行命令用来处理已声明的选项。\ProcessOptions 命令会解析文档类的所有选项，并根据已声明的选项执行相应的命令。\relax 是一个占位符，用于告诉\ProcessOptions 命令处理完所有选项后停止执行。

然后，加载带有 twocolumn 选项的 article 类：

```
\LoadClass[twocolumn]{article}
```

因为 newsletter 需要彩色打印，现在加载 color 包。该类没有指定设备驱动程序选项，因为这应该由 smplnews 类的用户指定。

```
\RequirePackage{color}
```

然后该类重定义了 \maketitle 命令，以在适当的颜色下产生 72pt 的粗斜体标题。

```
\renewcommand{\maketitle}{%
   \twocolumn[%
      \fontsize{72}{80}\fontfamily{phv}\fontseries{b}%
      % \fontsize{72}{80}指定了字体大小为72点，行距为80点。
      % \fontfamily{phv}指定了字体家族为“phv”，即Arial字体。
      % \fontseries{b}指定了字体系列为“b”，即粗体。

      \fontshape{sl}\selectfont\headlinecolor
      %\fontshape{sl}指定了字体形状为“sl”，即斜体。
      %\selectfont命令用于应用所选字体。

      \@title
      %\@title 是LaTeX内部命令，用于获取文档标题的内容。
      %在这个环境下，它会将文档标题居中对齐。
   ]%
}
```

还重定义了 \section 命令用于取消章节编号。具体来说：

```
\renewcommand{\section}{%
   \@startsection
      {section}{1}{0pt}{-1.5ex plus -1ex minus -.2ex}%
      {1ex plus .2ex}{\large\sffamily\slshape\headlinecolor}%
}

\setcounter{secnumdepth}{0}
```

上面代码的具体含义为：

1. \renewcommand{\section}{...} 这行命令是用来重新定义 LaTeX 中的\section 命令。这个命令通常用来创建一个新的章节并生成相应的标题。通过重新定义\section，我们可以自定义这些标题的外观和行为。
2. \@startsection 是一个内部 LaTeX 命令，用于定义新节的结构。它有六个参数：
   (a) 第一个参数表示该级别节的类型，通常是 section, subsection 等。
   (b) 第二个参数是节的级别，1 对应于 section 级别。

(c) 第三个参数是该级别节的左缩进量，0pt 表示没有缩进。
(d) 第四个参数是该级别节的头部间距，这里使用了一个正负结合的值来控制间距，其中负值表示紧贴上一节的底部，正值则增加额外的间距。
(e) 第五个参数是该级别节的尾部间距，同样使用了正负结合的值。
(f) 第六个参数是该级别节的字体格式定义。

3. `{\large\sffamily\slshape\headlinecolor}` 定义了节标题的字体属性。`\large` 表示使用较大的字体，`\sffamily` 表示使用 Sans Serif 字体系列（通常是 Arial 或 Helvetica），`\slshape` 表示斜体，`\headlinecolor` 表示使用文章的标题颜色。
4. `\setcounter{secnumdepth}{0}` 这行命令用来设置章节编号的深度。这里的参数 0 表示只有级别 0 的章节（如章节）会被编号，而子章节（如小节）不会被编号。

除此之外，还设置了三个基本命令：

```
\renewcommand{\normalsize}{\fontsize{9}{10}\selectfont}
\setlength{\textwidth}{17.5cm}
\setlength{\textheight}{25cm}
```

实际上，一个类需要的不止这些，还会提供期刊号、文章作者、页面样式等命令；但是这个框架提供了一个起点，`itnews` 类文件并没有比这个复杂多少。

4 用于类和宏包编写的命令

本节简要描述了每个用于类和包编写的命令，要了解系统的其他方面，您还应该阅读：*LaTeX: A Document Preparation System*, *The LaTeX Companion* 和 *LaTeX 2ε for Authors*。

4.1 标识

这里讨论的第一组命令是用于识别您的类或包文件的命令。

`\NeedsTeXFormat {`⟨*format-name*⟩`} [`⟨*release-date*⟩`]`

该命令用于告知 TeX 该文件应使用名称为 ⟨*format-name*⟩ 的格式进行处理，您可以使用可选参数 ⟨*release-date*⟩ 进一步指定所需格式的最早发布日期。当格式的发布日期早于指定日期时，将生成警告。标准的 ⟨*format-name*⟩ 是LaTeX 2ε，日期必须采用 YYYY–MM–DD 的形式。例如：

```
\NeedsTeXFormat{LaTeX2e}[2022-06-01]
```

人们通常不知道在这里填写什么日期，对于内核，可以查阅 change.txt 找到正确日期，并选择您感兴趣的新功能的发布日期。对于包而言，稍有不同，因为他们在一年中发布，需要查阅他们的变更历史。

`\ProvidesClass {`⟨*class-name*⟩`} [`⟨*release-info*⟩`]`
`\ProvidesPackage {`⟨*package-name*⟩`} [`⟨*release-info*⟩`]`

这声明了当前文件包含文档类 ⟨*class-name*⟩ 和包 ⟨*package-name*⟩,如果使用了 ⟨*release info*⟩ 选项，则必须包含:

1. 文件当前版本的发布日期，形式为 YYYY-MM-DD。
2. 可选的后跟一个空格和简短的描述，可能包括版本号。

上述语法必须严格遵循,以便 LoadClass,\documentclass 或 \RequirePackage,\usepackage 可以测试该版本是否过旧。

整个⟨*release-info*⟩的信息会被 \listfiles 全部列出，因此其不宜过长，例如：

```
\ProvidesClass{article}[2022-06-01 v1.0 Standard LaTeX class]
\ProvidesPackage{ifthen}[2022-06-01 v1.0 Standard LaTeX package]
```

`\ProvidesFile {`⟨*file-name*⟩`} [`⟨*release-info*⟩`]`

这与前两个命令类似，除了这里必须给出完整的文件名，包括扩展名。它用于声明除主类和包文件之外的任何文件。例如：

```
\ProvidesFile{T1enc.def}[2022-06-01 v1.0 Standard LaTeX file]
```

请注意，短语 Standard LaTeX 不得用于任何文件的标识中，除了标准 LaTeX 发行版中的文件。

4.2 加载文件

这一组命令用于在已存在的类或包的基础上创建你自己的文档类或包。

`RequirePackage [`⟨*options-list*⟩`] {`⟨*package-name*⟩`} [`⟨*release-info*⟩`]`
`\RequirePackageWithOptions {`⟨*package-name*⟩`} [`⟨*release-info*⟩`]`

类或包应该使用上面的命令去加载其他包，\RequirePackage 命令的使用和作者命令 \usepackage 相同，例如：

```
\RequirePackage{ifthen}[2022-06-01]
\RequirePackageWithOptions{graphics}[2022-06-01]
```

\LoadClass [⟨*options-list*⟩] {⟨*class-name*⟩} [⟨*release-info*⟩]
\LoadClassWithOptions {⟨*class-name*⟩} [⟨*release-info*⟩]

这些命令只能用于类文件，不能用于包文件中；在一个类文件中这些命令最多使用一次。

\LoadClass 的使用和 \documentclass 相同，例如：

```
\LoadClass{article}[2022-06-01]
\LoadClassWithOptions{article}[2022-06-01]
```

带有 WithOptions 版本的命令仅加载当前文件所使用的具有确切选项的类或包文件，有关其使用的进一步讨论，请参见4.5。

4.3 延迟代码

如之前所说，ithooks 中描述了一个复杂的钩子系统，在这里，记录了一小组常用的钩子的简称。

下面两个命令主要用于 \DeclareOption 或 \DeclareOption* 中的 ⟨*code*⟩ 参数。

\AtEndOfClass {⟨*code*⟩}
\AtEndOfPackage {⟨*code*⟩}

这两个命令声明 ⟨*code*⟩ 在内部保存，然后再处理完当前类或包文件后执行。重复使用这些命令是允许的：参数中的代码按照他们声明的顺序被存储（稍后执行）。

\AtBeginDocument {⟨*code*⟩}
\AtEndDocument {⟨*code*⟩}

这两个命令声明 ⟨*code*⟩ 被存储在内部,并在 LaTeX 执行 \begin{document} 或 \end{document} 时执行。

在设置了字体选择表之后，在 \AtBeginDocument 命令中指定的 ⟨*code*⟩ 参数将会在 \begin{document} 代码末尾附近执行。因此，这是防止需要在用于排版的一切被准备好之后，并且当文档的正常字体是当前字体时执行的代码的有用位置。

\AtBeginDocument 钩子不应该用于执行任何排版的代码，因为排版的结果将是不可预测的。

在最后一页完成以及处理任何剩余的浮动环境之前，在命令 `\AtEndDocument` 中指定的 ⟨*code*⟩ 将会在 `\end{document}` 开始时执行。如果某些 ⟨*code*⟩ 需要在这两项处理之后执行，应该在 ⟨*code*⟩ 的适当位置添加 `\clearpage` 命令。

允许重复使用这些命令：参数中的代码按照声明的顺序存储（并在之后执行）。

4.4 创建和使用键-值选项

与任何键值输入一样，使用键值对作为包或类选项有两个部分；创建选项和设置它们。以这种方式创建的选项可以在包加载后作为一般键值设置使用，这取决于底层代码的性质。

`\DeclareKeys [`⟨*family*⟩`] {`⟨*declarations*⟩`}`

该命令从逗号分隔的 ⟨*declarations*⟩ 列表创建一系列选项。该列表中的每个条目都是一个键值对，其中 ⟨*key*⟩ 有一个或多个 ⟨*properties*⟩。下面描述了少部分的 '基本' ⟨*properties*⟩。全部的属性由 `l3keys` 提供，可被用于更强大的处理。查看 `interface3` 获取全部细节。

一些基本属性为：

1. `.code` – 执行任意代码
2. `.if` – 设置一个 TeX `\if...` 开关
3. `.ifnot` – 设置一个反向 TeX `\if...` 开关
4. `.store` – 在宏中存储值
5. `.usage` – 定义选项是否只能在加载时、前言中给出，或在范围上没有限制

⟨*key*⟩ 在 ⟨*property*⟩ 之前的部分是 ⟨*name*⟩，⟨*value*⟩ 和 ⟨*property*⟩ 一起工作以定义选项的行为。例如，使用

```
\DeclareKeys[mypkg]
 {
   draft.if          = @mypkg@draft      ,
   draft.usage       = preamble          ,
   name.store        = \@mypkg@name      ,
   name.usage        = load              ,
   second-name.store = \@mypkg@other@name
 }
```

将创建三个选项。选项 `draft` 可以在前言中的任何地方给出，并将设置一个名为 `\if@mypkg@draft` 的开关。选项 `name` 只能在包加载期间给出，并将其值保存在 `\@mypkg@other@name` 中。

在使用 \ProcessKeyOptions 之前创建的 Key 将作为包选项。

\DeclareUnknownKeyHandler [⟨*family*⟩] {⟨*code*⟩}

命令 \DeclareUnknownKeyHandler 可用于定义在遇到未定义键时的行为。⟨*code*⟩ 将接收未知键名称作为 #1 和值作为 #2。这些可以根据需要进行处理，例如，通过转发到另一个包。整个选项可作为 \\CurrentOption 使用，以便在需要传递可能包含或不包含 an= sign 符号的选项时。例如，这可以用于将未知选项传递给非键值类，例如 article：

```
\DeclareUnknownKeyHandler{%
  \PassOptionsToClass{\CurrentOption}{article}
}
```

\ProcessKeyOptions [⟨*family*⟩]

\ProcessKeyOptions 函数用于检查当前选项列表与为 ⟨*family*⟩ 定义的键。全局（类）选项和局部（包）选项在此函数在包中被调用时会被检查。该命令将处理当前包或类给出的所有选项：不需要再应用 \\ProcessOptions 。

\SetKeys [⟨*family*⟩] {⟨*keyvals*⟩}

为 ⟨*family*⟩ 设置（应用）显式的 ⟨*keyvals*⟩ 列表：如果未给出后者，则使用 \@currname 的值（value）。此命令可在包内使用，以便在使用 \ProcessKeyOptions 之前或之后设置选项。

4.5 传递选项

\PassOptionsToPackage {⟨*options-list*⟩} {⟨*package-name*⟩}
\PassOptionsToClass {⟨*options-list*⟩} {⟨*class-name*⟩}

这两个命令在选型的 ⟨*code*⟩ 参数中也非常有用。命令 \PassOptionsToPackage 将 ⟨*opyions-list*⟩ 中的选项名称传递给包 ⟨*package-name*⟩。这意味着它将 ⟨*options-list*⟩ 添加到任何未来的针对 ⟨*package-name*⟩ 包的 \RequirePackage 或 \usepackage 命令所使用的选项列表中。例如：

```
\PassOptionsToPackage{foo,bar}{fred}
\RequirePackage[baz]{fred}
```

其相当于：

```
\RequirePackage[foo,bar,baz]{fred}
```

相似的，\PassOptionsToClass 可以在类文件中使用，以将选项传递给另一个类，该类将通过 \LoadClass 加载。

这两个命令的效果应该和下面两个命令对比学习（在4.2中提及过）。

```
\LoadClassWithOptions
\RequirePackageWithOptions
```

\RequirePackageWithOptions 命令和 \RequirePackage 相似，但它始终以与当前类或包使用的选项列表完全相同的选项加载所需的包，而不是使用任何显示提供或通过 \PassOptionsToPackage 传递的选项。

\LoadClassWithOptions 的目的主要是允许一个类简单的基于另一个类构建，例如：

```
\LoadClassWithOptions{article}
```

这应与稍微不同的结构相比较

```
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{article}}
\ProcessOptions\relax
\LoadClass{article}
```

其效果与上面的相同，但第一个输入的字符要少得多。此外，\LoadClassWithOptions 方法运行的稍微快一些。然而，如果类声明了自己的选项，那么这两种构造是不同的，例如：

```
\DeclareOption{landscape}{\@landscapetrue}
\ProcessOptions\relax
\LoadClassWithOptions{article}
```

与

```
\DeclareOption{landscape}{\@landscapetrue}
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{article}}
\ProcessOptions\relax
\LoadClass{article}
```

相比较。在第一个例子中，当当前类使用此选项调用时，article 类将精确加载选项 landscape。相比之下，在第二个示例中，它将永远不会以选项 landscape 被调用，因为在这种情况下，文章仅通过默认选项处理程序传递选项，但该处理程序不用于 landscape，因为该选项被显示声明。

4.6 安全文件命令

这些命令处理文件输入；它们可以在请求不存在的文件时确保可以以用户友好的方式处理请求。

`\IfFileExists {`⟨*file-name*⟩`} {`⟨*true*⟩`} {`⟨*false*⟩`}`

如果文件存在，⟨*true*⟩ 中指定的代码将执行，如果文件不存在，⟨*false*⟩ 中指定的代码将执行。该命令并不会输入文件。

`\InputIfFileExists {`⟨*file-name*⟩`} {`⟨*true*⟩`} {`⟨*false*⟩`}`

如果文件 ⟨*file-name*⟩ 存在，该命令会输入该文件。在输入之前，⟨*true*⟩ 中指定的代码会执行。如果文件不存在，⟨*false*⟩ 中指定的代码会执行，其实现基于 `\IfFileExists` 命令。

4.7 报告错误等

第三方类和宏包应该由第三方使用这些命令来报告错误，或向作者提供信息。

`\ClassError {`⟨*class-name*⟩`} {`⟨*error-text*⟩`} {`⟨*help-text*⟩`}`
`\PackageError {`⟨*package-name*⟩`} {`⟨*error-text*⟩`} {`⟨*help-text*⟩`}`

这将产生一个错误信息。⟨*error-text*⟩ 将被显示，并且会出现 `?` 错误提示。如果用户输入 `h`，将会显示 ⟨*help-text*⟩。

在 ⟨*error-text*⟩ 和 ⟨*help-text*⟩ 中，`\protect` 可以用来阻止命令展开；`\MessageBreak` 会导致换行，而 `\space` 则打印一个空格。请注意，⟨*error-text*⟩ 会自动添加句号，因此无需在参数中添加句号。例如：

```
\newcommand{\foo}{FOO}
\PackageError{ethel}{%
   Your hovercraft is full of eels,\MessageBreak
   and \protect\foo\space is \foo
}{%
   Oh dear! Something's gone wrong.\MessageBreak
   \space \space Try typing \space «return»
   \space to proceed, ignoring \protect\foo.
}
```

将会显示：

```
! Package ethel Error: Your hovercraft is full of eels,
(ethel)                and \foo is FOO.

See the ethel package documentation for explanation.
```

如果用户输入 h，将会显示：

```
Oh dear! Something's gone wrong.
  Try typing  «return»  to proceed, ignoring \foo.
```

\ClassWarning {⟨*class-name*⟩} {⟨*warning-text*⟩}
\PackageWarning {⟨*package-name*⟩} {⟨*warning-text*⟩}
\ClassWarningNoLine {⟨*class-name*⟩} {⟨*warning-text*⟩}
\PackageWarningNoLine {⟨*package-name*⟩} {⟨*warning-text*⟩}
\ClassInfo {⟨*class-name*⟩} {⟨*info-text*⟩}
\PackageInfo {⟨*package-name*⟩} {⟨*info-text*⟩}

上面四个 warning 命令和 error 命令非常相似，但它们只会产生警告而没有错误提示。

当引发警告时，前两个 Warning 版本会显示行号，后两个 WarningNoLine 版本不会。

两个 Info 命令类似，只是他们仅在 log 文件中记录信息，包括行号，没有 NoLine 版本。

在 ⟨*warning-text>*⟩ 和 ⟨*info-text*⟩ 中，\protect 可以用来阻止命令展开；\MessageBreak 会导致换行，而 \space 则打印一个空格。这些命令会自动添加句号，因此无需在参数中添加句号。

5 杂项命令等

5.1 布局参数

\paperheight
\paperwidth

这两个参数用于设置纸张的尺寸,这是纸张的实际尺寸,不像是 textheight 和 \textwidth 设置的是边界内主要文字区域的尺寸。

5.2 大小写转换

\MakeUppercase {⟨*text*⟩}
\MakeLowercase {⟨*text*⟩}
\MakeTitlecase {⟨*text*⟩}

正如 userguide 中所述，文本的大小写转换应使用上面三个命令。如果需要更改程序材料的大小写，强烈建议使用 L3 编程层命令。如果您不希望这么做，则应仅在这种情况下使用 TeX 中的 \uppercase 和 \lowercase 原始语法。

5.3 更好的用户定义的数学显示环境

\ignorespacesafterend

假设您想定义一个用于显示编号为方程的文本的环境。一种简单的方法如下：

```
\newenvironment{texteqn}{%
  \begin{equation}%
    \begin{minipage}{0.9\linewidth}%
}{%
    \end{minipage}%
  \end{equation}%
}
```

然而，尝试后您会发现，这在段落中使用时并不完美，因为在环境之后的第一行开头出现了一个单词的间距，您可以通过使用 \ignorespacesafterend 来避免这个问题：

```
\newenvironment{texteqn}{%
  \begin{equation}%
    \begin{minipage}{0.9\linewidth}%
}{%
    \end{minipage}%
  \end{equation}%
  \ignorespacesafterend
}
```

这个命令也许会有其他问题。

5.4 规范化间距

`\normalsfcodes`

此命令应当用于恢复影响单词、句子等之间间距的参数的正常设置。此功能的一个重要用途是纠正一个问题，该问题由 Donald Ar-seneau 报告，即在所有已知的 TeX 格式中，每当在局部设置的间距代码生效时，页面标题中的标点符号在页面换行时总是可能出错。这些间距代码可以通过例如命令 `\frenchspacing` 和 `verbatim` 环境进行更改。它通常在 `\begin{document}` 中自动给出正确的定义，因此不需要显式设置；然而，如果在类文件中显式设置为非空，则自动默认设置将被覆盖。

5.5 查询本地化

本地化信息是定制一系列输出所需的。LaTeX 内核本身不管理本地化，这由 babel 和 polyglossia 套件很好地处理。为了允许内核和其他软件包访问 babel 或 polyglossia 提供的当前本地化信息，内核定义了命令 `\\BCPdata` 。初始内核定义扩展为 `en-US` 的标签部分，因为内核不跟踪本地化，但确实以广泛的美式英语设置开始。然而，如果加载了 babel 或 polyglossia，它将重新定义为适当软件包的 BCP-47 信息。支持的参数是 BCP-47 标签的细分：

1. tag 全部的 BCP-47 标签（如 en-US）
2. language（如 de）
3. region（如 AT）
4. script（如 Latn）
5. variant（如 1901）
6. extension.t（转换，如 en-t-ja）
7. extension.u（附加区域信息，如 ar-u-nu-latn）
8. extension.x（私用区域，如 la-x-classic）

如果这些以 *main* 为前缀，则将提供文档的主要语言信息，例如 `main.language` 将扩展为主要语言，即使当前激活的是另一种语言。除了标签细分外，支持以下语义参数

1. `casing` The tag for case changing, e.g. `el-x-iota` could be selected rather than `el` to select a capital adscript iota on uppercasing an *ypogegrammeni*

例如，大小写更改命令 `\\MakeUppercase` （概念上）定义为

```
\ExpandArgs{e}\MakeUppercaseAux{\BCPdata{casing}}{#1}
```

其中 `#1` 是用户输入，`\MakeUppercaseAux` 的第一个参数接受两个参数，区域和输入文本。

5.6 属性的扩展和可扩展引用

属性是 LaTeX 在处理文档时可以跟踪的内容，例如页码、标题编号、其他计数值、标题名称、页面上的位置等。这些属性的当前值（value）可以被标记并写入 aux 文件。然后可以在下次编译时引用，类似于标准的 `\label/\ref` 命令的工作方式（它们记录/引用一组固定的属性：标签、页码、标题和目标）。

`\RecordProperties{`⟨*label*⟩`}{`⟨*list of properties*⟩`}`

此命令写入由 ⟨*label*⟩ 标记的 aux 文件中的 ⟨*properties*⟩ 的值。记录的是在 `\RecordProperties` 被调用时的当前值，或者在下一个输出发生时的当前值——这取决于每个属性的声明。参数 ⟨*label*⟩ 和 ⟨*properties*⟩ 可以包含被扩展的命令。⟨*label*⟩ 可以扩展为任意字符串（只要它可以安全地写入 aux 文件），但请注意，⟨*label*⟩ 和 `\RecordProperties` 的标签名称共享一个命名空间。这意味着您会在以下代码中收到一个标签“A”被多重定义的警告：

```
\label{A}\RecordProperties{A}{abspage}
```

`\RefProperty{`⟨*label*⟩`}{`⟨*property*⟩`}`

此命令允许引用在上一次运行中记录并由 ⟨*label*⟩ 标记的属性的值。与标准 `\ref` 命令不同。该命令是可扩展的，例如值可以--如果是一个数字--在任务中被使用。[3]

```
\section{A section}
\RecordProperties{mylabel}{pagenum,counter}
\RefProperty{mylabel}{counter} % outputs section
\setcounter{mycounter}{\RefProperty{mylabel}{pagenum}}
```

由于 `\\RefProperty` 是可扩展的，如果未找到标签，它无法发出重新运行警告。如果需要，可以通过以下命令强制发出此类警告：

`\RefUndefinedWarn{`⟨*label*⟩`}{`⟨*property*⟩`}`

LaTeX 预定义了一组属性，这组属性也包含标准 `\label` 命令存储的属性。在下面的列表中，“default” 表示当值尚未知晓时返回的值（即，如果在上一次运行中未记录）。“at shipout” 意味着该属性在使用 `\RecordProperties` 时并未立即记录，而是在下一个 `\\shipout` 时记录。

[3]要实现这一点，属性的默认值也需要是一个数字，因为记录的值在第一次 LaTeX 运行中是未知的。

abspage (default: 0, at shipout) 当前页面的绝对值：从 1 开始，并在每个 shipout 单调增加。

page (default: 0, at shipout) 被 \thepage 给出的当前页面：这可能是一个数值，也可能不是，具体取决于当前样式。与 abspage 对比。您还可以通过标准的标 \label/\pageref 获取此值。

pagenum (default: 0, at shipout) 当前页码，以阿拉伯数字表示。这适用于整数运算和比较。

label (default: ??) \@currentlabel 的内容，您可以通过标准的标 \label/\ref 获取此值。

title (default: \textbf{??}) \@currentlabelname 的内容。此命令与其他命令一起由 nameref 包和某些类（例如 memoir）填充，通常给出文档中由某些分节命令定义的标题。

target (default: ⟨*empty*⟩) \@currentHref 的内容。该命令通常由 hyperref 填充，并保存它创建的最后一个目标的名称。

pagetarget (default: ⟨*empty*⟩, at shipout) \@currentHpage 的内容，此命令由 hyperref（版本 v7.01c 或更高版本）填充，并保存它创建的最后一个页面锚点的名称。

counter (default: ⟨*empty*⟩) \@currentcounter 的内容，该命令包含 \refstepcounter 之后的计数器名称。

xpos, ypos (default: 0, at shipout) 这些属性记录了先前 \pdfsavepos/\savepos 存储的点的 x 和 y 坐标。例如（如果使用 bidi，则需要保存标签前后的位置）：

```
\pdfsavepos
\RecordProperties{myposition}{xpos,ypos}%
\pdfsavepos
```

类和包的作者可以定义更多的属性来存储他们感兴趣的其他值。

```
\NewProperty{⟨name⟩}{⟨setpoint⟩}{⟨default⟩}{⟨code⟩}
\SetProperty{⟨name⟩}{⟨setpoint⟩}{⟨default⟩}{⟨code⟩}
```

这些命令可以声明或改变一个属性的 ⟨*name*⟩[4]。如果在包中声明了一个新属性，则建议其名称的结构始终如下：⟨*package-name*⟩/⟨*property-name*⟩。⟨*setpoint*⟩ 是 now 或 shipout，

[4]仅更改已声明的属性。LaTeX 的标准属性和其他包的性能声明不应该改变！

决定该值是直接写入还是在下一个 shipout 时写入。⟨*default*⟩ 用于如果属性被引用但还不知道时，例如，在第一次运行时。⟨*code*⟩ 是在存储值时执行的代码。例如，pagenum 属性被声明为

```
\NewProperty{pagenum}{shipout}{0}{\the\value{page}}
```

与属性相关的命令以一组 CamelCase 命令的形式提供，适用于传统的LaTeX 2ε包（如有需要，也可在文档前言中使用），以及适用于现代包的 expl3 命令，这些命令使用 LaTeX 的 L3 编程层。expl3 命令及更多细节可以在 ltproperties-doc.pdf 中找到。

5.7 准备链接目标

文档中的活动链接需要跳转的目标。这些目标通常由 \refstepcounter 命令自动创建（如果加载了 hyperref 包），但在某些情况下，类或包的作者需要手动添加目标，例如，在无编号的章节命令或环境中。为此，LaTeX 提供了以下命令。在没有 hyperref 的情况下，它们不执行任何操作或仅插入一个占位符（以确保在加载 hyperref 时间距不变），而在有 hyperref 的情况下，它们会添加必要的目标。关于以下命令的行为和参数的详细信息可以在 hyperref 包的 ltproperties-doc.pdf 中找到。

\MakeLinkTarget[⟨*prefix*⟩]{⟨*counter*⟩}
\MakeLinkTarget[⟨*prefix*⟩]{}
\MakeLinkTarget*{⟨*target name*⟩}

这些命令准备创建目标。

\LinkTargetOn
\LinkTargetOff

这些命令允许在局部启用或禁用目标创建，这对于抑制通过 \refstepcounter 自动创建的目标非常有用。

\NextLinkTarget{⟨*target name*⟩}

这将更改要创建的下一个目标的名称。

6 被新材料取代的命令

在 1990 年代中期，作为LaTeX 2ε一部分引入了一小部分命令，这些命令被广泛使用，但已被更现代的方法所取代。这里涵盖这些命令，因为在现有的类和包中，可能会经常遇

到它们。

6.1 定义命令

这些命令的 * 形式应该用于定义在 TeX 术语中不长的命令。这对于使用参数不打算包含整个文本段落的命令的错误捕获很有用。

\DeclareRobustCommand {⟨*cmd*⟩} [⟨*num*⟩] [⟨*default*⟩] {⟨*definition*⟩}
\DeclareRobustCommand* {⟨*cmd*⟩} [⟨*num*⟩] [⟨*default*⟩] {⟨*definition*⟩}

该命令接受与 \newcommand 相同的参数，但它声明一个强健的命令，即使在 ⟨*definition*⟩ 中的一些代码是脆弱的。您可以使用此命令定义新的强健命令，或重新定义现有命令并使其强健。如果一个命令被重新定义，则会在转录文件中记录日志。

例如，如果 \seq 被定义如下：

```
\DeclareRobustCommand{\seq}[2][n]{%
  \ifmmode
    #1_{1}\ldots#1_{#2}%
  \else
    \PackageWarning{fred}{You can't use \protect\seq\space in text}%
  \fi
}
```

然后命令 \seq 可以在移动参数中使用，即使 \ifmmode 不能，例如：

```
\section{Stuff about sequences $\seq{x}$}
```

还要注意，在定义的开头没有必要在 \ifmmode 前放置 \relax；这是因为这个 \relax 提供的防护在错误的时间扩展将会在内部提供。

\CheckCommand {⟨*cmd*⟩} [⟨*num*⟩] [⟨*default*⟩] {⟨*definition*⟩}
\CheckCommand* {⟨*cmd*⟩} [⟨*num*⟩] [⟨*default*⟩] {⟨*definition*⟩}

这接受与 \newcommand 相同的参数，但它并不是定义 ⟨*cmd*⟩，而只是检查当前的 ⟨*cmd*⟩ 定义是否与 ⟨*definition*⟩ 完全相同。如果这些定义不同，则会引发错误。

该命令对于在您的包开始更改命令的定义之前检查系统的状态是有用的。它允许您特别检查没有其他包重新定义相同的命令。

6.2 选项声明

以下命令处理文档类和包的选项声明和处理，采用经典的“简单文本”方法。每个选项名称必须是“LaTeX 名称”。

有一些命令专门设计用于这些命令的 ⟨*code*⟩ 参数中（如下）。

`\DeclareOption {`⟨*option-name*⟩`} {`⟨*code*⟩`}`

这使得 ⟨*option-name*⟩ 成为其所在类或包的“声明选项”。

⟨*code*⟩ 参数包含在为类或包指定该选项时要执行的代码;它可以包含任何有效的LaTeX 2ε构造。例如：

```
\DeclareOption{twoside}{\@twosidetrue}
```

`\DeclareOption* {`⟨*code*⟩`}`

这声明了 ⟨*code*⟩ 将为每个指定但未明确声明的选项执行；该代码称为“默认选项代码”，它可以包含任何有效的LaTeX 2ε构造。

如果类文件不包含 `\DeclareOption*`，则默认情况下，所有指定但未声明的选项将被静默传递给所有包（指定和声明的选项也将如此）。

如果包文件不包含 `\DeclareOption*`，则默认情况下，每个指定但未声明的选项将产生错误。

6.3 选项代码中的命令

这里有两个命令仅能用在 `\DeclareOption` 或 `\DeclareOption*` 的 ⟨*code*⟩ 参数中。在接下来的几小节中，可以找到这些参数中常用的其他命令。

`\CurrentOption`

该命令用于扩展当前选项的名称。

`\OptionNotUsed`

这会导致当前选项被添加到“未使用选项”的列表中。

6.4 选项处理

`\ProcessOptions`

此命令为每个选中的选型执行 ⟨*code*⟩。

我们将首先描述 `\ProcessOptions` 如何在包文件中工作，然后描述它在类文件中如何不同。

要详细了解 `\ProcessOptions` 在包文件中的作用，您必须了解局部选项和全局选项之间的区别。

- **Local options** 是在任何 ⟨*options*⟩ 的参数中为这个特定包明确指定的选项：

  ```
  \PassOptionsToPackage{<options>} \usepackage[<options>]
  \RequirePackage[<options>]
  ```

- **Global options** `\documentclass[<options>]` 的 ⟨*options*⟩ 参数中指定的任何其他选项。

例如，假设一个文档开始于：

```
\documentclass[german,twocolumn]{article}
\usepackage{gerhardt}
```

同时，包 `gerhardt` 调用包 `freq`：

```
\PassOptionsToPackage{german,dvips,a4paper}{fred}
\RequirePackage[errorshow]{fred}
```

然后：

- `fred` 的局部选项是 `german`、`dvips`、`a4paper` 和 `errorshow`
- `fred` 只有一个全局选项是 `twocolumn`。

当 `\ProcessOptions` 被调用时，会发生：

1. 首先，对于 `fred.sty` 中通过 `\DeclareOption` 所声明的每个选项，它将查看该选项是全局选项还是局部选项并执行相应的代码。这是按照在 `fred.sty` 中声明这些选项的顺序来完成的

2. 然后，对于每个余下的局部选项，如果命令 \ds@<option> 会被执行，如果他被定义在某个地方（不是通过 DeclareOption）。否则，将执行“默认选项代码”。如果没有声明任何默认选项代码，则会产生一条错误消息。这是按照指定这些选项的顺序来完成的。

在整个过程中，系统会确保为一个选项声明的代码最多只执行一次。

返回到这个示例，如果 fred.sty 包含：

```
\DeclareOption{dvips}{\typeout{DVIPS}}
\DeclareOption{german}{\typeout{GERMAN}}
\DeclareOption{french}{\typeout{FRENCH}}
\DeclareOption*{\PackageWarning{fred}{Unknown ‘\CurrentOption’}}
\ProcessOptions\relax
```

那么，处理此文档的结果将是：

```
DVIPS
GERMAN
Package fred Warning: Unknown ‘a4paper’.
Package fred Warning: Unknown ‘errorshow’.
```

需要注意以下几点：

1. dvips 选项的代码在 german 选项的代码之前执行，因为这是在 fred.sty 中声明它们的顺序
2. 当声明的选项被处理时，german 选项的代码只执行一次。
3. a4paper 和 errorshow 选项从 \DeclareOption* 声明的代码（按照它们指定的顺序）产生警告，而 twocolumn 选项不会：这是因为 twocolumn 是一个全局选项。

在类文件中,\ProcessOptions 的工作方式相同,除了:所有选项均为局部选项,\DeclareOption* 的默认值是 \OptionNotUsed 而不是错误。

请注意，因为 \ProcessOptions 有 * 形式，所以最好遵循 \relax 的非星形格式，就像前面的示例一样，因为这可以防止发出不必要的和可能产生误导性的错误消息。

\ProcessOptions*

该命令就像 \ProcessOptions，但它按照调用命令中指定的顺序执行选项，而不是按照类或包中的声明顺序执行选项。对于一个包，这意味着要首先处理全局选项

`\ExecuteOptions {⟨options-list⟩}`

它可以用于在 `\ProcessOptions` 之前提供一个“默认选项列表”。例如，假设在一个类文件中，您希望将默认设计设置为：双面打印、11pt 字体；分两列。然后可以指定：

```
\ExecuteOptions{11pt,twoside,twocolumn}
```