面向宏包与类文件编写者的 LaTeX ——当前版本

张泓知　　翻译

2023 年 10 月 24 日

目　录

1 介绍

LaTeX 2$_\varepsilon$于 1994 年发布，为 LaTeX 引入了一些新概念。这些概念在 `clsguide-historic` 中为宏包和类的作者进行了描述，该文档在很大程度上保持不变。此后，LaTeX 团队致力于多个想法：首先是为 LaTeX 设计的编程语言（L3 编程层），然后是一系列建立在该语言基础上的作者工具。在这里，我们概述了由 LaTeX 核心提供给宏包和类开发者的当前稳定工具集。我们假设读者作为文档作者熟悉一般的 LaTeX 用法，并建议阅读本文与 `usrguide` 结合，后者提供给一般 LaTeX 用户有关创建命令等当代方法的信息。

2 编写类和宏包

本节讨论了编写 LaTeX 类和宏包的基本要点。

2.1 是类还是宏包?

当您想在文件中添加一些新的 LaTeX 命令时，首先要决定它是一个*文档类*还是一个*宏包*。经验法则是:

> 如果这些命令可以与任何文档类一起使用，那么将它们制作为一个宏包；否则，将其制作为一个类。

类有两种主要类型：像 `article`、`report` 或 `letter` 这样的独立类；以及扩展或变体其他类的类，比如构建在 `article` 文档类之上的 `proc` 文档类。

因此，一个公司可能会有一个用于打印带有自己抬头纸的信件的本地 `ownlet` 类。这样的类将在现有的 `letter` 类的基础上构建，但它不能与其他任何文档类一起使用，因此我们有了 `ownlet.cls` 而不是 `ownlet.sty`。

相比之下，`graphics` 宏包提供了将图像包含到 LaTeX 文档中的命令。由于这些命令可以与任何文档类一起使用，我们有了 `graphics.sty` 而不是 `graphics.cls`。

2.2 使用 'docstrip' 和 'doc'

如果您打算为 LaTeX 编写一个大型的文档类或宏包，您应该考虑使用随 LaTeX 一起提供的 `doc` 软件。使用 `doc` 编写的 LaTeX 文档类和宏包可以以两种方式进行处理：它们可以通过 LaTeX 运行以生成文档；也可以通过 `docstrip` 运行以生成 `.cls` 或 `.sty` 文件。

`doc` 软件可以自动生成定义索引、命令使用索引和变更记录列表。它对于维护和记录大型 TeX 源文件非常有用。

LaTeX 核心本身以及标准文档类等的文档化源代码都是 `doc` 文档；它们位于分发中的 `.dtx` 文件中。实际上，您可以通过在 `source2e.tex` 上运行 LaTeX 来排版核心源代码作为一个长文档，包括索引。排版这些文档使用了类文件 `ltxdoc.cls`。

有关 `doc` 和 `docstrip` 的更多信息，请参阅文件 `docstrip.dtx`、`doc.dtx` 和 *The LaTeX Companion*。要了解其用法示例，请查看 `.dtx` 文件。

2.3 标准文档类的政策

我们收到的有关标准文档类的问题报告中，很多并不涉及错误，而是更或多或少地礼貌地暗示这些类中体现的设计决策“不够优秀”，并要求我们对其进行修改。

有几个原因我们不应该对这些文件进行此类更改：

- 不管多么误导，当前的行为显然是设计这些类时所预期的。
- 更改“标准类”的这些方面并不是一个好的做法，因为许多人将依赖于它们。

因此，我们决定甚至不考虑进行这种修改，也不花时间来证明这个决定。这并不意味着我们不同意这些类的设计存在许多缺陷，但是我们有许多更重要的任务，而不是不断解释为什么 LaTeX 的标准文档类不能改变。

当然，我们欢迎更好的类的产生，或者用于增强这些类的包的出现。因此，当您考虑到这样的不足时，我们希望您首先的想法是“我能做些什么来改善这个?”

2.4 命令名称

在介绍下一节中描述的 L3 编程层引入之前，LaTeX 有三种类型的命令。

第一种是作者命令，比如 `\section`、`\emph` 和 `\times`：这些命令通常有简短的名称，全部是小写。

第二种是类和宏包编写者命令：这些命令通常有长的混合大小写名称，例如以下几个。

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

最后一种是用于 LaTeX 实现的内部命令，比如 `\@tempcnta`、`\@ifnextchar` 和 `\@eha`：其中大多数命令的名称中包含 `@`，这意味着它们不能在文档中使用，只能在类和宏包文件中使用。

不幸的是，出于历史原因，这些命令之间的区别通常很模糊。例如，`\hbox` 是一个内部命令，只应在 LaTeX 核心中使用，而 `\m@ne` 是常数 -1，本来可以命名为 `\MinusOne`。

然而，这个经验法则仍然有用：如果一个命令的名称中包含 `@`，那么它不是支持的 LaTeX 语言的一部分——它的行为可能在未来版本中改变！如果一个命令是混合大小写的，或者在 *LaTeX: A Document Preparation System* 中有描述，那么您可以依赖于未来版本的 LaTeX 支持该命令。

2.5 编程支持

正如在介绍中所指出的，LaTeX 核心今天加载了来自编程的专用支持，这里称为 L3 编程层，通常也称为 expl3。L3 编程层的一般方法的详细信息在文档 `expl3` 中给出，而当前所有代码接口的参考可以在 `interface3` 中找到。这个层包含两种类型的命令：一个由 API 组成的文档化命令集，和大量的私有内部命令。后者都以两个下划线开头，不应在定义它们的代码模块之外使用。这种更有结构的方法意味着使用 L3 编程层不会像使用 '`@` 命令' 那样受到上面提到的某种程度上的不稳定情况的影响。

在这里我们不涉及使用 L3 编程层的细节。关于该方法的良好介绍可在 https://www.alanshawn.com/latex3-tutorial/ 找到。

2.6 盒子命令和颜色

即使您不打算在自己的文档中使用颜色，通过注意本节中的要点，您可以确保您的文档类或宏包与 `color` 宏包兼容。这可能有利于使用您的文档类或宏包并希望使用颜色的人。

确保“颜色安全”的最简单方法是始终使用 LaTeX 的盒子命令而不是 TeX 的原始命令，即使用 `\sbox` 而不是 `\setbox`，`\mbox` 而不是 `\hbox`，`\parbox` 或 `minipage` 环境而不是 `\vbox`。现在，LaTeX 的盒子命令具有新的选项，使其与 TeX 的原始命令一样强大。

举例说明可能出现的问题，考虑 `{\ttfamily <text>}` 中字体恢复的时间点在最后的 `}` *之前*，而在类似的结构 `{\color{green} <text>}` 中颜色恢复的时间点在最后的 `}` *之后*。通常，这种区别并不重要；但是考虑一个类似于原始 TeX 盒子赋值的命令，比如：

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

现在，颜色恢复在 `}` 之后发生，因此*不会*存储在盒子中。这可能会产生的坏影响取决于颜色的实现方式：从文档其余部分出现错误的颜色到导致用于打印文档的 dvi 驱动程序出现错误。

另一个有趣的命令是 \normalcolor。通常它只是 \relax（即什么也不做），但您可以像使用 \normalfont 那样将其用于设置页面上的区域，比如标题或节标题的“主文档颜色”。

2.7 通用风格

LaTeX 提供了许多命令，旨在帮助您生成结构良好、健壮且可移植的文档类和宏包文件。本节概述了一些使用这些命令的智能方式。

2.7.1 加载其他文件

LaTeX 提供了以下命令：

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

用于在其他文档类或宏包中使用类或宏包。我们强烈建议您使用这些命令，而不是原始的 \input 命令，原因有几个。

使用 \input <filename> 加载的文件不会列在 \listfiles 列表中。

如果一个宏包始终使用 \RequirePackage... 或 \usepackage 加载，即使请求多次加载，它也只会被加载一次。相比之下，如果使用 \input 加载，则可以加载多次；多余的加载可能会浪费时间和内存，并且可能会产生奇怪的结果。

如果一个宏包提供选项处理，那么如果不是使用 \usepackage 或 \RequirePackage... 加载，而是使用 \input，则可能会产生奇怪的结果。

如果宏包 foo.sty 使用 \input baz.sty 加载了宏包 baz.sty，则用户会得到警告：

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

因此，由于几个原因，使用 \input 加载宏包并不是一个好主意。

例如，article.sty 只包含以下几行：

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

您可能希望做相同的事情，或者如果认为安全的话，可以决定移除 myclass.sty。

2.7.2 使它健壮

在编写宏包和文档类时，我们认为尽可能使用 LaTeX 命令是一种良好的做法。

因此,我们建议在编程和定义文档接口时,使用 `\newcommand`、`\renewcommand` 或 `\providecommand`，以及 `\NewDocumentCommand` 等命令（有关这些命令的详细信息，请参阅 `usrguide`）。

当定义环境时，请使用 `\NewDocumentEnvironment` 等命令（对于简单情况也可以使用 `\newenvironment` 等），而不是使用 `\def\foo{...}` 和 `\def\endfoo{...}`。

如果需要设置或更改 ⟨*dimen*⟩ 或 ⟨*skip*⟩ 寄存器的值，请使用 `\setlength`。

要操作盒子，请使用诸如 `\sbox`、`\mbox` 和 `\parbox` 等 LaTeX 命令，而不是 `\setbox`、`\hbox` 和 `\vbox`。

使用 `\PackageError`、`\PackageWarning` 或 `\PackageInfo`（或相应的类命令），而不是 `\@latexerr`、`\@warning` 或 `\wlog`。

这种做法的优点是您的代码更易读，并且更容易让其他有经验的 LaTeX 程序员理解。

2.7.3 增强可移植性

使您的文件尽可能具有可移植性也是明智的。为了确保这一点，文件不能与标准 LaTeX 发行版中的文件同名，即使其内容可能与这些文件之一非常相似也不行。而且，最好仍然只使用 ASCII 范围内的文件名：尽管 LaTeX 原生支持 UTF-8，但并不能肯定所有工具都支持。出于同样的原因，避免文件名中出现空格。

如果本地的文档类或宏包有一个共同的前缀，例如 Nowhere 大学的类可能以 `unw` 开头。这有助于避免每个大学都拥有自己的名为 `thesis.cls` 的论文类。

如果您依赖于 LaTeX 核心的某些功能，或者依赖于某个宏包，请指定您所需的发布日期。例如，错误处理宏包是在 2022 年 6 月的发布中引入的，所以如果您使用它们，应该加上：

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

2.7.4 有用的钩子

有时候，一个宏包需要安排在导言区的开头或结尾、文档末尾或每次使用某个环境的开头执行一些代码。这可以通过使用钩子来实现。作为文档作者，您可能熟悉 `\AtBeginDocument`，它是一个对更强大的 `\AddToHook` 的包装。LaTeX 核心提供了大量专用钩子（应用于预定义位置）和通用钩子（适用于任意命令）：使用这些钩子的接口在 `lthooks` 中有描述。还有用于文件的钩子，在 `ltfilehooks` 中有描述。

3 文档类或宏包的结构

文档类或宏包文件的大纲如下：

标识 文件声明自己是一个 LaTeX 2_ε 宏包或文档类，并简要描述自身。

初步声明 在这里，文件声明一些命令，并可以加载其他文件。通常，这些命令将只是声明所用选项中需要的代码。

选项 文件声明并处理其选项。

更多声明 这是文件执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 标识

文档类或宏包文件的第一件事是标识自己。宏包文件的标识如下：

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

例如：

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

文档类文件的标识如下：

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

例如：

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

⟨*date*⟩ 应以 'YYYY-MM-DD' 的形式给出，如果使用了可选参数，必须存在（这对于 `\NeedsTeXFormat` 命令也是如此）。任何偏离此语法的情况都会导致低级别的 TEX 错误——这些命令期望有效的语法以加快宏包或文档类的日常使用，并不考虑开发者犯错的情况！

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

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

那么在不同位置的用户会收到警告，指出他们的 `article` 副本已过时。

当使用类时，类的描述会显示出来。宏包的描述会被放入日志文件中。这些描述也会被 `\listfiles` 命令显示。短语 `Standard LaTeX` **绝对不应**出现在除标准 LATEX 发行版之外的任何文件的标识中。

3.2 使用类和宏包

一个 LATEX 宏包或文档类可以加载另一个宏包，方法如下：

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

例如：

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

此命令与作者命令 `\usepackage` 具有相同的语法。它允许宏包或文档类使用其他宏包提供的功能。例如，通过加载 `ifthen` 宏包，宏包作者可以使用该宏包提供的“if...then...else...”命令。

一个 LATEX 文档类可以加载另一个类，方法如下：

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

例如：

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

此命令与作者命令 `\documentclass` 具有相同的语法。它允许类基于另一个类的语法和外观。例如，通过加载 `article` 类，类作者只需更改他们不喜欢的 `article` 的部分，而不是从头开始编写一个新的类。

以下命令可以在常见情况下使用，即您希望简单地加载一个具有当前类所使用的确切选项的类或宏包文件。

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

例如：

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

3.3 声明选项

宏包和文档类可以声明选项，作者可以指定这些选项；例如，`twocolumn` 选项由 `article` 类声明。注意，选项的名称应仅包含“LaTeX 名称”中允许的字符；特别地，不能包含任何控制序列。

LaTeX 支持两种创建选项的方法：键-值系统和“简单文本”方式。键-值系统推荐用于新的类和宏包，并且在处理选项类方面比简单文本方式更灵活。这两种选项方法在 LaTeX 源文件中使用相同的基本结构：首先声明选项，然后在第二步处理选项。两者都允许将选项传递给其他宏包或底层类。由于“经典”的简单文本方式在概念上更直观，用于展示一般结构；详细的键-值方法请参阅第 4.4 节。

声明一个选项如下：

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

例如，`graphics` 宏包中的 `dvips` 选项（略作简化）的实现如下：

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

这意味着当作者写 \usepackage[dvips]{graphics} 时，文件 dvips.def 会被加载。再举个例子，article 类中声明了 a4paper 选项以设置 \paperheight 和 \paperwidth 长度：

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

有时用户会请求类或宏包未明确声明的选项。默认情况下，这将产生警告（对于类）或错误（对于宏包）；此行为可以通过以下方式更改：

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

例如，为使宏包 fred 对未知选项产生警告而不是错误，您可以指定：

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

这样，如果作者写了 \usepackage[foo]{fred}，他们会收到警告 Package fred Warning: Unknown option `foo'. 再举个例子，fontenc 宏包在使用 ⟨*ENC*⟩ 选项时会尝试加载文件 <ENC>enc.def。这可以通过如下方式完成：

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

可以使用 \PassOptionsToPackage 或 \PassOptionsToClass 命令将选项传递给另一个宏包或类（请注意，这是一种专门的操作，仅适用于选项名称）：请参阅第 4.5 节。例如，要将每个未知选项传递给 article 类，可以使用：

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

如果这样做，您应该确保在稍后某个时刻加载该类，否则选项将永远不会被处理！

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。要处理文件调用时使用的选项，应使用：

```
\ProcessOptions\relax
```

这将为每个已指定和已声明的选项执行相应的 ⟨*code*⟩（请参阅第 6.4 节以了解详细信息）。

例如，如果 `jane` 宏包文件包含：

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

并且作者写了 `\usepackage[foo,bar]{jane}`，那么他们将看到消息 `Saw foo.` 和 `What's bar?`。

3.4 一个最小的文档类文件

一个文档类或宏包的大部分工作在于定义新命令或更改文档的外观。这是在文档类的主体中完成的，使用诸如 `\newcommand` 或 `\setlength` 等命令。

每个文档类文件*必须*包含四个内容：定义 `\normalsize`、设置 `\textwidth` 和 `\textheight` 的值，以及指定页码。因此，一个最小的文档类文件[1] 看起来像这样：

```
\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}       % 即使这个类不显示页码也需要
```

然而，这个类文件不支持脚注、边注、浮动对象等，也不提供像 `\rm` 这样的两个字母的字体命令；因此，大多数文档类会包含比这更多的内容！

3.5 示例：一个本地信函类

一家公司可能有自己的信函类，用于按公司风格设置信函。本节展示了这样一个类的简单实现，实际的类需要更多的结构。

该类首先以 `neplet.cls` 的身份宣布自己。

[1] 这个类现在已经在标准发行版中，名为 `minimal.cls`。

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

然后，下面的部分将任何选项传递给 `letter` 类，并使用 `a4paper` 选项加载它。

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

为了使用公司信头，重新定义了 `firstpage` 页样式：这是用于信函第一页的页样式。

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

就是这样！

3.6 示例：一个新闻简报类

可以使用 LaTeX 来排版简单的新闻简报，使用的是 `article` 类的变体。该类首先以 `smplnews.cls` 的身份宣布自己。

```
\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 `onecolumn` 选项被关闭，`green` 选项将标题设置为绿色。

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

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

\ProcessOptions\relax
```

然后加载 article 类，使用 twocolumn 选项。

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

由于新闻简报将以彩色打印，它现在加载 color 宏包。该类不指定设备驱动程序选项，因为这应由 smplnews 类的使用者指定。

```
\RequirePackage{color}
```

然后重新定义 \maketitle，以 72 pt Helvetica 粗斜体显示标题，使用合适的颜色。

```
\renewcommand{\maketitle}{%
   \twocolumn[%
      \fontsize{72}{80}\fontfamily{phv}\fontseries{b}%
      \fontshape{sl}\selectfont\headlinecolor
      \@title
   ]%
}
```

它重新定义了 \section 并关闭了章节编号。

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

\setcounter{secnumdepth}{0}
```

它还设置了三个基本要素。

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

在实践中，一个类需要比这更多：它将提供用于期号、文章作者、页面样式等的命令；但是这个框架已经提供了一个开始。ltnews 类文件和这个类文件相比也没有更复杂多少。

4 用于类和宏包编写者的命令

本节简要描述了用于类和宏包编写者的每个命令。要了解系统的其他方面，您还应阅读 *LaTeX: A Document Preparation System*、*The LaTeX Companion* 和 *LaTeX 2ε for Authors*。

4.1 识别

首先讨论的一组命令是用于识别您的类或宏包文件的命令。

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

该命令告诉 TeX 应使用名称为 ⟨*format-name*⟩ 的格式处理此文件。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的日期时，将生成警告。标准的 ⟨*format-name*⟩ 是 `LaTeX2e`。如果存在日期，则必须采用 YYYY-MM-DD 的形式。

示例：

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

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

此声明当前文件包含文档类 ⟨*class-name*⟩ 或宏包 ⟨*package-name*⟩ 的定义。

可选的 ⟨*release-info*⟩，如果使用，必须包含：

- 此版本文件的发布日期，形式为 YYYY-MM-DD；
- 可选地跟着一个空格和一个简短描述，可能包括版本号。

上述语法必须严格遵循，以便 `\LoadClass` 或 `\documentclass`（对于类）或 `\RequirePackage` 或 `\usepackage`（对于宏包）测试发布是否太旧。

整个 ⟨*release-info*⟩ 信息将由 `\listfiles` 显示，因此不应太长。

示例：

```
\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⟩]`

这类似于前两个命令，只是这里必须给出完整的文件名，包括扩展名。用于声明除主类和宏包文件以外的任何文件。

示例：

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

请注意，在除了标准 LaTeX 发行版之外的任何文件的标识横幅中，`Standard LaTeX` **绝不能**使用。

4.2 加载文件

这组命令可用于通过基于现有类或宏包来创建自己的文档类或宏包。

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

宏包和文档类应使用这些命令加载其他宏包。

使用 `\RequirePackage` 与作者命令 `\usepackage` 相同。示例：

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

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

这些命令仅用于类文件中，不能用于宏包文件；在类文件中最多只能使用一次。

使用 `\LoadClass` 与使用 `\documentclass` 加载类文件相同。

示例：

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

两个带 `WithOptions` 的版本仅加载具有当前文件（类或宏包）使用的选项的类（或宏包）文件。有关其用法的进一步讨论，请参见下文，4.5 节。

4.3 延迟代码

如前所述，lthooks 中提供了一个复杂的钩子系统。这里我们记录了一小组常见钩子的便捷简短名称。

这前两个命令也主要用于 \DeclareOption 或 \DeclareOption* 的 ⟨*code*⟩ 参数内。

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

这些命令声明了 ⟨*code*⟩，会在整个当前类或宏包文件处理完毕后保存并执行。

允许重复使用这些命令：参数中的代码按声明顺序存储（并后续执行）。

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

这些命令声明了将在 LaTeX 执行 \begin{document} 或 \end{document} 期间保存并执行的 ⟨*code*⟩。

\AtBeginDocument 中的参数指定的 ⟨*code*⟩ 在 \begin{document} 代码的末尾执行，*在*设置字体选择表之后。因此，这是放置需要在所有内容准备好进行排版，并且正常字体为当前字体时执行的代码的有用位置。

\AtBeginDocument 钩子不应用于进行任何排版，因为排版结果将是不可预测的。

\AtEndDocument 中的参数指定的 ⟨*code*⟩ 在 \end{document} 代码的开头执行，*在*最终页面完成之前，也在处理任何剩余的浮动环境之前。如果某些 ⟨*code*⟩ 需要在这两个过程之后执行，应在 ⟨*code*⟩ 的适当位置包含 \clearpage。

允许重复使用这些命令：参数中的代码按声明顺序存储（并后续执行）。

4.4 创建和使用键值选项

与任何键-值输入一样，使用键-值对作为宏包或类选项有两部分：创建键选项和设置（使用）它们。以这种方式创建的选项 *可能*作为一般键-值设置在加载宏包后使用：这将取决于底层代码的性质。

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

此命令从逗号分隔的 ⟨*declarations*⟩ 列表中创建一系列选项。列表中的每个条目都是键-值对，其中 ⟨*key*⟩ 具有一个或多个 ⟨*properties*⟩。下面介绍了一小部分“基本”的 ⟨*properties*⟩。由 l3keys 提供的完整范围的属性也可用于更强大的处理。有关完整细节，请参见 interface3。

这里提供的基本属性包括：

- .code — 执行任意代码
- .if — 设置一个 TEX \if... 开关
- .ifnot — 设置一个反转的 TEX \if... 开关
- .store — 将值存储到宏中
- .usage – 定义选项是否仅能在加载时使用（load）、在导言区使用（preamble）或在范围上没有限制（general）

在 ⟨*property*⟩ 之前的 ⟨*key*⟩ 部分是 ⟨*name*⟩，⟨*value*⟩ 与 ⟨*property*⟩ 一起定义了选项的行为。

例如，使用以下声明：

```
\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 可以在导言区的任何位置给出，并将设置一个名为 \if@mypkg@draft 的开关。选项 name 只能在加载宏包时给出，并将保存给定的任何值到 \@mypkg@name 中。最后，选项 second-name 可以在任何位置给出，并将其值保存到 \@mypkg@other@name 中。

在使用 \ProcessKeyOptions 之前创建的键将作为宏包选项。

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

命令 \DeclareUnknownKeyHandler 用于定义遇到未定义键时的行为。参数 ⟨*code*⟩ 将接收未知键名为 #1，值为 #2。然后可以根据需要处理它们，例如将其转发到另一个宏包。整个选项作为 \CurrentOption 可用，如果需要传递可能包含 = 符号的选项。例如，这可用于将未知选项传递给 article 等非键值类：

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

\ProcessKeyOptions [⟨*family*⟩]

函数 \ProcessKeyOptions 用于检查当前选项列表与 ⟨*family*⟩ 定义的键。在包中调用此函数时将检查全局（类）选项和局部（宏包）选项。该命令将处理当前宏包或类给定的*所有*选项：不需要额外应用 \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。此命令可用于在使用 \ProcessKeyOptions 之前或之后在宏包中设置选项。

4.5 选项传递

这两个命令也在选项的 ⟨*code*⟩ 参数中非常有用。

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

命令 \PassOptionsToPackage 将 ⟨*options-list*⟩ 中的选项名称传递给宏包 ⟨*package-name*⟩。这意味着它将 ⟨*option-list*⟩ 添加到任何未来对宏包 ⟨*package-name*⟩ 的 \RequirePackage 或 \usepackage 命令所使用的选项列表中。

示例：

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

等同于：

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

类似地，\PassOptionsToClass 可以在类文件中用于将选项传递给另一个要由 \LoadClass 加载的类。

这两个命令的效果和用法应与上文（4.2）中的以下两个命令进行对比：

```
\LoadClassWithOptions
\RequirePackageWithOptions
```

命令 \RequirePackageWithOptions 类似于 \RequirePackage，但它始终使用与当前类或宏包使用的选项列表完全相同的选项列表加载所需的包，而不是使用由 \PassOptionsToPackage 明确提供或传递的选项。

\LoadClassWithOptions 的主要目的是允许一个类简单地建立在另一个类之上，例如：

```
\LoadClassWithOptions{article}
```

这应该与略有不同的构造进行比较：

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

如上所用，效果几乎相同，但第一个命令要输入的内容较少；同时，\LoadClassWithOptions 方法稍微更快一些。

然而，如果类声明了自己的选项，则这两种构造是不同的。例如比较：

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

和：

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

在第一个示例中,只有当当前类以此选项调用时,article 类将使用选项 `landscape` 加载。相比之下，第二个示例中，它永远不会使用选项 `landscape`，因为在这种情况下，article 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`，因为该选项已明确声明。

4.6 安全文件命令

这些命令处理文件输入；它们确保以用户友好的方式处理请求的文件不存在的情况。

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

如果文件存在，则执行 ⟨*true*⟩ 中指定的代码。

如果文件不存在，则执行 ⟨*false*⟩ 中指定的代码。

此命令不会输入文件。

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

如果文件存在，则输入文件 ⟨*file-name*⟩，并在输入之前立即执行 ⟨*true*⟩ 中指定的代码。

如果文件不存在，则执行 ⟨*false*⟩ 中指定的代码。

它是使用 `\IfFileExists` 实现的。

4.7 报告错误等

这些命令应由第三方类和宏包用于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*⟩`}`

这些命令产生一个错误消息。显示 ⟨*error-text*⟩，并显示 `?` 错误提示符。如果用户键入 `h`，则会显示 ⟨*help-text*⟩。

在 ⟨*error-text*⟩ 和 ⟨*help-text*⟩ 中：`\protect` 可用于阻止命令展开；`\MessageBreak` 导致换行；`\space` 打印空格。

请注意，⟨*error-text*⟩ 将添加一个句号，因此不要在参数中放入句号。

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.
```

如果用户键入 h，则将显示：

```
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*⟩}

四个 Warning 命令类似于错误命令，但仅在屏幕上产生警告，没有错误提示。

前两个 Warning 版本还显示发生警告的行号，而后两个 WarningNoLine 版本不显示行号。

两个 Info 命令类似，但仅在传输文件中记录信息，包括行号。这两个命令没有 NoLine 版本。

在 ⟨*warning-text*⟩ 和 ⟨*info-text*⟩ 中：\protect 可用于阻止命令展开；\MessageBreak 导致换行；\space 打印空格。同时，这些内容不应该以句号结尾，因为会自动添加。

5 杂项命令等

5.1 版面参数

\paperheight
\paperwidth

这两个参数通常由文档类设置为所使用的纸张尺寸。这应该是实际的纸张尺寸，不同于 \textwidth 和 \textheight，它们是页面边距内主文本区域的尺寸。

5.2 大小写转换

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

如 usrguide 所述，对文本进行大小写转换应使用命令 \MakeUppercase、\MakeLowercase 和 \MakeTitlecase。如果需要更改编程材料的大小写，团队强烈建议使用 str 模块中 L3 编程层的命令。如果不希望这样做，在这种情况下应仅使用 TEX 的 \uppercase 和 \lowercase 原语。

5.3 更好的用户定义的数学显示环境

\ignorespacesafterend

假设你想定义一个以方程编号的文本显示环境。一个直接的方法是：

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

然而，如果你尝试过，你可能会注意到在段落中使用时并不完美，因为在环境之后的第一行开头会出现单词间的空格。

你可以使用 \ignorespacesafterend 避免这个问题；它应该如下所示插入：

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

此命令可能还有其他用途。

5.4 规范化间距

\normalsfcodes

应该使用此命令来恢复影响单词间、句子间等间距的参数的正常设置。

这个特性的一个重要用途是纠正 Donald Arseneau 报告的问题，在局部设置空格代码生效时，页面标题中的标点可能出现问题。这些空格代码会被改变，比如由命令 \frenchspacing 和 verbatim 环境引起。

通常在 \begin{document} 中会自动给出正确的定义，所以不需要显式设置；但是，如果在类文件中显式地使其非空，则默认的自动设置将被覆盖。

5.5 查询本地化信息

自定义一系列输出需要本地化信息。LaTeX 核心本身不管理本地化，而是由 babel 和 polyglossia 来提供良好的支持。为了允许核心和其他宏包访问由 babel 或 polyglossia 提供的当前本地化信息，核心定义了命令 \BCPdata。初始核心定义会扩展到 en-US 的标记部分，因为核心不追踪本地化，但是会使用基本的美国英语设置。但是，如果加载了 babel 或 polyglossia，它会重新定义为相应包中的 BCP-47 信息。支持的参数是 BCP-47 标记的拆分：

- tag 完整的 BCP-47 标记（例如 en-US）
- language（例如 de）
- region（例如 AT）

- `script`（例如 `Latn`）
- `variant`（例如 `1901`）
- `extension.t`（转换，例如 `en-t-ja`）
- `extension.u`（附加区域信息，例如 `ar-u-nu-latn`）
- `extension.x`（私有使用区域，例如 `la-x-classic`）

如果这些参数以 `main.` 为前缀，则会提供文档的主要语言信息，例如 `main.language` 将扩展为主要语言，即使当前活动的语言是其他语言也是如此。

除了标记拆分，还支持以下语义参数

- `casing` 用于更改大小写的标记，例如 `el-x-iota` 可以被选择，而不是 `el`，以在大写时选择大写 adscript iota 的希腊字母

例如，大小写转换命令 `\MakeUppercase` 的（概念上的）定义如下：

```
\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，即地区和输入文本。

5.6 属性的扩展和可扩展引用

属性是 LaTeX 在处理文档时可以追踪的内容，比如页面编号、标题编号、其他计数器数值、标题名称、页面位置等等。这类属性的当前值可以被标记并写入 aux 文件。然后在下一次编译时，可以像标准的 `\label`/`\ref` 命令一样引用这些值（它们记录/引用一组固定的属性：标签、页面、标题和目标）。

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

这个命令将 ⟨*list of properties*⟩ 的值写入 aux 文件，并标记为 ⟨*label*⟩。记录的值要么是在调用 `\RecordProperties` 时的当前值，要么是在下次输出时的当前值——这取决于每个属性的声明方式。⟨*label*⟩ 和 ⟨*list of properties*⟩ 中的参数可以包含被展开的命令。虽然 ⟨*label*⟩ 可以展开为任意字符串（只要它可以安全地写入 aux 文件），但需要注意的是 `\label` 和 `\RecordProperties` 的标签名共享一个命名空间。这意味着使用以下代码会收到 `Label `A' multiply defined` 的警告：

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

\RefProperty{⟨*label*⟩}{⟨*property*⟩}

这个命令允许引用在上一次运行时记录并被 ⟨*label*⟩ 标记的 ⟨*property*⟩ 的值。与标准的 \ref 命令不同，这个命令是可展开的，并且该值可以例如——如果是数字——在赋值中使用。[2]

```
\section{A section}
\RecordProperties{mylabel}{pagenum,counter}
\RefProperty{mylabel}{counter} % 输出章节
\setcounter{mycounter}{\RefProperty{mylabel}{pagenum}}
```

由于 \RefProperty 是可展开的，如果找不到标签，它将不会发出重新运行的警告。如果需要，可以通过以下命令强制发出这样的警告：

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

LaTeX 预定义了一组属性，这个集合也包括标准 \label 命令存储的属性。在下面的列表中，“默认”表示当值尚未知道时返回的值（即，如果它在上一次运行中没有被记录并在输出时），“在输出时”表示该属性在使用 \RecordProperties 时不会立即记录，而是在下一个输出时记录。

abspage（默认值：0，在输出时） 当前页面的绝对值：从 1 开始并在每次输出时单调递增。

page（默认值：0，在输出时） 由 \thepage 给出的当前页面：这可能是一个数字值，也可能不是，这取决于当前的样式。与 abspage 相对比。你也可以用标准的 \label/\pageref 得到这个值。

pagenum（默认值：0，在输出时） 当前页面的阿拉伯数字。这适用于整数操作和比较。

label（默认值：??） \@currentlabel 的内容。这就是你使用标准的 \label/\ref 得到的值。

[2] 为了使其工作，属性的默认值也需要是一个数字，因为在第一次 LaTeX 运行时无法得知记录的值。

`title`（**默认值：** `\textbf{??}`） `\@currentlabelname` 的内容。这个命令由 nameref 包和一些类（例如 memoir）填充，通常给出文档中由某些节命令定义的标题。

`target`（**默认值：** ⟨*empty*⟩） `\@currentHref` 的内容。这个命令通常由 hyperref 填充，并保存它所创建的最后一个目标的名称。

`pagetarget`（**默认值：** ⟨*empty*⟩，**在输出时**） `\@currentHpage` 的内容。这个命令由 hyperref（版本 v7.01c 或更新）填充，并保存它所创建的最后一个页面锚点的名称。

`counter`（**默认值：** ⟨*empty*⟩） `\@currentcounter` 的内容。这个命令在 `\refstepcounter` 后包含计数器的名称。

`xpos`、`ypos`（**默认值：** `0`，**在输出时**） 这些属性记录了先前使用 `\pdfsavepos` /`\savepos` 存储的一个点的 x 和 y 坐标。例如（如果使用了 bidi，可能需要在标签之前和之后保存位置）：

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

类和宏包的作者可以定义更多属性来存储他们感兴趣的其他值。

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

这些命令声明或更改一个属性 ⟨*name*⟩[3]。如果在一个包中声明一个新的属性，建议将其命名为 ⟨*package-name*⟩`/`⟨*property-name*⟩。⟨*setpoint*⟩ 是 `now` 或 `shipout`，决定值是直接写入还是在下次输出时写入。如果属性被引用但尚未知道，则使用 ⟨*default*⟩，例如，在第一次运行时。⟨*code*⟩ 是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}}
```

与属性相关的命令提供了一组传统 LaTeX 2$_\varepsilon$ 包的驼峰命令（如果需要，可以在文档导言部分使用）以及现代包的 `expl3` 命令，它们使用了 LaTeX 的 L3 编程层。`expl3` 命令和更多细节可以在 `ltproperties-doc.pdf` 中找到。

[3]只更改你声明过的属性。绝对不要更改 LaTeX 的标准属性和其他包的属性声明！

5.7 准备链接目标

文档中的活动链接需要跳转到的目标。这些目标通常是自动创建的（如果加载了 hyperref 包），通过 `\refstepcounter` 命令实现。但也存在类或包的作者需要手动添加目标的情况，例如，在无编号的节命令或环境中。为此，LaTeX 提供了以下命令。在 *无* hyperref 的情况下，它们不会执行任何操作或者仅插入一个 whatsits（确保在加载 hyperref 时不会更改间距）；在 *有* hyperref 的情况下，它们会添加必要的目标。有关以下命令的行为和参数的详细信息，可以在 hyperref 包的 `hyperref-linktarget.pdf` 中找到。

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

这个命令准备创建目标。

`\LinkTargetOn`
`\LinkTargetOff`

这些命令允许在局部启用和禁用目标的创建。这对于抑制通过 `\refstepcounter` 自动创建的目标非常有用。

`\NextLinkTarget{`⟨*target name*⟩`}`

这会改变下一个将被创建的目标的名称。

6 被新材料所取代的命令

少量命令是在 LaTeX 2_ε 于 1990 年代中期引入的，被广泛使用，但已被更现代的方法所取代。这些命令在这里被介绍，因为它们很可能会在现有的类和包中被频繁遇到。

6.1 定义命令

这些命令的 * 形式应该用于定义在 TeX 术语中不是长命令的命令。这对于带有不打算包含整段文本的参数的命令进行错误捕获非常有用。

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

这个命令与 \newcommand 接受相同的参数，但它声明了一个强韧命令，即使 ⟨*definition*⟩ 中的某些代码是脆弱的。你可以使用此命令来定义新的强韧命令，或重新定义现有命令并使其强韧。如果重新定义了一个命令，会在转录文件中放入日志。

例如，如果 \seq 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
}
```

那么命令 \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 前加上 \relax；这是因为该 \relax 提供的保护，防止在错误的时候扩展，将在内部提供。

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

这与 \newcommand 接受相同的参数，但它不会定义 ⟨*cmd*⟩，而是检查当前的 ⟨*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。

此命令对于在你的包开始更改命令定义之前检查系统状态非常有用。特别是它允许你检查是否有其他包重新定义了相同的命令。

6.2 选项声明

以下命令处理文档类和包使用经典的“简单文本”方法声明和处理选项。每个选项名称必须是一个“LaTeX 名称”。

有一些命令特别设计用于这些命令的 ⟨*code*⟩ 参数中（见下文）。

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

这将 ⟨*option-name*⟩ 设为放置其中的类或包的“声明选项”。

⟨*code*⟩ 参数包含了如果为该类或包指定了该选项时要执行的代码；它可以包含任何有效的 LaTeX 2_ε 结构。

示例：

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

`\DeclareOption* {`⟨*code*⟩`}`

这声明了一个对于每个为类或包指定但未显式声明的选项要执行的 ⟨*code*⟩；这段代码称为“默认选项代码”，它可以包含任何有效的 LaTeX 2_ε 结构。

如果一个类文件没有包含 `\DeclareOption*`，那么默认情况下，为该类指定但未显式声明的所有选项将悄悄地传递给所有的包（就像对于该类指定并声明的选项一样）。

如果一个包文件没有包含 `\DeclareOption*`，那么默认情况下，为该包指定但未显式声明的每个选项都将产生错误。

6.3 选项代码中的命令

这两个命令只能在 `\DeclareOption` 或 `\DeclareOption*` 的 ⟨*code*⟩ 参数中使用。其他通常在这些参数中使用的命令可以在接下来的几个子节中找到。

`\CurrentOption`

这个命令展开为当前选项的名称。

`\OptionNotUsed`

这会将当前选项添加到“未使用选项”的列表中。

6.4 选项处理

`\ProcessOptions`

这个命令会执行每个选择的选项的 ⟨*code*⟩。

我们首先描述在包文件中 `\ProcessOptions` 的工作原理，然后介绍在类文件中的区别。

要详细了解在包文件中 `\ProcessOptions` 的工作原理，您需要了解“局部选项”和“全局选项”的区别。

- **局部选项**是在任何以下情况的 ⟨*options*⟩ 参数中为特定包明确指定的选项：

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

- **全局选项**是作者在 `\documentclass[<options>]` 的 ⟨*options*⟩ 参数中指定的其他选项。

例如，假设文档开头是：

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

而包 `gerhardt` 使用以下方式调用包 `fred`：

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

那么：

- `fred` 的局部选项是 `german`、`dvips`、`a4paper` 和 `errorshow`；
- `fred` 的全局选项只有 `twocolumn`。

当调用 `\ProcessOptions` 时，会发生以下情况。

- 首先，对于 `fred.sty` 中到目前为止由 `\DeclareOption` 声明的每个选项，它会查看该选项是否是 `fred` 的全局或局部选项：如果是，就会执行相应的代码。

  这是按照这些选项在 `fred.sty` 中声明的顺序进行的。

- 然后，对于每个剩余的局部选项，如果它已在某处定义（而不是由 \DeclareOption 定义），则会执行命令 \ds@<option>；否则，会执行“默认选项代码”。如果没有声明默认选项代码，则会产生错误消息。

  这是按照这些选项被指定的顺序进行的。

在整个过程中，系统确保对于一个选项，其声明的代码最多执行一次。

返回到示例，如果 fred.sty 包含以下内容：

```
\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'.
```

请注意以下几点：

- dvips 选项的代码在 german 选项之前执行，因为它们在 fred.sty 中声明的顺序是如此；
- 只有在处理声明的选项时，才会执行 german 选项的代码一次；
- a4paper 和 errorshow 选项产生了 \DeclareOption* 中声明的警告（按照它们被指定的顺序），而 twocolumn 选项没有：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：所有选项都是局部的；而 \DeclareOption* 的默认值是 \OptionNotUsed 而不是错误。

请注意，因为 \ProcessOptions 有一个 * 形式，最好像前面的示例一样在非星号形式后面加上 \relax，这可以防止不必要的预读和可能引发误导性错误消息。

`\ProcessOptions*`

这与 `\ProcessOptions` 类似，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声明的顺序。对于包，这意味着首先处理全局选项。

`\ExecuteOptions {`⟨*options-list*⟩`}`

它可用于在 `\ProcessOptions` 之前提供“默认选项列表”。例如，在类文件中，如果您希望设置默认设计为：双面打印；11pt 字体；两栏排版。那么可以这样指定：

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