附录C　LaTeX程序设计

LaTeX 2ε的一个主要改进之处就是为类和宏包作者(简言之,即LaTeX程序开发者)提供大力支持。绝大多数用户并没有体味到新版本相比于LaTeX2.09的改进之处，只是知道多了一些新的字体命令，文档开头之处的声明换成了 `\documentclass`，用 `\usepackage` 命令上载原来的 *选项文件*。虽然在其它地方也有可能发现多了点新东西，但大致的感觉就是新版本并没有带来多少新事物。然而，对LaTeX程序开发者而言，尤其是对曾经编写过选项文件，或者安装过新字体框架的人员来说，他们就会非常欣赏NFSS以及新的类与宏包控制功能。随着时间的推移，基本LaTeX安装所提供的新扩展功能逐渐变得与LaTeX 2ε密不可分，从而想使用新功能的用户就不可避免地把自己的系统更新到新版本上。

在8.5节中描述了新字体选择框架。本附录讲解设计类与宏包文件的特殊命令，并给出了几个实用宏包的设计示例。

§C.1　类与宏包文件

§C.1.1　LaTeX2.09 中的样式文件

类与宏包文件是新出现在LaTeX 2ε中的概念，它取代了LaTeX2.09中的 *主样式* 和 *样式选项* 文件。在原来的系统中，要想选择主样式，就要利用声明

2.09 `\documentstyle[`*选项清单*`]{`*样式*`}`

这样就上载了一个名为 *样式*`.sty` 的文件，其定义了文档所需要的全局格式。几个主要的样式为 `article`, `report` 和 `book`，这也是LaTeX 2ε中最重要的几个类。在样式文件的某个地方，要用命令 `\@options` 如下处理 *选项清单* 中的选项：

1. 如果存在命令 `\ds@`*选项*，那就执行这条命令；否则，就把该选项放到第二个清单中；
2. 当遍历完 *选项清单*,就考虑第二个清单，对每一项，把文件 *选项*`.sty` 输入进文档。

这一过程使得选项可以定义在样式文件内部，也可以存贮在与选项同名的单独样式选项文件(扩展名为 `.sty`)中。这样做的想法就是有些选项的代码与主样式文件无关，从而可以存贮在外部的样式文件中。

这就可能是采取选项文件概念的初始动机；其直接结果就是有大批的爱好者开发了数目可观的添加‘选项’，并把它们存贮在一个文件中，读入到其它的文档中。通过网络服务器，添加的选项就传播开来，得到广泛的应用。由于其中有些只是拙劣地修补了LaTeX的内部命令，不能只是简单地用 `\input`

命令把它包含进来，而要把文件名后缀指定为 .sty，因此可以按照上面的第 2 点以准选项角色包含进来。但它们实际上并不是真正的选项，只是新增加的代码或功能。

在编写主文件样式时还会出现一个新的难题。如果需要适合于某期刊的文章样式，或者某出版社的书籍样式，那么可以利用已有的 article.sty 或 book.sty 样式为基础，进行必要的改动，得到新的主样式，但是不能保证在对原主样式做更新后，改动仍然有效。但是有时候一些重大的更新就与对LaTeX进行的改动是一致的。我们可以只是写一个 “选项” 来包含所做的改动，也可以写一个新的主样式，输入原来的样式。然而，它并不是相应于其选项清单的真正主样式。

§C.1.2 LaTeX 2ε的新概念

当 Leslie Lamport 发布 LaTeX时，他无论如何也不会预见到随后出现了那么多的 LaTeX程序。这一切现在已成为不争的事实，而且这也是该系统的魅力所在。LaTeX 2ε则不但包容这些 “外来” 成员，而且实际上它进一步支持和鼓励这种趋势，其中一个明证就是在 *The LaTeX Companion* (Goosens et al., 1994) 一书所介绍的大量宏包。

这就是事物发展的趋势。扩展的功能由那些需要该功能的人设计的，因为他们意识到 LaTeX缺少了某些对他们而言是很重要的功能。另一方面，要是把所有这些扩展的功能都加入到基本的 LaTeX安装中，那就会使得 90% 以上的用户虽然上载了它们，但从来不会用它们。现在解决这个问题的方法就是 LaTeX提供了一个基本的核心 (或称内核)，再首先用标准的类文件扩展其功能，然后利用那千变万化的宏包和类增加功能。

而 LaTeX Team 的任务就是建立程序设计的方针，从而确保宏包不会与内核或者其它宏包发生不必要的冲突，而且提供一种基本的稳定性，使得那些实用的宏包将来在更新后的内核和标准类下也工作正常。在 LaTeX2.09 中就缺少了这种安全机制，在那个版本中，程序设计者们被迫自己寻找门路，这实际上并不是真的程序开发。 LaTeX 2ε在类和宏包控制方面的新功能，随同一组程序开发工具，应该在宏包内部相互作用以及对内核更新的适应等方面达到比原来更强的可靠性和持久性。

§C.1.3 命令的层次

命令有许多层次，它们的安全程度也相应不同。

用户命令 (最高级命令) 在本书及其它手册中进行了描述，其名称由小写字母组成，例如 \texttt，它是永久被支持的 LaTeX外部定义；

类与宏包命令 其名称要稍长一些，而且大小写混杂 (如 \NeedsTeXFormat)，主要是为程序设计人员提供的，而且也是有保障的；绝大多数是只能用

在导言中的命令，但在类和宏包文件中并没有这个使用限制；

LaTeX内部命令 名称中包含 @ 字符，只能用在类与宏包文件中；虽然其中有些命令对得到特殊效果是密不可分的，但也无法保证永远可用；开发人员要使用该命令，那就有可能将来某一天自己设计的宏包变得不再能用了；

TeX低级命令 名称也是由小写字母组成，而且没有 @；即使 LaTeX继续演化，其功能也应该是稳定的，但这也不是绝对的；只要有可能，就尽量避免使用它们，见下面的解释；

内部专用命令 是用在其它人员开发的类与宏包文件内部的命令；建议所有命令都前缀大写字母 (以表示宏包的名称) 后接 @，这样可以避免与其它宏包发生冲突；例如， showkeys 宏包中有一条命令为 SK@cite 。

一个令 LaTeX开发人员感到迷惑的问题是 LaTeX内部命令在类和宏包文件中的应用范围到底有多大。总是存在着可能，将在某一天在新版本不存在这些新命令了，因为在正式的说明书中从没有给出其说明。而诸如下节讨论的 TeX命令，我们不应杜绝使用它们，但我们必须明白，用它们也同时伴随着一定程度的风险。

LaTeX Team 建议的方针就是只要有可能，就尽量使用 LaTeX高级命令。

- 要用 \newcommand 和 \renewcommand，而不用 \def；如果要使用某一个 TeX的定义命令 (因为调用某模板，或者因为必须用 \gdef 或 \xdef)，那么先调用一个空的 \newcommand 命令，以检测名称是否有冲突。如果无法确定命令名称是否存在，而且该命令不是很重要的，那么就调用一条空的 \providecommand，然后再调用 \renewcommand 。现在高级命令中可以定义有一个缺省值的命令，这使得原来经常要用低级命令的理由中缺少了重要的一条。
- 利用 \newenvironment 和 \renewenvironment 命令，而不用 \自己的环境 和 \end自己的环境 命令对。
- 要用 \setlength 命令给长度和橡皮长度赋值，而不要用直接等号方式。
- 避免使用 TeX盒子命令 \setbox, \hbox 以及 \vbox；而要用诸如 \sbox, \mbox, \parbox 一类的命令。利用 LaTeX 2_ε提供的可省参考值，原来对等价 TeX命令的需求现在大大降低，而且 LaTeX版本相比起来要透明得多。另外，当用了 color 宏包时， LaTeX的盒子仍然工作正常，而其它的命令结果就无法预料了。
- 如果想给出错误和警告消息，就用 \PackageError 和 \PackageWarning, 不要用 \@latexerr 或 \@warning；前面两条命令也同时告诉用户消息的来源，来不是只把它们标为 LaTeX消息。
- 我们不会建议你只使用 ifthen 宏包 (7.3.5 节) 中的 \ifthenelse 命令，以代替 TeX的条件命令。但是似乎用这个宏包可以简化对条件的应用，

而且符合 LaTeX的语法。本书所有的例子都用的是这个宏包，这就不需要再解释相应的 TeX命令。

遵从这些以及与之类似的规则，有益于在将来即使 LaTeX内核进行了更新，宏包也能继续保持有效的功能。

§C.1.4 TeX命令

为什么要避免用那些基本的 TeX命令呢？要想定义一条新命令，如果用 \def 就至少不会比用 \newcommand 差，而且有时候还必须用前者。那么在将来出现的 LaTeX3 中这条命令有可能被去掉吗？基本命令 (原语) 是所有 LaTeX风味得以建立的构造模块，因此它们一定会维持不变的。

而这并不是这样做的关键所在。基本 TeX命令构成了所有格式的基石，从而所有用它们直接定义的命令，其功能就永远与开发人员所期望的一样。然而，等价的 LaTeX工具所能做的事情却会随着时间的发展而增多。例如，\newcommand 命令可以检测新定义的命令是否与已有命令发生冲突。而且，以后完全有可能会加进一种调试机制，它可以跟踪所有的重定义；而用 \def 定义的命令则是排它的。而且现在的 LaTeX 2_ε中也有一种机制，它可以跟踪所有中级和高级命令的文件输入。

另外一个低层次程序开发人员容易误入歧途的例子的就是 LaTeX 2_ε中牢固命令处理的情形。有很多命令本质上是脆弱的，也就是说当把它们用做其它命令的参数值时，会过早被解释，可如果给它们前缀 \protect，一般可以使其变得牢固。在 LaTeX2.09 中，有几条脆弱命令在定义时采用的是一种牢固方式，即定义中包含了 \protect，例如 LaTeX的标志命令：

```
\def\LaTeX{\protect\p@latex}
\def\p@latex{...}
```

真正的定义是在内部的 \p@latex 中，并不是在外部的 \LaTeX 中。由于如此的标志定义中有很多缺陷，因此有几个宏包引入了一个改进的版本。它们只是重定义 \LaTeX，从而使得这条命令变成脆弱的了；因此聪明的方法是重定义 \p@latex，这就利用了隐藏在 LaTeX 2_ε后台的结果，但是命令却以一种完全不同 (而且更好) 的方式变得牢固了。(顺便说一下， LaTeX标志的内部定义已有了很大的改进。)

虽然我们希望只使用正式发布的 LaTeX命令，但也有很多情形，我们必须用 LaTeX内部命令或者 TeX基本命令。在目前阶段，为了得到一个工作稳定的宏包，我们就必须考虑在将来可能会不兼容的风险。而且如果有等价的高级命令可用，我们就不应冒这个风险。

§C.2 LaTeX 2ε程序设计语言

本节描述的所有命令都是在 LaTeX 2ε中新增加的。虽然它们对类和宏包文件而言，并不是很本质的，但它们确实扩展了类与宏包的用途，并保证使用时的正确性。

§C.2.1 文件识别

有三条命令用来测试类或宏包插入时所处的外部环境是否正确。其中第一条为

2ε `\NeedsTeXFormat{`格式`}[`版本`]`

在类或宏包中的第一条语句就应该是所需要的 TeX格式声明。虽然已有很多其它名称的格式，但只有名为 LaTeX2e 的格式才认识这条声明。而所有其它格式都会给出错误消息：

```
! Undefined control sequence.
1.1 \NeedsTeXFormat
                      {LaTeX2e}
```

此时，这条消息实际上就给出了提示信息。

在 LaTeX 2ε中这条命令可能更有用的地方是它的可省参数值 版本，该参数的形式必须为表示发行日期的 yyyy/mm/dd。如果一个宏包利用的功能是在某一个版本中才引进的，那就必须给出这个日期，因此如果用的是 LaTeX 2ε一个更早的版本，就会显示出一条警告。例如，在 LaTeX 2ε起初的测试版本中并没有提供命令 `\DeclareRobustCommand`，只是在 1994 年 6 月 1 日正式发行时才有了这条命令。因此使用了这条命令的宏包就应该以下面这条语句开头：

```
\NeedsTeXFormat{LaTeX2e}[1994/06/01]
```

这里日期的形式是很重要的，必须有零和斜杠。

这条声明并不是只限于用在类和宏包文件中，也可以在文档开头调用它，以保证用正确的 LaTeX处理该文档。然而调用的位置就必须是在导言中。

下面两条命令用来标明类或宏包文件自身：

2ε `\ProvidesClass{`类`}[`版本`]`

2ε `\ProvidesPackage{`宏包`}[`版本`]` 在这两条命令中，版本 都是由三部分组成的：日期，版本号以及附加信息。日期与上面的格式相同，而版本号可以是任何没有空格的标志，附加信息可以是有或没有空格的文本。例如，

```
\ProvidesPackage{shortpag}[1995/03/24 v1.4 (F. Barnes)]
```

LaTeX只会检查其中的日期部分，也就是说如果使用了 `\usepackage` 命令并给出了日期，那么 LaTeX就会对两处的日期进行比较。如果调用了 `\listfiles`，

那么会显示出来版本号和附加信息部分。然而，对于 doc 宏包 (D.3.2 节) 中的 \GetFileInfo 而言，上述格式就是必须的了。

\documentclass 和 \usepackage(以及 \LoadClass 和 \RequirePackage) 命令都可以包含一个可省参数，以指定类 / 宏包可接受的最早发行日期。例如，当声明为

```
\documentclass[12pt]{article}[1995/01/01]
```

这时如果使用的 article 类文件中包含

```
\ProvidesClass{article}[1994/07/13 v1.2u
   Standard LaTeX document class]
```

那么就会显示出一条警告消息。对于 \usepackage 和 \ProvidesPackage 命令，也是同样的机理。

版本检测机制使得文档可以索取处理自己的合适版本的类和宏包文件。当然这里要假设所有以后版本都与以前的版本完全兼容。

对于那些用 \input 命令上载的普通文件，还有一条识别命令：

2ε `\ProvidesFile{`*文件名*`}[`*版本*`]`

此时不会检测名称或版本，但是利用 \listfiles 可以列出这两部分信息。

§C.2.2 上载其它类和宏包

在主文档文件中，类的读入是利用初始化 \documentclass 命令来实现的，而宏包用的则是 \usepackage 命令。在类和宏包文件内部，就必须使用下述命令：

2ε `\LoadClass[`*选项*`]{`*类*`}[`*版本*`]`

2ε `\RequirePackage[`*选项*`]{`*宏包*`}[`*版本*`]`

其中第一条命令可使得一个类文件上载另一个类文件，并且需要的话，可以给出选项；而第二条命令使得类和宏包文件上载其它的宏包。在任何类文件中只能有一条 \LoadClass 命令；不能在宏包文件中使用。这两条命令都可以用在文档文件中。其中的 *宏包* 参数值可以是几个宏包名称组成的清单，中间用逗号分开。

可省 *版本* 参数与相应的 \Provides.. 命令之间的关系在前一节中做了介绍；而我们下面将介绍 *选项* 参数的处理方式。

§C.2.3 选项的处理

在类和宏包中都可以有选项，其定义方式为

2ε `\DeclareOption{`*选项*`}{`*代码*`}`

其中 *选项* 就是选项的名称，而 *代码* 就是选项要执行的指令集。在 LaTeX内部，实际上创建了一条叫 \ds@*选项* 的命令。通常这些代码并不做任何事，只

是设置一些标志，或输入一个选项文件。(\RequirePackage 不可以用在选项代码中！) 在 article.cls 文件中的两个示例为

```
\DeclareOption{fleqn}{\input{fleqn.clo}}
\DeclareOption{openbib}{\setboolean{\@openbib}{true}}
```

可以用 \DeclareOption* 定义一个缺省选项，这条命令并不需要选项名称，只是指定适用于所有被调用的未定义选项的执行代码。

有两条特殊命令，可能用在缺省选项定义的 *代码* 中：

\CurrentOption 由正在被处理的选项名称组成；

\OptionNotUsed 把 \CurrentOption 声明为未处理的。

例如，若想有一个类文件，模拟 LaTeX2.09 在所有未定义选项上载同名的 .sty 文件时的行为，可以如下定义：

```
\DeclareOption*{\InputIfFileExists{\CurrentOption.sty}%
                {}{\OptionNotUsed}}
```

这样就会首先检测是否存在指定名称的 .sty 文件，如果不存在，就把选项声明为没有使用的。要求的选项没有使用（处理）的话，就会列在一条警告消息中。

接下来就用下面的命令处理选项：

2ε \ExecuteOptions{ *选项清单* }

2ε \ProcessOptions

2ε \ProcessOptions*

其中 \ExecuteOptions 会为 *选项清单* 中的每个选项调用 \ds@*选项* 命令。在默认方式下通常就是建立起特定的选项配置。 \ProcessOptions 按照所有选项定义的顺序执行调用的选项，然后删除它们。这也就是说这条命令只能执行一次。有星号的命令功能类似，只是它是按调用的顺序执行。为了与 LaTeX2.09 样式兼容，现在仍然保留了命令 \@options ，它只是 \ProcessOptions* 命令的另一个名称而已。

也可以用下面的命令为类或宏包定义选项：

2ε \PassOptionsToClass{ *选项* }{ *类名* }

2ε \PassOptionsToPackage{ *选项* }{ *宏包名称* }

其中 *选项* 是一串指定类或宏包文件可以识别的合法选项。这两条命令可以用在其它选项的定义中。最常见的用法就是把缺省选项传递给另一个类，如下例所示：

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

这里所定名的类或宏包必须稍后用 \LoadClass 或 \RequirePackage 命令进行上载。

如果类和宏包文件的缺省选项并没有用 \DeclareOption* 进行改变，那么处理未定义的被调用选项的标准程序如下：

- 所有在 \documentclass 语句中的选项标记为 *全局的*; 认为其要应用于后面所有宏包，但用 \LoadClass 上载的类除外；如果在主类中没有定义该选项，不会给出错误或警告消息；
- 所有其它语句（包括 \LoadClass 和 \PassOptionsTo..）给出的选项都是 *局部的*；如果在相应的类或宏包中没有定义该选项，会给出一条错误消息；
- 当所有宏包都读进来后，如果某一个全局选项还从来没有用过（从没有被定义），就会给出一条警告消息；
- 无论是全局选项，还是局部选项，都按照定义的顺序执行（除非调用了 \ProcessOptions* ）。

§C.2.4 延期处理

有时候，为了得到某种特殊效果，或者避免与其它宏包发生冲突，希望有些命令是在类或宏包结束处执行，或者在文档的开头或结尾处执行。这可以用下面的命令实现这种效果：

2ε \AtEndOfClass{ 命令集 }

2ε \AtEndOfPackage{ 命令集 }

2ε \AtBeginDocument{ 命令集 }

2ε \AtEndDocument{ 命令集 }

前两条命令把 *命令集* 保留到类或宏包结尾时才执行。配置文件利用它们在开始部分读入，但它由在结尾处所应进行的修改组成，这样在结束时就不会由于缺省方式而被重写。后两条声明把 *命令集* 分别保留到 \begin{document} 和 \end{document} 中执行。上述所有命令都可以不只调用一次，这样 *命令集* 执行时就会按照调用的顺序进行。

保存在 \AtBeginDocument 中的 *命令集* 是会准确地插入在导言的处理流中，但它是在 \begin{document} 命令已几乎做完自己该做的事情之后。因此可以认为 *命令集* 是正文的一部分，但是那些只能用在导言中的命令也可以用在其中。

§C.2.5 牢固的命令

在 7.3 节中我们详细讲述了如何使用 \newcommand 定义新的命令，使用 \renewcommand 重定义命令，以及使用 \providecommand 临时创建命令。实际上还有另外两条语法相同的定义命令的语句：

2ε \DeclareRobustCommand{ *命令名* }[*参数个数*][*可省参数*]{ *定义* }

用来定义或重定义一个名为 *命令名* 的命令，而且它是 *牢固的*，也就是说它可以用做其它命令的参数值，前面不需前缀 \protect 。如果命令已经存在了，就会向抄本文件中写入一条消息，并覆盖原来的定义。

另外一条命令可以用来检测 \命令名 的当前定义。

2ε \CheckCommand{ 命令名 }[参数个数][可省参数]{ 定义 }

如果命令的定义与 定义 不同，或者参数个数不同，等等，就会给出一条警告消息。这可以用来确认系统的状态是我们所希望的样子，并不存在已上载的宏包修改了某些重要的定义。

\DeclareRobustCommand 和 \CheckCommand 命令都可以用在文档的任何地方。

§C.2.6 有短参数值的命令

通常在用户定义命令的参数值中可以包含用 \par 或空行表示的新段落。按 TeX的行话来说，这些命令都是 '长' 的。这也不是用 \def 定义命令的标准行为，因为用它定义的命令必须是短的，这样可以检测是否遗漏了右大括号。

到了 1994 年 6 月 1 日，新发行的 LaTeX 2ε提供了所有定义命令的星号形式：

```
\newcommand*              \renewcommand*
\newenvironment*          \renewenvironment*
\providecommand*
\DeclareRobustCommand*    \CheckCommand*
```

上述命令创建的命令都具有 '短' 参数，从而行为与 \def 一样。

我们建议总是用星号形式命令来定义新的命令，除非有足够的理由取某些参数为 '长' 的，即参数中包含段落。长参数应当是例外，而不是规则。

§C.2.7 给出错误和警告消息

在设计类和宏包时，也应当使它们具有自己的错误和警告消息。这对帮助我们辨别到底哪个文件发出这条消息是非常有用的。

错误消息是用下列命令生成的：

2ε \ClassError{ 类名 }{ 错误消息文本 }{ 帮助 }

2ε \PackageError{ 宏包名 }{ 错误消息文本 }{ 帮助 }

其中 错误消息文本 就是显示在监视器或抄本文件中的消息，而 帮助 就是当用户反映为 H 时显示的进一步文本。如果文本中包含命令名，而且要按原样显示，那就必须前缀 \protect；空格是用 \space 生成的，新行用 \MessageBreak 开始。例如，

```
\PackageError{ghost}{%
  The \protect\textwidth\space is too large\MessageBreak
  for the paper you have selected}
  {Use a smaller width.}
```

就会生成如下错误消息：

```
! Package ghost Error: The \textwidth is too large
(ghost)                for the paper you have selected.

See the ghost package documentation for explanation.
Type H <return> for immediate help.
```

当LaTeX停下来等用户给出一个反应时，若按9.1节中描述那样输入H⟨回车⟩，就会得到

```
Use a smaller width.
```

在类和宏包中也可以按类似的方法给出警告消息。差别就在于后者没有*帮助*文本，而且处理过程也不会停下来等待反应。可以包含警告消息出现时在输入文件中所处的行号。

2ε \ClassWarning{*类名*}{*警告消息文本*}

2ε \ClassWarningNoLine{*类名*}{*警告消息文本*}

2ε \PackageWarning{*宏包名*}{*警告消息文本*}

2ε \PackageWarningNoLine{*宏包名*}{*警告消息文本*}

例如，当定义了

```
\PackageWarning{ghost}
   {This text is haunted}
```

就会得到了消息

```
Package ghost Warning: This text is haunted on input line 20.
```

而且处理不会停下来。警告消息可以用 \MessageBreak 分成几行，这一点与错误消息中类似。

此类型的最后两条命令是

2ε \ClassInfo{*类名*}{*信息文本*}

2ε \PackageInfo{*宏包名*}{*信息文本*}

它只把文本写到抄本文件中，不会显示在监视器上。从其它角度来看，就如同没有 NoLine 的警告消息。

§C.2.8 输入文件

不是类和宏包的文件也可以输入到文档中，当这样做的时候，通常必须事先保证文件是存在的。或者，根据文件是否存在来决定要采取的进一步行动。这一目标是用如下命令实现的。

2ε \IfFileExists{*文件名*}{*真*}{*假*}

2ε \InputIfFileExists{*文件名*}{*真*}{*假*}

这两条命令都会在LaTeX文件搜索路径中看看有没有指定的*文件名*，如果找到了文件，就执行*真*，否则就执行*假*。而且，在 \InputIfFileExists 命令

中，执行了 真 后还会读入该文件。

这些命令并不限于只用在导言中，也不限于只用在类或宏包文件中。实际上，通常的 \input 命令就是用它们定义的。

许多特殊的类利用这些命令读入局部的配置文件。例如，在类 ltxdoc 中包含语句

```
\InputIfFileExists{ltxdoc.cfg}
    {\typeout{Local config file ltxdoc.cfg used}}
    {}
```

它就放在 \ProcessOptions 前面。这样可以有一个局部配置文件，其相应于欧洲安装版本，指定

```
\PassOptionsToClass{a4paper}{article}
```

而不用修改用 ltxdoc 类处理的文件。

§C.2.9 检测文件

我们在此描述两条跟踪使用文件的命令，它们并不是程序设计的一部分。其中第一条命令就是我们在 8.1.1 节已提到的命令

2ε \listfiles

这条命令可以放在导言中，甚至 \documentclass 命令的前面。在处理过程结束后，它会生成并显示出所有输入文件的清单，同时包括文件的版本和发行数据。用这种方法，我们就可以得到所有被包含进来的文件记录，当要把一个文件送到另外的地方，用不同的安装版本进行处理时，这一记录信息就可能非常有用。由于非标准文件也有可能被包含进来，那么从上面的清单中可以很容易识别出来。

例如，输入下面这个简单的文档文件：

```
\documentclass{article}
\usepackage{ifthen}
\listfiles
\begin{document}
  \input{mymacros}
   This is \te.
\end{document}
```

就会得到如下的清单

```
*File List*
article.cls 1994/07/13 v1.2u Standard LaTeX document class
 size10.clo 1994/07/13 v1.2u Standard LaTeX file (size option)
 ifthen.sty 1994/05/27 v1.0i Standard LaTeX ifthen package (DPC)
```

```
mymacros.tex
 **********
```

在这种情形里，局部文件 mymacros.tex 中不包含版本信息，因为其中没有用 \ProvidesFile 命令。

如果要把上面这个文档文件关到其它地方进行处理，那么该如何处理像上面 mymacros.tex 那样的局部文件呢？当然可以与主文件一起寄送该局部文件，那么这就需要告诉收件人更多指令，以确定该如何行事。另外一种方法就是为了寄送文件，把局部文件内容直接包含在主文件中。对于宏包文件，这样做可就不是那么容易了，因为内部命令中有 @ 符号，会造成一些麻烦，从而不能正确处理一些选项。现在 LaTeX 2ε提供了如下环境：

2ε `\begin{filecontents}{`文件名`}`

文件内容

`\end{filecontents}`

这个环境可以用在文档开头，即 \documentclass 命令的前面。这个环境首先会检测系统中是否存在一个文件，名为 文件名，如果不存在，它就会把 文件内容 照原样写到那个名称相应的文件中。该文件可以是一个宏包，随后要用 \usepackage 上载它。利用这种方法，少掉的非标准文件就可以与主文档文件一起寄送了。

我们推广上面那个简单例子，在开头部分输入

```
\begin{filecontents}{mymacros}
\newcommand{\te}{the end}
\end{filecontents}
```

那么新生成的 mymacros.tex 内容为

```
%% LaTeX2e file `mymacros'
%% generated by the `filecontents' environment
%% from source `mydoc' on 1994/09/27.
%%
\newcommand{\te}{the end}
```

注意 filecontents 环境加进了一些注释行，说明新文件来自于何处。如果不希望加进这些注释行，那可以用 filecontents* 环境。

§C.2.10 兼容模式

为了使得专门为 LaTeX2.09 编写的老文档也可以用 LaTeX 2ε进行处理，现在存在着一个 兼容 模式，它用 \documentstyle 取代 \documentclass 。这样整篇文档就可以只遵从原来的标准，但不能使用 LaTeX 2ε的所有新功能。

然而，兼容模式仍旧上载新的类文件，而不是原来的样式文件，因为这

些样式文件将来可能不会存在的。该模式首先查找后缀为 .cls 的文件，只有当该文件不存在时，它才会上载 .sty 文件。这就是为了迁就原来的非标准样式，使其功能就像类一样。

类似于 article 这样后缀为 .cls 的标准类文件，甚至用 \documentstyle 也可以上载。但是这时其功能必须与将要废除的 .sty 文件功能一样。实际上它们是有很多差别的，例如如何设置页边和文本尺寸。不但如此，而且我们要求兼容模式的输出必须与用 LaTeX2.09 以及 article.sty 处理时完全一样。为了做到这一点，在兼容模式中把 boolean 开关 (7.3.5 节)@compatibility 设置成 ⟨真⟩，这样任一类或宏包都可以检测该模式。检测的形式为

\ifthenelse{\boolean{@compatibility}}{ 真 }{ 假 }

其中 真 表示只适用于兼容模式的命令，而 假 表示那些可以用在真正 LaTeX 2ε 模式中的普通命令。

对于所有的标准类文件，仍然存在着 .sty 文件，例如 article.sty，只不过内容是空的，它只是给出一条警告消息，并上载类文件。这是为了兼容那些旧宏包，它们有可能显式上载这些文件。

§C.2.11 有用的内部命令

尽管在 C.1.3 节中的方针建议最好避免用 LaTeX内部命令，但是其中有些命令是非常基本的，它们组成了 LaTeX内核和许多标准宏包的构造模块。由于它们终究还是内部命令，因此无法保证在以后的更新中它们还是存在的。然而，如果真的没有了它们，那么由 LaTeX Team 所提供的大量的有趣扩展功能宏包就要进行全面的大检修。我们这里只是为那些勇敢的用户们简单地介绍一下这些命令。在 LaTeX 2ε和 2.09 中都存在这些命令：

\@namedef{ 命令 }{ 定义 }

\@nameuse{ 命令 }

就会定义并执行名为 \命令 的新命令，其中命令名称中并不包含反斜杠。这个名称中可以包含任意字符，即使通常在命令名称中禁止使用的字符也可以。

\@ifundefined{ 命令 }{ 真 }{ 假 }

如果 \命令 不存在，就执行 真 ，否则执行 假 。同样这里 命令 中也不包含反斜杠，而且任何字符都可以出现在命令名称中。这个检测语句通常用来有条件地定义命令，其功能现在已经被 \providecommand 代劳。也可以用它确定主类是否是 article： \@ifundefined{chapter}{..}{..} 可以用来检测 \chapter 命令是否存在。

\@ifnextchar 字符 { 真 }{ 假 }

用来检测下一个字符是否是给定的 字符 ，如果是的话，执行 真 ，否则执行 假 。这条命令通常用来定义有可省参数的命令，这时 字符 就是 [。新扩展的 \newcommand 命令用高级方法得到了这一效果。

`\@ifstar{ 真 }{ 假 }`

用来检测下一个字符是否是星号 `*`，如果是的话，就执行 *真*，否则执行 *假*。可以用它来定义带星号的命令和环境，这是高级命令做不到的。

`\@for \ 对象 := \ 列表 \do { 命令 }`

其中 `\列表` 就是一条命令，被定义成一串用逗号分开的元素，而 `\对象` 就相继取值等于每个元素，并对每个元素执行一次 *命令* 代码。例如，

```
\newcommand{\set}{start,middle,end}
\@for \xx := \set \do {This is the \xx. }
```

就会显示出 'This is the start. This is the middle. This is the end.'

§C.2.12 有用的 TeX命令

LaTeX中许多复杂的功能以及宏包都只能用 Plain TeX命令开发出来。这些命令的描述不但可以在 Knuth 的 *The TeXbook*(1984 年) 一书中找到，而且还有一本相当好的参考书，那就是 Eijkhout 的 *TeX by Topic, a TeXnician's Reference* (1992 年)。

我们不打算要把本书写成 TeX手册；不但如此，而且常用的出现在许多宏包以及后面示例中的 TeX命令数目也很少。只要给出一个简短的描述，就对理解这些命令的作用非常有帮助。真正的 TeX专家和 TeX技术人员可以略过这一节。

`\def\ 命令 #1#2..{ 定义 }`

是 TeX中标准的定义命令。它等价于 `\newcommand`，只是它不会检查是否有名称冲突，而且参数的指定也不同。例如，显示科学记数法的命令 `\Exp` 可以如下定义：

```
\def\Exp#1#2{\ensuremath{#1\times10^{#2}}}
```

或者

```
\newcommand{\Exp}[2]{\ensuremath{#1\times10^{#2}}}
```

对于这两种定义方式，`\Exp{1.1}{4}` 的结果都是 1.1×10^4。然而，`\def` 还可以做得更多。它可以把参数放在一个模板中，例如

```
\def\Exp#1(#2){\ensuremath{#1\times10^{#2}}}
```

这样可以得到更方便的记号 `\Exp1.1(4)`，而这是 `\newcommand` 命令所做不到的。当定义命令时，不知道 (或不关心) 是否已存在同名命令时，或者要用模板时就用 `\def`。有可省参数的命令实际上就是用模板定义的。

`\gdef \edef \xdef`

是 `\def` 的变形；第一条命令给出一个全局定义，所得命令即使在当前环境或者 `{..}` 括号对外面也仍然有效；第二条命令是一个展开的定义，其中任何命令都具有自己本身的意义，而并不是把命令插入在定义中；最后那条是前面两条的组合，即展开的全局性定义。

`\noexpand` `\expandafter`

控制命令在定义和执行时的展开。在 `\edef` 中定义部分的任何命令都要被展开(插入其含义)，除非其前缀 `\noexpand` 。相反的效果可以用 `\expandafter` 得到，这条命令跳过后接命令，展开下一条命令，然后执行被跳过的命令。这就是相当深奥的 TEX技术了，我们最好还是用上面那条 `\Exp` 命令来演示一下结果。

```
\newcommand{\mynums}{1.1(4)} \expandafter\Exp\mynums
```

就等价于 `\Exp1.1(4)` ，而 `\Exp\mynums` 并不等价；在执行 `\Exp` 之前先把 `\mynums` 展开成 `1.1(4)` 。

`\let\` 命令一 `=\` 命令二 或 `\let\` 命令一 `\` 命令二

使得 `\`命令一 取 `\`命令二 当前含义。这通常用来在重定义命令前保存其原来定义，从而可以同时使用其原来定义。

`\relax`

绝对不做任何事，但通常用来插在应该有些什么东西的地方，但我们不想有内容的地方。

`\if` 条件 真 `\else` 假 `\fi`

就是 TEX中条件语句的形式。有许多不同形式的条件，我们这里就不一一介绍了，但常见的应用就是等价于 LATEX 的 boolean 开关命令：

```
\newif\if 标志          = \newboolean{ 标志 }
\ 标志 true             = \setboolean{ 标志 }{true}
\ 标志 false            = \setboolean{ 标志 }{false}
\if 标志 ..\else..\fi = \ifthenelse{\boolean{ 标志 }}{..}{..}
```

对那些经常用这些语句的人而言， TEX形式要紧凑一些，但它并不与一般的LATEX行事方式一致。

`ifcase` 数 文本 0 `\or` 文本 1 `\or...\fi`

会根据 数 的值来决定执行哪个 文本 。

`\endinput`

终止对当前文件的输入。这并不是必需的方式，但所有的文件都用它结束不失为一个好的程序设计习惯。在主文档中不必用它，因为 `\end{document}` 可以得到同样的效果。

§C.3 宏包示例

我们下面给出几个示范性宏包，以说明前面一节给出的程序设计命令的使用方法。这些宏包并不是微不足道的，从某种角度来看，它们都是相当有用的。

§C.3.1 修改文本尺寸

在标准LaTeX类中，要根据 \documentclass 中指定的尺寸选项（如选项 a4paper 或 legalpaper 来设置文本尺寸参数 \textwidth 和 \textheight 。而且同时调整页边距，使得文本水平和竖直居中。\textwidth 的值是有限制的，每行上最多有 60–70 个字符，这主要是出于排版中达到最佳视觉效果的考虑。

而有时候我们想去掉这个限制，充分利用纸张的最大幅度。只有通过多次尝试，我们才有可能找到恰当的参数组合，匹配特定的纸张尺寸。如果有一个宏包可以帮你做到这一点，那就太好了。

下面给出的 fullpage 宏包利用了 \paperwidth 和 \paperheight 的值（这两个值是要根据纸张尺寸选项进行设置的），以生成两边只有一英寸页边的页面。它甚至可以考虑到当前页面样式，从而为必要的页眉和页脚留下空间。如果通过选项，可以得到只有 1.5cm 窄的页边。

在继续下面的设计之前，我们有必要先复习一下在 31 页和 ?? 页插图中的页面格式参数。

这个宏包的开头部分就声明自己需要 LaTeX 2_ε ，然后标明自己的身份。宏包信息由预定义格式的日期，版本号以及作者姓名大写首字母组成。由于需要条件判断，接着就上载了 ifthen 宏包。

```
\NeedsTeXFormat[LaTeX2e}
\ProvidesPackage{fullpage}[1994/02/15 1.0 (PWD)]
\RequirePackage{ifthen}
```

下面就准备好选项。选项有 in 和 cm 分别相应于 1 英寸和 1.5 厘米（ 1 厘米就太窄了）的页边。要用一个特殊长度存贮页边长度值。这个长度是私有的内部命令，遵从 295 页上的约定。

```
\newlength{\FP@margin}
\DeclareOption{in}{\setlength{\FP@margin}{1in}}
\DeclareOption{cm}{\setlength{\FP@margin}{1.5cm}}
```

另外，四个标准的页面样式也做为选项名。它们将会设置两个内部 boolean 开关和页面样式。

```
\newboolean{FP@plain}
\newboolean{FP@empty}
\DeclareOption{plain}{\setboolean{FP@plain}{true}
                      \setboolean{FP@empty}{false}
        \pagestyle{plain}}
\DeclareOption{empty}{\setboolean{FP@plain}{true}
                      \setboolean{FP@empty}{true}
```

```
        \pagestyle{empty}}
\DeclareOption{headings}{\setboolean{FP@plain}{false}
                         \setboolean{FP@empty}{false}
        \pagestyle{headings}}
\DeclareOption{myheadings}{\setboolean{FP@plain}{false}
                         \setboolean{FP@empty}{false}
        \pagestyle{myheadings}}
```

最后，就执行选项的缺省集，再处理选定的选项，在这种情形中，是按指定的顺序进行的。只所以这样做，就是如果给出了不只一个页面样式，最后那个起作用。

```
\ExecuteOptions{in,plain}
\ProcessOptions*
```

下面开始进行计算。首先对于 plain 和 empty 样式，没有页眉行，因此把相应的参数取值为零。这时 FP@plain 为 ⟨真⟩ 。然后把为页脚行保留的空间设为零 (此时 FP@empty 为 ⟨真⟩)。对于 headings 和 myheadings ，这些空间维持不变，因为在这些样式中通常第一页就是 plain 页 (有页脚)。

```
\ifthenelse{\boolean{FP@plain}}
   {\setlength{\headheight}{0pt}
    \setlength{\headsep}{0pt}}{}
\ifthenelse{\boolean{FP@empty}}
   {\setlength{\footskip}{0pt}}{}
```

在页边，页眉和页脚空间已设置好后，就可以进行实际的计算。注意打印驱动程序要在左边和上边留下 1 英寸的边界，因此要从 \topmargin 和 \oddsidemargin 中减去这个长度。

```
\setlength{\textwidth}{\paperwidth}
\addtolength{\textwidth}{-2\FP@margin}
\setlength{\oddsidemargin}{\FP@margin}
\addtolength{\oddsidemargin}{-1in}
\setlength{\evensidemargin}{\oddsidemargin}

\setlength{\textheight}{\paperheight}
\addtolength{\textheight}{-\headheight}
\addtolength{\textheight}{-\headsep}
\addtolength{\textheight}{-\footskip}
\addtolength{\textheight}{-2\FP@margin}
\setlength{\topmargin}{\FP@margin}
\addtolength{\topmargin}{-1in}
```

注意 \paperheight 和 \paperwidth 是被纸张尺寸选项设置成恰好等于纸张的完全尺寸的。如果这种指定是错误的，那么当然最后的结果也不会正确。

至此就完成了宏包文件 fullpage.sty 的所有代码。下面是调用它的一个示例：

```
\documentclass[a4paper,12pt]{article}
\usepackage[headings]{fullpage}
. . . . . .
```

§C.3.2 重新设计页眉和页脚

LaTeX用户经常要做的一件事，可能就是调整每页上页眉和页脚的显示了。标准的 LaTeX虽然提供了有限数目的 页面样式(3.2 节)，但是通常是不够用的。它们中间最有弹性的就是 myheadings 页面样式了，它可以让作者利用 \markright 和 \markboth 来确定活动页眉的文本内容。然而，包括字体样式和页码位置等一般性格式仍然由 LaTeX决定。

我们下面给出一个例子，说明如何为一个大工程文档修改页眉，使得其包含特定的信息。除了标题和节的简写形式，章节号与页码外，还可以有文档序列号，日期，以及版本号。如果能设计一个广义文档页面样式，可以在页面定义外面指定这些条目，那就非常方便了。

由于这里的编码只适用于 article 类，因此我们就创建一个新类，名称为 dochead，它输入 article.cls，然后定义一个新的页面样式。实际上这个类中可以包含更多的其它特殊功能，我们的新页面样式只是其中一个。

我们还是首先声明所需要的 TeX版本，并标明类文件。

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{dochead}[1994/09/28 1.0 (PWD)]
```

这个类没有自己的新选项，只是把指定的选项传递给后台的 article 类。然而，它自己自动选择 12pt 和 a4paper 选项。

```
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{article|
\ProcessOptions
\LoadClass[12pt,a4paper]{article}
```

接下来就是用来输入页眉中四部分信息(短标题，日期，序列号，以及版本)的命令。其中每条命令都把自己的参数存贮在一条内部的 \DH@ 命令里，这几条内部命令起初设成空。最后，声明这些输入命令只能用在导言中，因为如果在正文已开始后再调用它们，就会导致灾难性的后果。

```
\newcommand{\DocTitle}[1]  {\renewcommand{\DH@title}{#1}}
\newcommand{\DocDate}[1]   {\renewcommand{\DH@date}{#1}}
\newcommand{\DocNumber}[1] {\renewcommand{\DH@number}{#1}}
\newcommand{\DocVersion}[1]{\renewcommand{\DH@version}{#1}}
```

```
\newcommand{\DH@title}{}   \newcommand{\DH@date}{}
\newcommand{\DH@number}{}  \newcommand{\DH@version}{}
\@onlypreamble{\DocTitle}  \@onlypreamble{\DocDate}
\@onlypreamble{\DocNumber} \@onlypreamble{\DocVersion}
```

\@onlypreamble 是一条 LaTeX的内部命令。

我们下面定义新的页面样式 dochead，也就是说我们必须创建一个叫 \ps@dochead 的命令，它由 \pagestyle{dochead} 执行。这条命令要做的事情就是重定义四条内部命令 \@oddhead, \@evenhead, \@oddfoot, \@evenfoot。我们把页设成空的，奇偶页眉一样。页眉是一个小页，用的是页面宽度，由一个表格组成，表格中就是相关的文档信息。

```
\newcommand{\ps@dochead}{%
 \renewcommand{\@oddhead}{%
    \begin{minipage}{\textwidth}\normalfont
      \begin{tabular*}{\textwidth}{@{}l@{\extracolsep{\fill}}%
                                   l@{\extracolsep{0pt}:~}l@{}}%
       \DH@number      & Version & \DH@version \\
       \DH@title       & Section & \thesection \\
       Date:~\DH@date  & Page    & \thepage
      \end{tabular*}\vspace{0.5ex} \rule{\textwidth}{0.6pt}%
    \end{minipage}}
 \renewcommand{\@evenhead}{\@oddhead}
 \renewcommand{\@oddfoot}{}
 \renewcommand{\@evenfoot}{}
}
\pagestyle{dochead}
```

最后那行就激活了新的页面样式。

还需要增加 \headheight 和 \headsep 的尺寸，因为我们这里的页眉要比通常的高很多。我们选取的高度是通常行距的 3.5 倍。

```
\setlength{\headheight}{3.5\baselineskip}
\setlength{\headsep}{3em}
```

我们下面进一步对上述定义进行修改。类似于上面这样的文档，它通常希望页码是在一节内部进行编号的。因此，我们要重定义 \thepage 命令，使其从页码记数器中显示页码 (7.1.4 节)，而且修改 \section 命令，使得它开始一个新页，并重设页码记数器。这要首先保存 \section 的当前定义，然后才进行重定义。

```
\let\DH@section=\section
\renewcommand{\thepage}{\thesection-\arabic{page}}
```

```
\renewcommand{\section}{\newpage\setcounter{page}{1}\DH@section}
```

注意这里新的 \section 命令要调用保存在内部命令 \DH@section 中的原来定义。

如果上述代码保存在一个叫 dochead.cls 的文件中，那么下面这样的主文件就可以调用它：

```
\documentclass{dochead}
\DocTitle{Spacecraft Cleanliness}
\DocNumber{ESA--XY--123}
\DocDate{1995 Feb 26}
\DocVersion{3.1}
\begin{document}
. . . . . .
```

那么在第 3 节中第 4 页上的页眉就如下所示

ESA–XY–123 Version: 3.1
Spacecraft Cleanliness Section: 3
Date: 1995 Feb 26 Page : 3-4

以这个例子为榜样，要为其它应用创建不同条目的页眉就应该不会是一件困难的事情。

§C.3.3 为其它语言改编 LaTeX

标准 LaTeX是基于英文设计的，因此在很多地方会自动显示出英文单词，如 'Abstract' 或 'Contents' 。虽然确实可以显示出有重音的字母，但是对于一长节文本而言，这样的输入方法就太复杂了。现在对于英文单词的断词也可以有带重音的字母。因此，如果要为其它语言改编 LaTeX ，那么应该进行三方面的修改。为了降低复杂性，它们是：

1. 用另外语言的翻译替换原来的英文单词，
2. 为重音或特殊标点符号定义简化的命令，
3. 改变断词式样。

D.1.3 节处理的就是这种多语言 LaTeX情形。我们这里只是创建一个宏包，它可以处理一种非英文语言和英文本身。

在 D.1.4 节和 D.4 节讨论了第 3 点。如果 TeX的版本要早于 3.0 ，只可能保存一组断词式样，从而不可能进行真正的语言切换。

利用 TeX相当较近的版本，可以很好地做到第 1 点。起初，类似于 'Abstract' 和 'Contents' 这样的单词是显式包含在生成这些章节的命令中。在欧洲，根据 Technical University of Vienna 的 H. Partl 的建议，来自于 Darmstadt 的 J. Schrod 建立起 LaTeX一个修正版本，在这个版本中，上面所有英文单词都被类似于 \abstractname 和 \contentsname 这样的命令所代替。因此现在

就非常容易进行相应于某语言的修改，因为只要改变这些名称命令的定义就可以，而不用重定义整个 \abstract 或 \tableofcontents 命令。这些名称命令自从 1991 年 12 月 1 日开始已成为正式 LaTeX的一部分。

相应于世界语的改编

我们选取一个示例，来说明把 LaTeX改编到国际语言 — 世界语中的原则。上面所列的三个问题都要加以考虑；解决方法并不是显而易见的，但相对于改编到法语和德语中而言，复杂性则要低很多。而且，通过选择 *la internacia lingvo* ，这里并没有语言学上的分组，这是一个特别的好处，这也是世界语的基本哲学。

有两个可用的世界语宏包，一个叫 espo.sty，它没有署名，也没有日期，相当初等，另一个叫 esperant.sty，由德国 Mainz 大学的 Jörg Knappen 设计，注明日期为 1991 年 12 月 10 日。作者保证上面这个宏包以及 german.sty 的功能是稳定的。（还有第三个宏包，名称也是 esperant.sty，它是 babel 系统的一部分，我们在 D.1.3 节中描述它。）

世界语使用有 28 个字母的字母表，其包括通常拉丁字母表中除 q, w, x 和 y 外的所有字母，并增加了特殊字母 $\hat{c}$, $\hat{g}$, $\hbar$, $\hat{j}$, $\hat{s}$ 和 $\breve{u}$，分别对应于英语中的 *ch*, *j*，德语中的 *ch*，法语中的 *j*，英语中的 *sh*, *w* 。这些加了重音的字母看成是单独的字符，并不像法语中认为是被装饰的字母，或者德语中认为其它字母组合的取代。抑扬重音在 LaTeX中是用 \^ 得到的，而二全音符是用 \u 生成的。但这里要稍微复杂一些，因为在 h 上的符号位置要有点儿改变，而 j 的点要去掉。而且用 \u{u} 得到 ŭ 要输入五个键。为了给用户简化这种操作， Esperanto 样式生成了一个新的单字符命令 ^，对所有的特殊字母都有效。因此 ^c ^g ^h ^j ^s ^u 的结果就是 $\hat{c}$ $\hat{g}$ $\hbar$ $\hat{j}$ $\hat{s}$ $\breve{u}$ 。

然而，要想得到上面那样的字符，只是把 ^ 改变成主动（命令）字符，并把它定义成 \^ 是不够的。不但要把这条命令设成牢固的 (2.6 节)，而且对于包含 ^ 的单词在断词的时候，语言之间的切换，以及最重要的利用名称命令对英文标题进行翻译的时候，都有问题要解决。

我们这里所给出的例子是 Jörg Knappen 对 LaTeX 2_ε进行改编而设计的宏包 esperant.sty，它全部解决了上面提出的几个问题。这个宏包原来必须在 Plain TeX下执行，做为 LaTeX2.09 的一个样式选项。它利用相当多的 TeX技巧，我们在此不做详细解释，只说明一下最后的结果是什么。

做为一个宏包文件，其开始也是鉴定：

```
\NeedsTeXFormat{LaTeX2e}[1994/06/01]
\ProvidesPackage{esperant}
      [1994/06/08 1.1a2e Daly's adaption of Knappen's]
```

这是对 LaTeX 2ε的一个极小改编，因此没有定义任何选项。我们就直接进入主体，其中第一项利用一个 TeX技巧使得有重音的单词可以有断词。做法是加入一个空的单词断点，但不允许断行。

```
\newcommand{\allowhyphens}{\penalty\@M \hskip\z@skip}
```

第二条语句把字符 ^ 加到某个特殊列表中，而且必须把源文本原样输出。不必知道如何做到这点，只要接受它就可以了。

```
\begingroup
  \def\do{\noexpand\do\noexpand}%
  \xdef\dospecials{\dospecials\do\^}%
  \def\@makeother{\noexpand\@makeother\noexpand}%
  \xdef\@sanitize{\@sanitize\@makeother\^}%
\endgroup
```

然后，定义两条命令，让抑扬字符 ^ 的功能在做为上标运算符的普通角色与做为主动字符，即单字符命令之间切换。

```
\newcommand{\modifiedhaton}{\catcode`\^\active}
\newocmmand{\modifiedhatoff}{\catcode`\^7 }
```

第三步，定义抑扬命令，然后把它变成主动字符，并把它定义成有一个参数的牢固命令。在数学模式 (\ifmmode) 中就调用通常的上标函数 \sp，否则就在处理之前检测参数值是否是特殊字符。如果参数是 U 和 u，就显示二全音符重音；如果是 j，就用没有点的相应字符；如果是 h，就调用重叠函数。

```
\modifiedhaton
\DeclareRobustCommand{^}[1]{\ifmmode\sp{#1}
        \else\ifx#1u\u u\allowhyphens
\else\ifx#1U\u U\allowhyphens
        \else\ifx#1h{h\llap{\^{}}}\allowhyphens
        \else\ifx#1j{\^\j}\allowhyphens
        \else\ifx#1|\discretionary{-}{}{}\allowphyens
        \else{\^{#1}}\allowphyens
\fi\fi\fi\fi\fi\fi}
\modifiedhatoff
```

注意这里又为 ^ 增加了一项功能： ^| 加入一个自由连字符，即建议的单词断点，利用这种方法，单词余下的部分仍可以自动断词。通常在 TeX中，单词断点只能出现在有自由连字符的地方。

第四步，建立三条命令，以定义世界语，美国英语和英国英语中的日期。在解释 \hodiau 和 \hodiaun 这两条命令时需要一点世界语的语法：表示今天的单词是 *hodiaŭ*，例如 *Hodiaŭ estas la 15a de novembro, 1994*，当做为一

封信的日期时，日期通常必须是宾格的 (*la 15an de novembro, 1994*) 。我们对这两种可能都做了考虑。

我们不采用直接重定义 \today 的方法，而是生成几个重定义的命令，我们可以利用它们在世界语、美国英语和英国英语之间来回切换。后两者是用 \providecommand 定义的，主要是为了以防已有宏包对它进行了定义。

```
\newcommand{\dateesperanto}{\renewcommand{\today}
  {la \number\day -an de \ifcase\month\or
  januaro\or februaro\or marto\or aprilo\or majo\or junio\or
  julio\or a\u{u}gusto\or septembro\or oktobro\or novembro\or
  decembro\fi, \space \number\year}%
 \let\hodiaun=\today}
\newcommand{\hodiau}{la \number\day -a de \ifcase\month\or
  januaro\or februaro\or marto\or aprilo\or majo\or junio\or
  julio\or a\u{u}gusto\or septembro\or oktobro\or novembro\or
  decembro\fi, \space \number\year}

\providecommand{\dateUSenglish}{\renewcommand{\today}{%
  \ifcase\month\or January\or February\or March\or April\or
  May\or June\or July\or August\or September\or October\or
  November\or December\fi \space \number\day, \number\year}}

\providecommand{\dateenglish}{\renewcommand{\today}{%
  \ifcase\day\or 1st\or 2nd\or 3rd\or 4th\or 5th\or 6th\or 7th\or
  8th\or 9th\or 10th\or 11th\or 12th\or 13th\or 14th\or 15th\or
  16th\or 17th\or 18th\or 19th\or 20th\or 21th\or 22th\or
  23th\or 24th\or 25th\or 26th\or 27th\or 28th\or 29th\or
  30th\or 31st\fi~\ifcase\month\or
  January\or February\or March\or April\or May\or June\or
  July\or August\or September\or October\or November\or
  December\fi \space \number\year}}
```

接下来要用 LaTeX(即 1991 年 12 月 1 日以后发行的 LaTeX版本) 中的名称命令对章节标题进行翻译。同前面处理日期时一样，我们也不是直接进行，而是用 \captionsesperanto 和 \captionsenglish 进行我们的翻译工作。（注意，下面是把两组标题并列摆放，它们实际上是一个接在另一个后面的。）定义是用 TeX的 \def 命令进行的，因为它并不在意所定义的命令是否已经存在。

```
\def\captionsesperanto{%
\def\contentsname{Enhavo}%
\def\listfigurename{Listo
    de figuroj}%
\def\listtablename{Listo
    de tabeloj}%
\def\abstractname{Resumo}%
\def\partname{Parto}%
\def\chaptername{\^Capitro}%
\def\prefacename{%
     Anta\u{u}parolo}%
\def\appendixname{Apendico}%
\def\refname{Cita\^{\j}oj}%
\def\bibname{Bibliografio}%
\def\indexname{Indekso}%
\def\figurename{Figuro}%
\def\tablename{Tabelo}%
\def\pagename{Pa\^go}%
\def\seename{vidu}%
\def\alsoname{vidu anka\u{u}}%
\def\notesname{Rimarkoj}%
\def\enclname{Aldono(j)}%
\def\ccname{Kopie al}%
\def\headtoname{Al}%
\def\subjectname{Temo}%
}
```

```
\def\captionsenglish{%
\def\contentsname{Contents}%
\def\listfigurename{List
    of Figures}%
\def\listtablename{List
    of Tables}%
\def\abstractname{Abstract}%
\def\partname{Part}%
\def\chaptername{Chapter}%
\def\prefacename{Preface}%

\def\appendixname{Appendix}%
\def\refname{References}%
\def\bibname{Bibliography}%
\def\indexname{Index}%
\def\figurename{Figures}%
\def\tablename{Table}%
\def\pagename{Page}%
\def\seename{see}%
\def\alsoname{see also}%
\def\notesname{Notes}%
\def\enclname{encl}%
\def\ccname{cc}%
\def\headtoname{To}%
\def\subjectname{Subject}%
}
```

这些名称只会出现在特定的类中，而且有些也不是标准的名称。例如，\refname 只能用在 article 类中，在 book 和 report 类中其被 \bibname 所代替，而 \abstractname 也只会用在 article 和 report 类中。 \pagename, \enclname, \ccname 和 \headtoname 只会出现在 letter 类中， \seename 是为 makeidx 宏包准备的。其它命令都是非标准的，只用于特殊用途或改编，它们有： \prefacename, \notesname, \alsoname （ 为 makeidx 准备的) 以及 \subjectname (为 letter 准备的) 。

正如上面所指出的，在世界语中的重音字母是看成单独字符的。这也就是说在某些情况况下以字母为序进行排列时，其应显示为 'A B C Ĉ ...' 或 'a b c ĉ ...' 。为了做到这一点，要定义两个新的记数器类型： \Esper 和 \esper ，其功能类似于 \Alph 和 \alph 。

```
\newcommand{\esper}[1]{\@esper{\value{#1}}}
\newcommand{\Esper}[1]{\@Esper{\vaule{#1}}}
\newcommand{\@esper}[1]{\ifcase#1\or a\or b\or c\or \^c\or d\else
   \@iesper{#1}\fi}
\newcommand{\@iesper}[1]{\ifcase#1\or \or \or \or \or \or
   \or e\or f\or g\or \^g\or h\or h\llap\^{}\or i\or j\or \^\j\or
   k \or l\or m\or n\or o\or p\or r\or s\or \^s\or t\or u\or
   \u{u}\or v\or z\else\@ctrerr\fi}
\newcommand{\@Esper}[1]{\ifcase#1\or A\or B\or C\or \^C\or D\else
   \@Iesper{#1}\fi}
\newcommand{\@Iesper}[1]{\ifcase#1\or \or \or \or \or \or \or E\or
   F\or G\or \^G\or H\or \^H\or I\or J\or \^J\or K\or L\or M\or
   N\or O\or P\or R\or S\or \^S\or T\or U\or \u{U}\or V\or
   Z\else\@ctrerr\fi}
```

下面定义几个语言记数器以跟踪当前语言。其只对 3.0 版本以后的 TEX 才有效，这时把 \language 设置为一个数值，以激活存贮在该语言编号下面的断词式样。这里我们假定英语断词式样保存在零号语言中，而世界语式样保存在一号语言中。（在 D.1.3 节介绍的 babel 系统使用了一种更安全的方法进行语言编号赋值与选择。）\selectlanguage 命令通过调用日期和标题定义命令来打开选定的语言。

```
\newcounter{USenglish}   \setcounter{USenglish}{0}
\newcounter{esperanto}   \setcounter{esperanto}{1}
\newcounter{english}     \setcounter{english}{0}

\providecommand{\selectlanguage}{}
\renewcommand{\selectlanguage}[1]{%
  \ifcase \vaule{#1}%
  \dataUSenglish    \captionsenglish   \or
  \dateesperanto    \captionsesperanto \or
  \dateenglish      \captonsenglish    \fi
  \language=\value{#1}}
```

最后，定义切换世界语的开关，并用来在文档开头处打开世界语。

```
\newcommand{\EsperantoOff}{\modifiedhatoff}
\newcommand{\EsperantoOn}{\modifiedhaton}
\AtBeginDocument{\EsperantoOn\selectlanguage{esperanto}}
```

§C.3.4 作者–年代 引用

在 B.3.1 节我们指出标准的 LaTeX不能处理引用是由作者和年代组成的参考文献样式，而只能处理引用是数字型的。我们在此给出一个宏包和参考文献样式，以解决这个问题。

宏包文件

我们给宏包命名为 authyear 。这是相当广泛的 作者–年代 引用样式宏包 natbib(由 Patrick W. Daly 完成) 的一个非常简化的版本。但它确实演示了最重要的功能。

在 作者–年代 引用机制中，我们需要放在括号内的引用和平铺直述的引用。区分这两种样式的方法是采用放在 \cite 命令中的可省注释参数值来实现的。假设在 thebibliography 中有如下一项条目：

```
\bibitem[{\it Jones et~al.}(1990)]{jone90}
Jones, F. B., Smith, T. E., and Harris, R. E. ...
```

jone90 就是 \cite 命令和标签 (即 {\it jones et~al.}(1990)) 的关键词。对于这种情形，标签中年代两边的括号就相当于一个定界符，从而把作者与年代分开。然后如何处理它们，那就要看如何用 \cite 命令了：

\cite{jone90}	结果为	*Jones et al. [1990]*
\cite[]{jone90}	结果为	[*Jones et al., 1990*]
\cite[page~30]{jone90}	结果为	[*Jones et al., 1990, page 30*]

包围引用的括号可以是圆括号，也可以是方括号；在引用的中标点符号可以是逗号，也可以是分号。

宏包文件的开头仍是鉴别命令。它也需要 ifthen 宏包，因此就紧接着调用了该宏包。

```
\NeedsTeXFormat{LaTeX2e}[1995/06/01]
\ProvidesPackage{authyear}[1995/10/09 2.1 (PWD)]

\RequirePackage{ifthen}
```

下面就准备并执行选项。这些选项将设置各种命令以保存标点符号和括号类型。由于这些存贮命令并不是独有的内部命令，因此用户也可以在任何时候进行重定义。然而，标准选项应该可以满足通常的需要。首先定义存贮命令，然后用选项进行设置，并选定缺省选项 round 和 colon ，最后就执行所选定的选项。

```
\newcommand{\citebegin}{} \newcommand{\citeend}{}
\newcommand{\citesep}{}
\newcommand{\auyrsep}{,}
```

```
\DeclareOption{round}{\renewcommand{\citebegin}{(}
                      \renewcommand{\citeend}{)}}
\DeclareOption{square}{\renewcommand{\citebegin}{[}
                      \renewcommand{\citeend}{]}}
\DeclareOption{comma}{\renewcommand{\citesep}{,}}
\DeclareOption{colon}{\renewcommand{\citesep}{;}}
\ExecuteOptions{round,colon}
\ProcessOptions
```

如果用下面定义的命令对内部命令 \AY@cite 进行了格式化，那么它就会显示出实际的引用。其第一个参数就是有格式的引用，第二个就是 \cite 的可省参数值，即可省的补注。如果没有给出这个补注， #2 就是句号 (见下面的 \cite)，引用显示时并不放在括号内 (即平铺直述的引用)；否则，要把它以及补注包装在括号内 (放在括号内的引用)。

```
\newcommand{\AY@cite}[2]
   {\ifthenelse{\equal{.}{#2}}
     {#1}
     {\citebegin#1%
      \ifthenelse{\equal{#2}{}}
        {}
        {, #2}%
      \citeend}}
```

实际上主要工作是由 \cite 完成的。这里所发生的事情实际上是基于 LaTeX 2$_\varepsilon$对 \cite 的定义这一 LaTeX内部技术。新的 \cite 命令有两个参数值，第一个就是可省的补注。然而，有补注就意味着要使用放在括号内的引用。补注实际上可以是空的，这得到一个放在括号内的没有补注的引用。为了做到这一点，(可省的) 第一个参数值的缺省值为句号。它表示调用 \cite 时根本没有可省参数值，更不用说是一个空参数值了。

主要参数值 (#2) 就是一串引用关键词。它们必须一个一个地用 \@for \do 进行处理。并把 \@citeb 相继设成列表中的关键词。 \@iden 所在的行从 \@citeb 中去掉了前导空格；然后在辅助文件中为 BibTeX生成一项条目；进行一次测试，看看关键词是否有相应的转换 (保存在 \b@\@citeb 中)，如果没有的话就显示出一条警告消息。此时，就有了新的功能：调用 \AY@parse 对关键词进行解释，把作者部分放到 \AY@author 中，年代部分放到 \AY@year 中；在平铺直述引用 (#1=.) 中，要把这两部分放在一起，年代放在括号内。最后，调用引用显示命令 \AY@cite，第一个参数值为一串转换后的引用，第二个参数是 \cite 的可省参数值。注意在列表中每项引用前面加上了 \AY@sep；它起初是空的，但在第一项引用后变成 \citesep。这条命令可以在引用之间

插入逗号或分号。

```
\newcommand{\AY@sep}{}

\renewcommand{\cite}[2][.]{%
  \renewcommand{\AY@sep}{}%
  \AY@cite{\@for\@citeb:=#2\do
    {\edef\@citeb{\expandafter\@iden\@citeb}%
      \ifthenelse{\boolean{@filesw}}
        {\immediate\write\@auxout{\string\citation{\@citeb}}}
        {}%
      \@ifundefined{b@\@citeb}
      {\AY@sep{\reset@font\bfseries ?}\G@refundefinedtrue
       \@latex@warning
        {Citation `\@citeb' on page \thepage \space undefined}}
      {\AY@parse{\@citeb}%
        \ifthenelse{\equal{.}{#1}}
   {\AY@sep{\AY@author} \citebegin\AY@year
    \renewcommand{\AY@sep}{\citeend\citesep\ }}
   {\AY@sep{\AY@author}\auyrsep\ \AY@year
    \renewcommand{\AY@sep}{\citesep\ }}}}% ends \do
\ifthenelse{\equal{.}{#1}}
    {\citeend}
    {}%
  }{#1}}
```

下面的命令对引用关键词转换后的结果进行解析，提取出作者和年代消息，它做为关键词调用时的参数。展开 (转换) 后的结果用做 `\AY@split` 的参数值。在此必须加上一个没有内容的 `()` 以防转换后在括号内并不包含年代。`\let\protect=` 是为了得到牢固命令的特殊处理，用它是为了防止在此处就被展开，从而导致严重破坏。

```
\newcommand{\AY@parse}[1]{{\let\protect=\@unexpandable@protect
     \xdef\@tempa{\@nameuse{b@#1}}%
     \expandafter\AY@split\@tempa()@}}
\def\AY@split#1(#2)#3@{\gdef\AY@author{#1}\gdef\AY@year{#2}}
```

这就完成了对 `\cite` 及相关内部命令的重定义。我们现在来修补环境 `thebibliography` 。不幸的是，这意味着需要重定义整个环境，因为这里并不存在便利的内部命令可以修改。而且，我们需要间断是否处在 `article` 类中，即有没有 `\chapter` 命令，因为这确定了索引清单是否在 `\section*` 或

\chapter* 中。

```
\@ifundefined{chapter}
  {\newcommand{\AY@bibsect}{\section*{\refname
   \@mkboth{\MakeUppercase{\refname}}{\MakeUppercase{\refname}}}}}
  {\newcommand{\AY@bibsect}{\chapter*{\bibname
   \@mkboth{\MakeUppercase{\bibname}}{\MakeUppercase{\bibname}}}}}
\providecommand{\refname}{References}
\providecommand{\bibname}{Bibliography}
```

注意: \MakeUppercase 命令 (以及这里没有用的 \MakeLowercase 命令) 可以改变其参数值的大小写，相比于在 LaTeX2.09 中所用的 TeX命令而言，这两条命令要更好一些。

我们现在来重定义 thebibliography 环境，使得标签并不会显示出来，第一行左对齐，所有后面的行向右缩进 1 em 。我们这里也要利用上面定义的 \AY@bibsect 命令。

```
\renewenvironment{thebibliography}[1]
{\AY@bibsect
  \setlength{\parindent}{0pt}\list{}
  {\setlength{\leftmargin}{1em}%
   \setlength{\itemindent}{-\leftmargin}}
  \ifthenelse{\boolean{@openbib}}
    {\renewcommand\newblock{\newline}}
    {\renewcommand\newblock{\hskip .11em plus.33em minus.07em}}
  \sloppy\clubpenalty4000\widowpenalty4000
  \frenchspacing}
{\renewcommand{\@noitemerr}{\@latex@warning
      {Empty `thebibliography' environment}}%
  \endlist\vspace{-\lastskip}}
```

最后要做的一件事就是中止 \bibitem 命令的功能，不让它显示出标签。这里我们只需修改一条内部命令，并忽略其参数。

```
\renewcommand{\@biblabel}[1]{\hfill}
```

参考文献样式文件

上面完成了 authyear.sty 宏包文件。然而，只有当 thebibliography 环境中的 \bibitem 命令具有本节开始所示形式时才有效。此时参考文献可以手工得到，也可以利用 BibTeX，但必须存在一个正确格式的参考文献样式文件 (.bst) 才可以。在标准 BibTeX宏包中没有这样的样式文件，但只要对 alpha.bst 文件稍做一点儿改变就可以了。我们下面只是列出做到这点的必

要代码，而并不详加解释，因为BibTeX用的是与众不同的语言。

把alpha.bst文件复制到一个叫authyear.bst的新文件中，并在其中寻找文本FUNCTION {output.bibitem}。这个函数的第五行是"]{" write$；把它改成")]{" write$，即在右方括号前面加上右小括号。

然后，找到FUNCTION {format.date}，并在这个函数最后那个右大括号前面加上extra.label *，即现在为

```
  if$
  extra.label *
}
```

现在寻找文本FUNCTION {format.lab.name}。把这个函数的文本(从开头到FUNCTION {author.key.label}之前的所有内容)替换为下面的代码

```
FUNCTION {format.lab.names}
{ 's :=
  s #1 "{vv~}{ll}" format.name$
  s num.names$ duplicate$
  #2 >
    { pop$ " et~al." * }
    { #2 <
        'skip$
        { s #2 "{ff }{vv }{ll}{ jj}" format.name$ "others" =
            { " et~al." * }
            { " and " * s #2 "{vv~}{ll}" format.name$ * }
          if$
        }
      if$
    }
  if$
}
```

最后，查找FUNCTION {calc.label}，并定位到包含duplicate$的那行。在其后加上emphasize "(" *。在接下来的一行中，把#2改成#4。因此这些行及相邻行现在应该为

```
  if$
  duplicate$
  emphasize "(" *
  year field.or.null purify$ #-1 #4 substring$
  *
  'label :=
```

(上面的 emphasize 指的是引用中的作者，而不是索引中的作者要用斜体显示。如果不希望这样，那就忽掉它。)

注意：这个新参考文献样式只在 0.99 或以后版本的 BibTeX中才可以用。

为了演示如何使用这一样式，假定在参考文献数据文件 mylit.bib (见 B.2 节) 中有一项条目：

```
@ARTICLE{jone90,
   AUTHOR = {Fred B. Jones and
             Thomas Elwood Smith and
             Richard E. Harris},
   TITLE = {Activity in the Night-Side Ionosphere},
   YEAR = {1990},
   JOURNAL = {J. Geophys. Res.},
   VOLUME = {101},
   PAGES = {1234-1254}  }
```

如果在 LaTeX文件中现在包含如下文本行

```
\documentclass{article}
\usepackage[square]{authyear}
\begin{document}
. . . as shown by \cite{jone90}, it is possible . . .
. . . has been measured \cite[]{jone90}. This is . . .
\bibliographystyle{authyear}
\bibliography{mylit}
\end{document}
```

所得的输出会是

. . . as shown by *Jones et al.* [1990], it is possible . . .

. . . has been measured [*Jones et al.*, 1990]. This is . . .

Reference

Fred B. Jones, Thomas Elwood Smith, and Richard E. Harris. Activity in the night-side ionosphere. *J. Geophys. Res.*, 101:1234–1254, 1990.