



## 第八章 高级功能

本章描述 L<sup>A</sup>T<sub>E</sub>X 用来调整所谓‘文档准备系统’的那些功能，而前面几章主要着重于文本处理。修饰词‘高级’可能会有误导作用，因为这里只是指这些功能对于高效地创建长而复杂的文录是相当重要的。这里的内容包括把一个文档分成几个文件，文档中不同部分的选择处理，章节、插图和公式的交叉引用，自动生成参考文献、索引和汇总，处理不同的字体集合，准备报告材料。

### §8.1 处理文档的各个部分

我们已经多次指出，L<sup>A</sup>T<sub>E</sub>X 文档是由导言和实际的文本两部分组成。对于较短的文档，如初学者通常处理的文档，可以用文本编辑器输入到单个文本中就可以了，而且可以在第一次显示之后再进行修改。当用户积累了足够的经验和信心后，L<sup>A</sup>T<sub>E</sub>X 文档就可能在长度上急速膨胀，甚至有可能想生成长达 1000 页的整本书籍。

理论上这样长的文档是可以保存在一个文件中的，虽然这会使得整个操作变得非常笨拙。对于长文件，文本编辑器的功能大打折扣，而且 L<sup>A</sup>T<sub>E</sub>X 处理起来也会相应地花费很多时间。比较好的主意是把工作分成几个文件，在处理时再由 L<sup>A</sup>T<sub>E</sub>X 把它们合并起来。

#### §8.1.1 `\input` 命令

可以用下面这条命令把另一个文件的内容读进 L<sup>A</sup>T<sub>E</sub>X 文档：

`\input{ 文件名 }`

这里另一个文件的名称是 `文件名.tex`。只需要指定另一个文件的基本名，不需要扩展名 `.tex`。在 L<sup>A</sup>T<sub>E</sub>X 处理过程中，包含在这第二个文件中的文本将会被包含进来，放到第一个文件给出命令的地方。

`\input` 命令的结果与直接把文件 `文件名.tex` 中的内容输入到文档文件中该处是完全一样的。这条命令可以放在文档的任何地方，既可以放在导言中，也可以放在正文部分。

由于 `\input` 命令可以放在导言中，因此可以把整个导言本身放到单独一个文件中。这样 L<sup>A</sup>T<sub>E</sub>X 实际处理的文件甚至可以只包含命令 `\begin{document}` ... `\end{document}`，中间有很多 `\input` 命令。当有一系列的文档使用相同的导言时，把导言放在单独一个文件中就是非常合理的。这样即使修改了导言，也不必在所有文档中进行另外的修改。可以为各种不同类型的处理提供不同的导言文件，然后用 `\input{处理类型}` 选择。

一个用 `\input` 命令读入的文件中也可以包含 `\input` 命令。嵌套深度只受计算机能力的限制。

为了得到所有读入文件的清单，可以把命令

```
\listfiles
```

放在导言中。当处理完毕，这个清单会同时出现在计算机显示器和抄本文件中。版本号和其它的上载信息也会显示出来。这提供了一种检查到底有哪些文件被输入的方法。详情请见 C.2.9 节。

**练习 8.1:** 把前面一致在用的练习文件 `exercise.tex` 中的导言放在单独一个文件 `preamble.tex` 中。把正文分成三个文件 `exer1.tex`, `exer2.tex` 和 `exer3.tex`。那么为了保证 L<sup>A</sup>T<sub>E</sub>X 处理整个练习文本，现在主文件中应该包含什么内容呢？

### §8.1.2 `\include` 命令

把文档分成几个文件，对于输入和编辑来说是比较实用的，但是当通过 `\input` 命令把它们合并起来后，处理的仍是整个文档。这样即使在一个文件中进行了很小的改动，所有的文件都要被重新读入和处理。因此我们希望能够提供一种方法，可以只重新处理被修改的那个文件。

一个比较粗糙的方法是写一个临时性文件，只包含导言（无论如何，它都需要被读入）和一条 `\input` 命令读入那个特定的文件。这种方法的缺陷是所有页码、章节、插图和公式等等的自动编号都是从 1 开始的，因为所有来自于前面文件的信息都没有了。而且，来自于其它文件中的交叉引用也行不通了。

更好的方法是用 L<sup>A</sup>T<sub>E</sub>X 命令

```
\include{文件名}
```

这条命令只能用于文档的正文部分，另外命令

```
\includeonly{文件清单}
```

只能位于导言中，文件清单包含了所有要被读入的文件。文件名之间用逗号分开，后缀 `.tex` 不需要的。这两条命令要结合使用。

如果文件清单中包含文件名，或者在导言中没有 `\includeonly` 命令，那么 `\include{文件名}` 等价于

```
\clearpage \input{文件名} \clearpage
```

然而，如果文件名不包含在文件清单中，`\include` 等价于 `\clearpage`，其中内容并不被读入。

`\include` 命令要比 `\input` 命令少一些普遍性，因为它总是在新页上开始。因此文档应该在开始新页的地方分成文件，比如章与章之间。另一个局限性是 `\include` 命令不可以嵌套：它只能位于主处理文件中。然而，`\input` 命令可以出现在 `\include` 进来的文件中。

`\include` 命令的主要好处在于关于页面、章节和公式编号的附加信息由 `\includeonly` 命令提供的, 因此选择处理时这些计数器的值也是正确的。来自于其它文件中交叉引用信息也是可用的, 因此 `\ref` 和 `\pageref` 命令 (8.3.1 节) 生成正确的结果。所有这些值是由前面一次完整处理 (用 L<sup>A</sup>T<sub>E</sub>X 编译文件) 确定下来的。

如果对被选择处理的文件进行了修改, 导致页码的增加或减少, 后面的文件也需要被重新处理以校正页码。如果增加或去掉了某些章节, 或者公式、脚注和插图等等的编号发生了变化, 也需要进行同样的操作。

例如, 假设文件三 在第 17 页上结束, 但是经过选择处理后, 它现在长度扩展到了第 22 页。但是后接的文件四 还是从第 18 页开始, 所有后面的其它文件也都具有自己的起始页码。如果文件四 被选择处理, 那么它将会得到正确的起始页码, 即根据修改后的文件三 保存的信息, 确定现在是从第 23 页开始。其它依次类推。然而, 如果在文件三 后选择处理的是文件六, 那么它将得到来自于文件五 的起始页码, 而这个页码还没有修正, 因此这之间会差 5 页。对其它结构、计数器也有同样的问题。只有当文件按正确的顺序进行了重新处理, 才能保证它们取正确的值。

虽然有这些限制, 对于长文档 `\include` 命令是相当有用的, 它可以节省相当可观的用机时间。长文档在输入和编辑时通常要分很多步。`\include` 命令可以使得在很短时间内有选择地重处理改动之处, 即使编号系统工作不正常也可以。随后进行一次完整的处理, 即去掉导言中的 `\includeonly` 命令就可以做到这一点。

用 `\include` 读入的文件中不能包含任一 `\newcounter` 声明。这并不是一个过分的限制, 因为通常它们就放在导言中。

本书的每一章就是用单独的文件输入的, 它们名称分别是 `chap1.tex`, `chap2.tex`, ...。处理文件本身就包含如下文本:

```
\documentclass{book}
. . . . .
\includeonly{...}
\begin{document}
\frontmatter \include{toc}
\mainmatter
\include{chap1} . . . \include{chap8} . . .
\backmatter \printindex
\end{document}
```

这里文件 `toc.tex` 就是由如下文本组成:

```
\setcounter{page}{7}
\tableofcontents \listoftables \listoffigures
```

通过在 `\includeonly` 命令中填加适当的项，就可以有选择地处理各章：例如，利用 `\includeonly{toc,chap8}` 就可以只处理目录表和第 8 章。

### §8.1.3 终端输入和输出

有的时候希望在处理过程中  $\text{\LaTeX}$  能在计算机终端上显示出一条消息。这可以用如下命令来做到：

```
\typeout{信息}
```

这里的 信息 就代表要显示在计算机屏幕上的文本。当  $\text{\LaTeX}$  处理到这条命令时，就会显示出这段文本。同时，消息也写入到 `.log` 文件中 (8.9 节)。

如果 信息 中包含用户定义的命令，那么它要被解释，并把翻译后的结果显示在屏幕上。对  $\text{\LaTeX}$  命令也要做同样的事情。如果命令（无论是用户定义的，还是  $\text{\LaTeX}$  命令）并不是可显示的，那么这会造成可怕的后果。要显示命令名，就在命令的前面加上 `\protect` 命令。

命令

```
\typein[\命令名]{信息}
```

也会把 信息 显示在终端上，但是它会等待用户从键盘上输入一行文本，用回车结束。如果没有可省参数值 `\命令名`，那么这文本就直接插入在处理过程中。举个例子，这样我们就可以重复利用一封信的相同文本，而写上几个不同的地址，只要每次从键盘上输入不同信息就可以了。假设文本是如下组成的：

```
Dear \typein{Name:}\ ...
```

那么就会在屏幕上显示：

Name:

`\@typein=`

此时，我们可以输入收信人的姓名。如果的接连几次处理中输入了 ‘George’，‘Fred’ 和 ‘Mary’，那么结果就是同样的信件内容，只是称呼不一样，它们是 ‘Dear George’，‘Dear Fred’ 和 ‘Dear Mary’。

如果 `\typein` 命令包含可省参数值 `\命令名`，那么就认为它等价于

```
\typeout{信息}\newcommand{\命令名}{输入的定义}
```

这样就会交互式地把定义保存在名称为 `\命令名` 的命令中，可以在文档的其它部分同其它  $\text{\LaTeX}$  命令一样调用和执行它。

当对  $\text{\LaTeX}$  方法有了一定经验后，那么就会发现利用 `\typein` 命令进行交互处理是可行的。例如，如果导言中包含

```
\typein[\files]{Which files?}
\includeonly{\files}
```

那么就会在屏幕上显示如下信息：

Which files?

$\backslash\text{files=}$

$\text{\LaTeX}$  等待用户输入一个或多个要处理的文件名（中间用逗号分开）。这就避免了每次必须用编辑器修改主处理文件。

本书就是用这种方法结合上一页的主处理文件生成的。

当一封礼仪信件要送给不同的收信人时，也可以采用类似的过程。我们可以交互式地输入收信人的姓名、地址，甚至包括称呼。整封信由  $\text{\LaTeX}$  用这种方法处理，用户从键盘上输入各项信息。

警告： $\backslash\text{typein}$  命令不能用作其它  $\text{\LaTeX}$  命令的参数值！然而它可以出现在类似于 `minipage` 这样的环境中。

练习 8.2: 修改练习 8.1 中的主文件，使得文件 `exer1.tex`, `exer2.tex` 和 `exer3.tex` 可以用  $\backslash\text{include}$  命令读入。而且你可以交互式地确定要处理的文件。

练习 8.3: 生成下面这种结构：

Certificate  
Olympic Spring Games  
Waterville 1992  
Finger Wrestling

<b>Gold</b>	A. T. Glitter	AUR	7999.9	Points
<b>Sliver</b>	S. Lining	ARG	7777.7	Points
<b>Bronze</b>	H. D. Tarnish	CUP	7250.0	Points

使得在屏幕上依次显示下列要求：

信息	命令 =	输入
Sport:	$\backslash\text{@typein=}$	Finger Wrestling
Unit:	$\backslash\text{unit}$	= Points
Gold:	$\backslash\text{@typein=}$	A. T. Glitter
Country:	$\backslash\text{@typein=}$	AUR
Value:	$\backslash\text{@typein=}$	7999.9
Silver:	$\backslash\text{@typein=}$	S. Lining
.	.	.
.	.	.
.	.	.

这样可以交互式生成所需的条目。第三列中包含生成上面输出所需的输入。用不同的条目重复上面的程序。尽你能力想像输入。

## §8.2 在 $\text{\LaTeX}$ 中包含 $\text{\TeX}$ 命令

$\text{\TeX}$  应用的快速扩展主要归功于  $\text{\LaTeX}$  的实用性，因此有些用户可能根本不知道  $\text{\TeX}$ 。他们认为  $\text{\LaTeX}$  就是一个单独可执行的程序，是与  $\text{\TeX}$  并存

的。实际上， $\text{\LaTeX}$ 只是用户与  $\text{\TeX}$ 处理程序之间的一个界面，它显著简化了  $\text{\TeX}$ 操作。它把输入的逻辑结构翻译成起作用的  $\text{\TeX}$ 命令，在内部通过调用  $\text{\TeX}$ 来处理它们。

因此可以在  $\text{\LaTeX}$ 内部包含纯粹的  $\text{\TeX}$ 命令。这理所当然地适合于所有  $\text{\TeX}$ 原语命令。除了大约 300 条  $\text{\TeX}$ 原语命令外，还有另外 600 个左右的定义在 Plain  $\text{\TeX}$ 格式中的宏。运行  $\text{\LaTeX}$ 和 Plain  $\text{\TeX}$ 之间的正式差别就在于装载的格式文件是 `latex.fmt`（或者  $\text{\LaTeX}2.09$  中的 `lplain.fmt`），而不是原来的  $\text{\TeX}$ 格式文件 `plain.fmt`。然而，由于  $\text{\LaTeX}$ 格式包含了 Plain  $\text{\TeX}$ 中绝大多数宏定义，注意并不是全部，绝大多数  $\text{\TeX}$ 宏也可以在  $\text{\LaTeX}$ 内部调用。

那些不能用在  $\text{\LaTeX}$ 中的  $\text{\TeX}$ 宏列在 ?? 节。那些就是唯一在  $\text{\LaTeX}$ 中不能用或者功能有些不同的  $\text{\TeX}$ 命令。

尽管可以在  $\text{\LaTeX}$ 文档中使用 (Plain)  $\text{\TeX}$ 命令，我们还是推荐只要有可能，就尽量用高级命令。这是对可移植性和稳定性的最好保证。然而，由于有很多精细的功能（或技巧）只能在  $\text{\TeX}$ 的层次上得到，因此禁止这种应用又是相当不明智的。关于编码指南方面的详细讨论请见 C.1.4 节，在 C.2.12 节中有一些有用的  $\text{\TeX}$ 命令。

### §8.3 正文内的引用

在长的文本中我们经常需要引用在其它地方进行了描述的章、节、表格和插图，或者页。同时也需要生成一个索引记录，它是对整篇文档中特定关键词的引用。在电子文本处理以前的时代中，如此的交叉引用和索引意味着要耗费作者或秘书大量的工作。现在计算机可以承担这一负担的绝大部分。

在以前岁月里，引用前面文本部分的页码虽然是费事的，但总是可行的。但是为了说明将要讲述的内容，引用还没有写出来的部分就只能局限于章节编号了，因为页码还不知道，或者留下空白以备将来填上页码。

编写一本书通常是一个渐进的按步就班的工作。手稿可能根本就不是按顺序写的，而且当初稿完成后，由于基于作者新的考虑或者来自于评论者的有见地的建议，有可能对它进行很大的修改。修订、删除或插入某些节，甚至某些章都是完全有可能的，更不用说有可能交换文本某些部分的顺序了。

$\text{\LaTeX}$ 把所有这些由于大改动造成的问题都变成了历史。无论作者进行了怎样的改动，交叉引用和索引记录所需要的信息都被保存起来，以供在正文中任何地方使用。

#### §8.3.1 交叉引用

前面有多处地方已经提到过，命令

```
\label{ 记号 }
```

用来给它所位于的文本中该点设置一个标记,以便在其它地方引用这一文本。区别标记的符号是记号文本,它可以是字母、数字和字符(除了那些表示单个字符命令的特殊符号: `\# $ % & ^ _ { }`)的任意组合。

`\label` 命令被调用时所处的页码可以用下面命令来显示:

`\pageref{记号}`

其可以位于文档的任何地方。

如果 `\label` 命令在章节命令后面给出,或者位于 `equation`, `eqnarray` 或 `enumerate` 环境中,或者是在 `figure` 或 `table` 环境中的 `\caption` 的参数值中,命令

`\ref{记号}`

就会用正确的格式显示出记号被定义处的章节、公式、插图、表格或枚举的编号。对于 `enumerate`, 显示出来的编号是 `\label` 所处的 `\item` 命令生成的编号。对于由 `\newtheorem` 命令创建的定理类型结构,如果 `\label` 命令出现在定理命令的文本中,那么也可以引用它。例如,在 70 页上 Bolzano-Weierstrass 定理的内容中用 `\label` 放上记号 `bo-wei`:

```
\begin{theorem}[Bolzano-Weierstrass]
    \label{bo-wei}...\end{theorem}
```

因此输入文本

`Theorem~\ref{bo-wei} on page~\pageref{bo-wei}`

将得到输出 ‘Theorem 1 on page 70’。类似地,输入文本

```
for Table~\ref{budget95} on page~\pageref{budget95},
see also Section~\ref{sec:figref}
```

结果为 ‘for Table 6.1 on page 159, see also Section 6.6.6’, 因为那一节包含记号 `\label{sec:figref}`。

标记命令 `\label` 可以位于章节命令的参数值内,也可以位于该节正文中的其它地方。由相应的 `\ref` 命令显示的编号是 `\label` 出现的最内层那节的编号。为了避免可能出现的混淆,我们建议把 `\label` 命令就放在被引用章节命令后面。

如果安装了 L<sup>A</sup>T<sub>E</sub>X, 那么处理文件 `labl.st.tex`— 就可以得到一张清单,其中包括所有的记号,记号转换后的结果,以及页码。

了解交叉引用信息的管理方式是相当有用的。实际上 `\label` 命令就是把记号文本连同相应记数器的值和当前页码写到一个辅助文件中,这个文件的基本名与正在处理的文档文件相同,后缀为 `.aux`。`\ref` 和 `\pageref` 命令就是从这个 `.aux` 文件中得到相关信息的,这条命令是由 `\begin{document}` 命令开始读入的。在创建目录表时出现的情形,这里也会发生:在第一次运行过程中, `.aux` 文件并不存在,因此不会输出任何交叉引用信息;而只是收集信息,并在第一次运行结束时写入一个新的 `.aux` 文件中。如果辅助文件已



发生了改变，需要再运行一次，在本次处理结束时会给出一条警告信息。

### §8.3.2 参考文献的引用

在 4.3.6 节已讲述了参考文献的创建及对它的引用，并在 323 页上给出了演示，这里重复讲述，只是为了给出完整的对交叉引用的讨论。参考文献是如下生成的：

```
\begin{thebibliography}{ 标签样本 }
  \bibitem[ 标签一 ]{ 关键词一 } 文本条目一
  \bibitem[ 标签二 ]{ 关键词二 } 文本条目二
  . . . . .
\end{thebibliography}
```

在 4.3.6 节解释了参数值的意义，这里只解释一下引用关键字，不再重复其它各项。标记 `关键词` 扮演了 `\label` 命令中 `记号` 同样的角色。而且它可以是字母、数字和字符的任意组合，只是不能用逗号。在正文中对参考文献的引用是用如下命令给出的：

```
\cite[ 附加信息 ]{ 关键词 }
```

它把相应 `\bibitem` 命令的标签放在中括号内。输入：

```
For additional information about \LaTeX\ and \TeX\
see~\cite{lamport} and \cite{knuth, knuth:a}.
```

那么结合在 4.3.6 节的参考文献样例，就会得到如下输出：

For additional information about  $\text{\LaTeX}$  and  $\text{\TeX}$  see [1] and [6,6a] .

引用标记 `lamport`, `knuth` 和 `knuth:a` 被转化成相应的参考文献中的引用标签。

如果在 `\cite` 命令中包含了可省参数值 `附加信息`，那么这块文本就会加在标签的后面，但仍然是在中括号内。

```
The creation of a bibliographic database is described in
\cite[Appendix B]{lamport}, while the program \BibTeX\
itself is explained in \cite[pages 74,75]{lamport}.
```

The creation of a bibliographic database is described in [1, Appendix B], while the program `BIBTeX` itself is explained in [1, pages 74,75].

在 `thebibliography` 环境中作者可以利用 `\bibitem` 命令逐项建立参考文献。另外还有一个单独的程序 `BIBTeX`，它通过从一个或多个参考文献数据库中搜索出现在 `\cite` 命令中的 `关键词` 标记自动生成参考文献。这里的关键词必须与数据库中的一致。

在正文中没有引用的项也可以包含在参考文献中。这是利用下面的命令：

```
\nocite{ 关键词 i, 关键词 j, ... }
```

把它放在正文任何地方都可以，这里 关键词 $i$ , 关键词 $j$ , ... 就是这些附加项的关键词。参考文献自身现在是用命令

```
\bibliography{ 数据库  $a$ , 数据库  $b$ , ... }
```

生成的，这里 数据库 $a$ , 数据库 $b$ , ... 就是包含要被搜索的参考文献数据库文件的基本名。这些文件的扩展名为 `.bib`。在 323 页上有一个关于如何使用 BibTeX 的例子。

在附录 B 中讲述了如何建立参考文献数据库，如何使 BibTeX 程序正常运行。概略地讲，第一次在 LaTeX 上运行之前，必须用文档基本名单独执行一次 BibTeX 程序。然后，LaTeX 必须至少执行两遍，第一次建立参考文献，建立关键词与标签之间的对应关系，第二次在 `\cite` 命令的关键词地方，插入标签。如果加进了新的引用或者去掉了原来的引用关键词，必须再执行一次 BibTeX。

### §8.3.3 索引记录

LaTeX 并不会像处理目录表那样自动生成一个索引记录，但它可以帮助作者建立关键词和页码的索引。

索引记录是用下面的环境生成的：

```
\begin{theindex} 索引条目 \end{theindex}
```

这个环境切换到两列页面格式，它具有一个活动标题 *INDEX*。在索引记录第一页上有一个标题 **Index**，它的尺寸在 `book` 和 `report` 文档类同章标题的一样，在 `article` 中同节标题的一样。（更精确地说，实际显示出来的单词是包含在命令 `\indexname` 中，可以重定义它，以适合其它语种。）每一项都是用如下命令

```
\item \subitem \subsubitem 和 \indexspace
```

后接关键词和相应的页码组成的。例如，

commands, 18	<code>\item commands, 18</code>
as environments, 42	<code>\subitem as environments, 42</code>
arguments, 19, 101	<code>\subitem arguments, 19, 101</code>
multiple, 103, 104	<code>\subsubitem multiple, 103, 104</code>
replacement symbol, 20	<code>\subsubitem replacement symbol, 20</code>
used as arguments for sectioning commands, 41, 42	<code>\subitem used as arguments for sectioning commands, 41, 42</code>
	<code>\indexspace</code>
displayed text, 21-32	<code>\item displayed text, 21--32</code>

如果文本项太长，在一行中放不下，就会断开，接在下一行上，并且向内缩进，深度比其它行都大，如上面例子中的 ‘used as argument for sectioning commands, 41, 42’。命令 `\indexspace` 在索引记录中插入一个空白行。

`theindex` 环境只是建立起索引记录的一个适当格式。其中的每一项，包括页码，都必须手工输入。然而，在确定页码方面 LaTeX 可以提供一些帮助。

可以在正文中任何地方调用下面这条命令

`\index{ 索引条目 }`

这里 索引条目 可以是任何文本，但它应是后面索引中的一项。它可以是字母、数字和符号的任意组合，甚至可以包含命令字符和空格。这也就是说命令也可以包含在 索引条目 中，如 `\index{\section}`、`\index{\[]` 或 `\index{%` }。即使从不能用作参数值的命令 `\verb` 也可以包含进来。然而，如果 索引条目 包含命令，那么它就不能再做为其它命令的参数值。另一个限制就是如果在 索引条目 中有一个左大括号，那就必须同时给出右大括号。因此不能用 `\index{\{}`，但可以用 `\index{\{\}}`。

除非导言中包含下面这条命令，否则所有 `\index` 命令都会被  $\text{\LaTeX}$  忽略：

`\makeindex`

这一命令激活所有的 `\index` 命令，并打开一个文件，其基本名与文档文件相同，后缀 `.idx`。现在 `\index` 命令就会向这个文件中写入 索引条目 和当前页码，格式为：

`\indexentry{ 索引条目 }{ 页码 }`

对于最简单的情形，可以输出 `.idx` 文件，从而得到一张索引项与对应页码的清单。利用这张清单，作者就可以用前面给出的那种方式生成 `theindex` 环境中的各项。这应该是文档编写已到了最后阶段时的工作，因为否则页码的改变会导致可怕的更正工作量。

在  $\text{\LaTeX}$  的安装文件中有一个叫 `idx.tex` 的文件，利用它可以提高 `.idx` 文件的可读性。当处理 `idx.tex` 文件（即调用 `latex idx`）时，用户会被提示输入要列出来的 `.idx` 文件名：

```
*****
* Enter idx file's first Name. *
*****
\filename=
```

当从键盘上输入了 `.idx` 文件的基本名后，就会输出两列页面格式，其由出现在 `Page n` 形式页标题下面的 索引条目 文本组成。从这个有格式的列表中再得不到其它的信息，但它要比直接利用输出的 `.idx` 文件容易得多。

即使当导言中没有 `\makeindex` 命令，从而使得 `\index` 命令无效的时候，在编写文档之初就应包含这些文本，这就不失为一个好主意。当文档终稿确定后，再包含 `\makeindex` 命令。最后就可以利用来自于 `.idx` 或 `idx.dvi` 文件的各项来组织 `theindex` 环境。这仍旧是一件烦人的工作，需要相当大的耐心和集中的精力，因为 `\item`、`\subitem` 和 `\subsubitem` 项必须按字母顺序，`.idx` 只是按在文档中的顺序给出了所需信息。

然而，还存在着更好的方法的！有一个可用的程序，它从 `.idx` 文件生成 `theindex` 环境，就如同 `BIB $\text{\TeX}$`  程序生成参考文献一样。在下一节讲述这一

工具。

在 L<sup>A</sup>T<sub>E</sub>X 安装文件中也包含一个软件包 `showidx`，它会把 `\index` 命令中的索引项显示为边注，在它所处的那页边界从页顶开始。当浏览文档的初始版本，以看看所有的索引项是否位于正确的页面或者是否生成了新增项，这一方法是相当有用的。这个软件包是用 `\usepackage` 命令调入的，或者把它放在 `\documentstyle` 的选项列表中。

如果用了 `showidx`，那么在导言中用 `\marginparwidth` 声明（4.10.7 节）增加宽度以适应边注宽度不失为一个好主意。不幸的是，现在书籍格式并不适合于进行这种演示。

### §8.3.4 汇总

所谓汇总，就是一类特殊的索引，它是术语和短语按字母顺序排列，连同其解释一起组成的。为了帮助建立一个汇总，L<sup>A</sup>T<sub>E</sub>X 提供了命令

`\makeglossary`            放在导言中，  
`\glossary{汇总条目}`      放在正文部分

这两条命令的作用方式同组织索引记录是一样的。当在导言中使用了命令 `\makeglossary` 时，就会把各项写到一个后缀为 `.glo` 的文件中。文件中来自于每条 `\glossary` 命令的条目格式为

`\glossaryentry{ 汇总条目 }{ 页码 }`

在 `.glo` 文件中的信息可以用来建立汇总。然而，并不存在着汇总的与 `theindex` 环境等价的 结构，因此可以用 `description` 环境（4.3.3 节）或者特殊的 `list` 环境（4.4 节）。

## §8.4 MakeIndex— 关键词处理器

如果可以使用 `MakeIndex` 程序，那么利用 `theindex` 环境生成一个索引记录的这种烦人苦差事就会免掉。这个程序是 Pehong Chen 在 Leslie Lamport 的帮助下做出的。我们这里只是给出其用法的简略描述。在随程序软件所带的文档中有详细的讲解。

程序 `MakeIndex` 处理 `.idx` 文件，得到输出文件，其基本名与文档文件的相同，但是后缀 `.ind`，这个文件由完整的 `theindex` 环境组成。程序的调用方法为：

`makeindex 基本名 .idx` 或者简单地 `makeindex 基本名`

接着当给出 `\printindex` 命令时，L<sup>A</sup>T<sub>E</sub>X 就经过处理，在该命令处给出索引，这条命令连同 `\see` 命令一起，都是在软件包 `makeidx.sty` 中定义的。因此要用这种方法得到索引记录，需要用 `\usepackage` 命令装载 `makeidx` 软件包。

`MakeIndex` 软件包希望 `\index` 项是下列三种形式之一：

```
\index{ 主条目 }
\index{ 主条目 ! 子条目 }
\index{ 主条目 ! 子条目 ! 子子条目 }
```

每个 主条目, 子条目 或者 子子条目 可以是除 `!`, `@` 和 `|` 字符外的任意组合。在这里惊叹号是各个条目之间的分隔符。如果 `\index` 命令中只有一个主条目, 那么它就会成为 `\item` 命令中的文本。主条目将以字母顺序排列。

如果 `\index` 命令由主条目和子条目组成, 那么子条目的文本给赋给相应主条目下面的 `\subitem` 命令。`\subitem` 的文本也要以字母顺序排列。同样地, 子子条目的文本将会跟在子条目文本下面的 `\subsubitem` 命令后, 并以字母顺序排列。

主条目和子条目也可以包含特殊字符, 甚至可以在排序过程会被忽略的  $\TeX$  命令。其标志是条目的形式为 排序列条目`@`显示条目, 这里 排序条目 是用来进行字母排序的, 而 显示条目 是实际要输出的文本。例如, 本书的索引就是用 `MakeIndex` 生成的, 所有的命令名都做为源文本出现, 在排序时忽略前面的 `\` 字符。这些条目就是用类似于 `\index{put@\verb=\put=}` 这样的文本得到的。

条目也可以用字符序列 `|` (或 `|`) 结尾, 以标志页码范围的开始与结束。例如, 如果

```
在第 136 页上有 \index{picture!commands|}
在第 146 页上有 \index{picture!commands|})}
```

结果就是在主条目 ‘picture’ 下面的子条目 ‘commands’ 后面显示页码为 136-146。

可以不在条目后面显示页码, 代之以对另一条目的引用。例如, 利用

```
\index{space|see{blank}}}
```

就会在索引中得到 ‘space, see blank’。(更准确地说, 是存贮在 `\seename` 命令中的文本显示在 `see` 的地方; 这样可以改变它以适应其它语言。)

对于 `MakeIndex` 程序, `!`, `@` 和 `|` 这三个字符因此具有特殊的作用。为了按原样在文本在显示这三个字符, 不让它们发挥本来的作用, 需要在其前面加上引号符号 `"`。例如, `"!` 表示的就是一个惊叹号, 而不是项分隔符。

因此引号符号就成为第四个特殊符号, 必须用输入 `""` 得到本来的样子。然而, 在 `MakeIndex` 语法中有一条特殊的规则, 那就是在引号前面加上反斜杠, 就会认为它是命令的一部分, 因此在条目中就可以用 `\` 来得到德语重音 (如 `\index{Knappen, J\"org}`)。这条特殊的规则有时会导致另外的问题: 在本书中索引条目 `\!` 就必须输入为 `"\!"`。

可以把页码指定为不同的字体。例如, 本书的索引记录中, 黑体的页码用来表示命令第一次被解释或定义的地方。这是用如下形式的项得到的:

```
在第 9 页上有 \index{blank}
```

在第 17 页上有 `\index{blank|bb}`

第二种情形中把该项的页码作为命令 `\bb` 的参数值。在 `theindex` 环境中的对应行变为

```
\item blank, 11, \bb{20}
```

这一方法只有当已经在导言中进行了 `\newcommand{\bb}[1]{\textbf{#1}}` 时才可行。注意：在 `\index` 项中的竖线并不是排版失误，只是在这种情况下用来代替  $\TeX$  命令符号反斜杠。

也可以定义其它的页码样式，如

```
\newcommand{\ii}[1]{\textit{#1}}
```

```
\newcommand{\zz}[1]{\textsl{#1}}
```

分别表示斜体和 *slanted* 字体。在 `\index` 命令中就是用 `!ii` 或 `!zz` 结尾该项以进行相应的定义。

在 `MakeIndex` 程序中的字母排序通常是按照标准的 ASCII 码，首先是符号，然后是数字，最后是字母，而且大写字母在小写字母前面。空格是包含在符号中。有许多选项可以改变这些规则。当调用程序时如何给出选项与计算机类型有关，这里我们假设在选项字母前面加连字符，如

```
makeindex -g -l 基本名
```

最重要的选项有：

- l 字母顺序：排序时忽略空格；
- c 压缩空格：同通常的  $\TeX$  一样，忽略多于一个的空格或者前导空格。
- g 德语顺序：根据德语顺序，符号在字母前面，小写字母在大写字母前面，然后是数字；而 "a", "o", "u" 和 "s"（在德语版的  $\TeX$  中表示 ä, ö, ü 和 ß）就当做它们是 ae, oe, ue 和 ss 处理，这与标准的德语中的用法一致。
- s 样式定义：允许给出索引格式文件的名称，以包含进来重定义的 `MakeIndex` 功能。

-s 选项读入一个索引样式文件，这个文件由定义 `MakeIndex` 程序输入和输出的命令组成。例如，可以用另外的字符来替换特殊字符 `!`, `@`, `|` 和 `"`，这样新的字符具有原来的作用，而原来的字符成为纯粹的文本。

样式定义文件由一串关键词 - 属性对组成。属性可以在单引号内的一个字符组成（例如 'z'），也可以是在双引号的字符串组成（如 "a string"）。

最重要的关键词，连同其默认定义为：

`quote`    ' "'    定义引号符号；

`level`    '!'    定义项分隔符号；

`actual`    '@'    定义词汇切换符号；

`encap`    '|'    定义页面格式中的虚命令符号。

有相当多的关键词用来定义复杂输出结构。这些关键词在随 `MakeIndex` 程序软件包一起的文档中有讲解。

文件 `makeindex.tex` 包含了 Leslie Lamport 给出的一个简略手册（其中没有提到样式定义）。

## §8.5 新字体选择框架 (NFSS)<sup>2ε</sup>

当刚发明  $\text{T}_{\text{E}}\text{X}$  和  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  的时候，其可用的字体在数量上是很有限制的。正是由于这个原因，在  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2.09$  中所用定义字体的系统弹性很差，因为我们需要改变它们的必要性实在不是很明显。高级字体命令（如 `\large` 和 `\bf`）同最终选定的外部字体名称之间的关联是严格地固定在文件 `lfonts.tex` 中，而这个文件就组合在 `lplain` 格式中。

到了今天，可用的字体很多，其中有些字体可以与标准 CM 字体一起共用，而其它一些则需要取代某些 CM 字体。例如，E.7 节中的 Cyrillic 字体须平行于 Latin 字体加进来，但是要想在标准  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  尺寸命令下自动对它进行操作，则是相当复杂的过程。（我们已经知道这是可以做到的！）同样，安装 PostScript 字体激活了对错综复杂的界面宏的调用，这些宏无疑是来自于 PostScript 驱动软件包，但是它包涵了一批  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  软件开发人员的大量心血。

在  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2.09$  中另一个问题是字体样式和尺寸命令的行为（4.1.5 节和 4.1.2 节）。字体声明 `\rm`, `\bf`, `\sc`, `\sl`, `\it`, `\sf` 和 `\tt` 中的每一个都激活与当前选定的尺寸有关的特定的字体。这些声明中每一个都是排它的。因此 `\bf\it` 的组合实际上就与 `\it` 相同。在这个框架中是无法选择黑体斜体字体的。而且，从 `\tiny` 到 `\Huge` 的尺寸声明都是自动切换到 `\rm`，因此 `\bf\large` 并不生成所期望的黑体大号字体。最后，这里使用的是声明，而不是有一个参数值的命令，这与  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  的基本哲学相矛盾。也就是说，为了强调单个词，输入 `\emph{single}` 就要比 `{\em single\}` 更合理些。（有经验的  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  用户可能会否认这一点，但只是因为他们已完全习惯于原来的那一套。）

在 1989 年，Frank Mittelbach 和 Rainer Schöpf 为  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  提出了一个新字体选择框架 (NFSS)，并在 1990 年给出了一个初步的测试软件包。在 1993 年年中，第二个版本 (NFSS2) 问世，这一版本做了相当大的改进。在 1994 年 4 月正式发行的  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$  中，NFSS 已经在新标准中占稳了脚跟。新字体声明和命令在 4.1.3 节和 4.1.4 中已做了讲述。这里我们更仔细地讲解这个系统，给出一些低级字体选择命令。

### §8.5.1 在 NFSS 中的字体属性

按照 NFSS 框架，每个字符集可以按照如下五种属性分类：编码、族、序列、形状和尺寸，可以用下面的命令来选择这些属性：

```
\fontencoding{ 编码 } \fontfamily{ 族 } \fontseries{ 权与宽度 }
```

`\fontshape{形状}` 和 `\fontsize{尺寸}{基线间距}`

编码属性是在 NFSS 第二版本中新出现的。它定义字体中字符的布局。在表 8.1 中给出了它的可能取值。我们一般不大可能需要在—篇文档中改变编码，当然激活 Cyrillic 字体除外。这个特征就是为了允许程序开发者可以在系统中安装新的字体。

表 8.1: NFSS 编码 框架

编码	描述	字体样例	页码
OT1	来自于 Knuth 的原始文本字体	cmr10	368
OT2	Washington 大学的 Cyrillic 字体	wncyr10	373
T1	Cork(DC) 字体	dcr10	—
OML	T <sub>E</sub> X 数学字母字体	cmmi10	371
OMS	T <sub>E</sub> X 数学符号字体	cmsy10	371
OMX	T <sub>E</sub> X 数学扩展字体	cmex10	372
U	未知编码	—	—

在 `\fontfamily` 命令中的族参数值表示字体的一组基本属性，或来源。对于计算机现代字体，列在 E.3.1 节中所有 serif 字体都属于 cmr 族。而族 cmss 包含 E.3.2 节中的所有 sans serif 字体，族 cmtt 包含 E.3.3 节中的所有打字机字体。在 E.4 节的一些特殊装饰性字体是它们所在族的唯一成员。在表 8.3 中根据族和其它属性列出了所有的 CM 字体。

注意：字体属性‘族’与原来 T<sub>E</sub>X 中同名概念之间没有任何关系。一个 T<sub>E</sub>X 的族可以由在数学公式中做为普通文本、第一层和第二层下标所用的三种不同尺寸的字体组成。

在 `\fontseries` 中的参数值权与宽度表示字符的权（= 粗细程度）和宽度。它们是由表 8.2 中所给的 1 到 4 个字母来定义。

`\fontseries{权与宽度}` 的参数值是由对应于权的字母后接对应与宽度的字母组成。因此 ebsc 表示权为较黑，宽度为较松，而 bx 意味着权为黑，宽度为松。同任何非正常权或宽度组合时，字母 m 可以忽略；如果两者都是正常，那么只要给出 m 就可以了。

在 `\fontshape` 中，参数值形状是 n, it, sl 或 sc 字母组合中的一种，分别表示正常（直立）、斜体、slanted 或小体大写字母。

`\fontsize` 属性命令有两个参数值，第一个参数值尺寸表示字体以点为单位的大小（不显式地给出 pt 单位），而第二个参数值基线间距是从一个基线到下一个基线的竖直距离。第二个参数值成为 `\baselineskip` 的新值（3.2.3 节）。例如，`\fontsize{12}{15}` 选择 12pt 的字体尺寸，行间距为 15pt。（第二个参数值可以给出单位，例如 15pt，但如果没有给出单位，就



表 8.2: NFSS 序列 属性

权类		宽度类		
超轻	ul	超紧	50%	uc
较轻	el	较紧	62.5%	ec
轻	l	紧	75%	c
半轻	sl	半紧	87.5%	sc
中间 (正常)	m	中间	100%	m
半黑	sb	半松	112.5%	sx
黑	b	松	125%	x
较黑	eb	较松	150%	ex
超黑	ub	超松	200%	ux

认为是 pt。)

一旦五个属性都已设置好, 就可以用 `\selectfont` 命令选择字体。这里的新特征是各种属性是彼此独立的。改变其中一个, 并不会改变另一个。例如选择:

```
\fontfamily{cmr} \fontseries{bx} \fontshape{n} \fontsize{12}[15]
```

已经生成了一种直立、黑体、松的罗马字体, 尺寸为 12pt, 行间距 15pt, 那么当后来用 `\fontfamily{cmss}` 选择一种 sans serif 字体, 那么属性中的权和宽度 `bx`, 形状 `n`, 尺寸 12 (15pt) 在进行下一 `\selectfont` 调用时继续有效。

等价地, 可以利用下面这条命令来定义除尺寸外的所有属性, 并同时马上激活字体;

```
\usefont{ 代码 }{ 族 }{ 序列 }{ 形状 }
```

下面这个表格 (表 8.3, 由 F. Mittelbach 和 R. Schöpf 提供) 列出了根据 `\fontfamily`, `\fontseries` 和 `\fontshape` 属性对计算机现代字符集的分类。有相当多的属性组合并不对应任一 CM 字体, 这看起来好像是 NFSS 系统的一种缺陷, 但我们要知道, 这一设计是为将来考虑的。它也可以应用于正变得越来越普及的 PostScript 字体, 以开发其完整用途。

形式上是可以任意设置属性的组合的; 然而, 可能不存在一种字体对应于所有选择的属性。如果出现这种情况, 那么当调用 `\selectfont` 时,  $\text{\LaTeX}$  会给出一条警告信息, 告诉你它用什么字体取代所需字体。在 `\fontsize` 命令中的字体尺寸属性通常可以取 5, 6, 7, 8, 9, 10, 10.95, 12, 14.4, 17.28, 20.74, 但也可以加上其它值。第二个参数值, 即基线间距, 可以取任何值, 因为它并不是字体固有的性质。

利用 `\begin{document}` 命令,  $\text{\LaTeX}$  给五种属性设置当前特定默认值。

表 8.3: 计算机现代字体的属性

序列	形状	外部字体名称示例
计算机现代罗马字体 —(\fontfamily{cmr})		
m	n, it, sl, sc, u	cmr10, cmti10, cmsl10, cmcsc10, cmu10
bx	n, it, sl	cmbx10, cmbxti10, cmbxsl10
b	n	cmb10
计算机现代 Sans Serif 字体 —(\fontfamily{cmss})		
m	n, sl	cmss10, cmssi10
bx	n	cmssbx10
sbc	n	cmssdc10
计算机现代打字机字体 —(\fontfamily{cmtt})		
m	n, it, sl, sc	cmtt10, cmitt10, cmsl10, cmtcsc10

这通常就是标准编码 OT1，族 **cmr**，中间序列 **m**，正常形状 **n** 和选择的基本尺寸。用户可以在导言中改变这些值，或者用特殊选项把它们设置成不同的值，例如当已经选定了一种 PostScript 字体的时候。

### §8.5.2 简化的字体选择

属性命令 `\fontencoding`, `\fontfamily`, `\fontseries`, `\fontshape` 和 `\fontsize`，连同命令 `\selectfont`，都是新字体选择框架中的基本工具。用户并不需要直接使用这些命令，而可以利用列在 4.1.2 节和 4.1.3 节的高级声明。事实上，类似于 `\itshape` 这样的字体声明就是定义为

```
\fontshape{it}\selectfont。
```

用来选择字体尺寸的高级命令有：

```

\tiny   (5pt)   \normalsize (10pt)   \LARGE (17.28pt)
\scriptsize (7pt)   \large (12pt)   \huge (20.74pt)
\footnotesize (8pt)   \Large (14.4pt)   \Huge (24.88pt)
\small (9pt)
```

当在 `\documentclass` 命令中选择 10pt（默认值）做为基本尺寸选项时，上面命令相应的尺寸就是列在括号内的数值；当选择了 11pt 或 12pt 时，这些尺寸就会相应的放大。

族声明及对应的标准族属性值为

```
\rmfamily (cmr)   \sffamily (cmss)   \ttfamily (cmtt)
```

这分别相应于计算机现代族中的罗马、Sans Serif 和打字机字体 (E.2 节)。

序列声明及其初始值为:

```
\mdseries (m)      \bfseries (bx)
```

这就是说在标准中只提供了中间和黑松。

最后, 形状声明和相应属性值为:

```
\upshape (n)      \itshape (it)
\slshape (sl)      \scshape (sc)
```

这可以选择直立、slanted、斜体和小体大写字母。

注意对编码并没有高级声明。这是因为通常并不需要在文档中改变编码。

当要用 Cyrillic 字体 (编码 OT2) 时就是一个例外, 这时可以进行如下定义:

```
\newcommand{\cyr}{\fontencoding{OT2}\selectfont}
\newcommand{\lat}{\fontencoding{OT1}\selectfont}
```

这样就可以方便地来回切换了。

利用 \normalfont 命令, 可以随时把族、形状和序列属性值重设为标准值, 这也激活了当前尺寸的那种字体。

对于上面每种字体属性声明, 也存在着一个相应的字体命令 (4.1.4 节), 用以设置其参数值的字体。因此 \textit{text} 就与 {\itshape text} 差不多一样, 唯一的差别就在于命令中自动包含倾斜校正。这些命令的完全清单如下:

```
族:      \textrm  \textsf      \texttt
序列:    \textmd  \textbf
形状:    \textup  \textit      \textsl  \textsc
其它:    \emph    \textnormal
```

在 4.1.1 节中描述了 \emph 命令; \textnormal 把其参数值设为 \normalfont。

### §8.5.3 默认属性值

在前面一节中的字体属性声明并没有显式地设置它们的值, 而是用某种默认的命令来进行。因此 \itshape 的真正定义是:

```
\fontshape{\itdefault}\selectfont
```

可用的默认命令有:

```
族:      \rmdefault  \sfdefault  \ttdefault
序列:    \mddefault  \bfdefault
形状:    \updefault  \itdefault  \sldefault  \scdefault
```

当调用了 \normalfont 命令时, 需要定义标准属性值。它们是由如下四个默认值组成的:

```
\encodingdefault  \familydefault  \seriesdefault|
\shapedefault
```

所有这一切都使得我们觉得好像高级命令与特定字体之间的联系是非常复杂的。实际上，它确实提供了足够的弹性和模块化结构。作者只需要知道这三个族、两个序列和四种形状是可以用的，而不用关心它们实际是什么。程序设计者用低级命令定义它们的默认值。

在 212 页上有一个例子，通过重定义三个族的默认值，来说明了如何利用 PostScript 字体取代所有四种标准字体。这样重定义比改变包括 `\normalfont` 在内的字体声明要简单得多。这些定义实际上要远比这里提到的复杂，然而默认命令确实就如这里所指出的那样简单。

#### §8.5.4 定义字体命令

有很多命令可以用来定义新的字体声明和命令。这些命令主要是为 L<sup>A</sup>T<sub>E</sub>X 宏包开发者提供的，但也可以用在普通文档中。

`\DeclareFixedFont{\命令}{编码}{族}{序列}{形状}{尺寸}`

把 `\命令` 定义为一个选择具有指定属性字体的声明。所有属性都是严格固定的。这与 `\newfont` 基本等价，除了这里的字体是由属性而不是由名称确定的。

`\DeclareTextFontCommand{\命令}{字体指定}`

定义 `\命令` 为一个字体命令，它按照 `字体指定` 设置其参数值。在内部就是用这条命令来定义所有类似于 `\textbf` 的命令，而定义 `\textbf` 时 `字体指定` 为 `\bfseries`。

`\DeclareOldFontCommand{\命令}{文本指定}{数学指定}`

定义 `\命令` 定义了按照 L<sup>A</sup>T<sub>E</sub>X 2.09 方式可以用在数学模式中的字体声明。注意它是一个声明，不是一条命令。这对于定义与原来版本兼容命令是相当有用的，但应尽量避免使用。例如，`\it` 的定义为

```
\DeclareOldFontCommand{\it}{\normalfont\itshape}{\mathit}
```

#### §8.5.5 数学字母表

已被激活的用于文本处理的字体对数学模式中的字符及其字体并没有作用，因为此时用的是特殊的数学符号字体。如果想使一个公式显示为黑体，那么就必须用 `\boldmath` 命令 (5.4.9 节)，该命令的作用持续到调用相反命令 `\unboldmath` 为止。这些声明都必须在数学模式外面给出。

在 NFSS 下也可以通过同样方式应用这些声明。然而，内部的数学字体选择命令为

`\mathversion{变体名称}`

这里的参数值 `变体名称` 通常就取 `normal` 和 `bold`。`\boldmath` 和 `\unboldmath` 声明就是用这一命令定义的。现在有人已计划提供一些特殊的宏包文件，从而可以使用其它的数学符号集合。

另一方面，下列数学公式中的字体命令也可以在数学模式内部调用，把字母设置成特定的字体（5.4.2 节）：

```
\mathrm \mathcal \mathnormal \mathbf \mathsf \mathit \mathtt
```

这些都是对参数值有作用的命令，而不是声明，这一点与 L<sup>A</sup>T<sub>E</sub>X2.09 中的不同。

新数学字体字母表也可以由用户定义。例如，为了定义 slanted 数学字体 `\mathsl`，可以用

```
\DeclareMathAlphabet{\mathsl}{OT1}{cmr}{m}{sl}
```

这就意味着新数学字体命令 `\mathsl` 选择族为 `cmr`，权为 `m` 和形状为 `sl` 的字体，在通常字体定义中，这就是适当尺寸的 `cmsl` 字体。然而，在所有数学变体下，都可以选择这种字体，尽管当 `\mathversion{bold}` 有效时，选择的是黑体字体时它可能更恰当。为此可以在 `\mathsl` 定义中加入该选项：

```
\SetMathAlphabet{\mathsl}{bold}{OT1}{cmr}{bx}{sl}
```

这样就定义 `\mathsl` 为期望的只适用于 `bold` 变体，权为 `bx` 的字体。对于普通的字体定义，这就是当前尺寸的 `cmbxsl`。

可以用下面的命令创建新的数学变体：

```
\DeclareMathVersion{变体名称}
```

属于它的字体是由对每个数学字母表或符号字体调用 `\Set...` 命令确定的。

### §8.5.6 数学符号字体

数学符号的定义方式必须与文本字符完全不同：它们拥有一个命令名（如 `\alpha`），可能来自于不同的字体，根据不同的类型有不同的行为，可以显示为不同的尺寸。在 L<sup>A</sup>T<sub>E</sub>X2.09 中，符号名称固定为计算机现代数学字体，但 NFSS 为其它（或者代替）符号字体提供了一个相当大的弹性。

用下面这条命令声明一个符号字体名称：

```
2s\DeclareSymbolFont{符号字体名称}{编码}{族}{序列}{形状}
```

这样就把符号字体名称与给定的属性集联系起来。这一名称不是一条命令，而是一个内部用来定义符号的标识。所选定的字体对所有变体都有效，除非在其它变体中指定了同名的不同字体。

```
2s\SetSymbolFont{符号字体名称}{变体}{编码}{族}
{序列}{形状}
```

可以用来重定义一种变体下的符号字体名称。

标准 L<sup>A</sup>T<sub>E</sub>X 安装中有如下声明

```
\DeclareSymbolFont{operators}{OT1}{cmr}{m}{n}
\DeclareSymbolFont{letters}{OML}{cmm}{m}{it}
\DeclareSymbolFont{symbols}{OMS}{cmsy}{m}{n}
\DelcareSymbolFont{largesymbols}{OMX}{cmex}{m}{n}
```

这一串声明是深建在  $\text{\LaTeX}$  内部的，这也是它们之所以重要的原因。

一旦已定义了符号字体名称，可以用它来构造数学字母表和各种不同类型的符号。

$\text{\textbackslash}$ `DeclareSymbolFontAlphabet`{*数学字母表* }{*数学字体名称* }

把 *数学字母表* 定义成基于内部名称 *数学字体名称* 的数学字母表。如果相应于这个数学字母表的字体合适属性已经存在，那么这条命令就要优于命令 `\DeclareMathAlphabet`。

定义符号的主要命令是

$\text{\textbackslash}$ `DeclareMathSymbol`{*符号* }{*类型* }{*符号字体名称* }{*位置* }

这使得 *符号* 显示在字体 *符号字体名称* 中处于给定 *位置* 的符号。这里 *位置* 是一个数，可以用十进制（如 10），八进制（如 '12）或十六进制（如 "0A）表示。类型 定义符号的功能，它可以取如下一个值：

<code>\mathord</code>	普通符号
<code>\mathop</code>	大运算符，如 $\sum$
<code>\mathbin</code>	二元运算符，如 $\times$
<code>\mathrel</code>	关系运算符，如 $\geq$
<code>\mathopen</code>	左括号，如 {
<code>\mathclose</code>	右括号，如 }
<code>\mathpunct</code>	标点符号
<code>\mathalpha</code>	字母表字符

数学字母表命令只作用在类型为 `\mathalpha` 的符号上；对于其它类型，在给定的数学变体里，在所有数学字母表中都生成同样的符号。

上面的数学字体声明中并没有指定尺寸。这是因为通常有四种尺寸可以使用，具体与数学样式有关，可看 5.5.2 节的解释。然而，这些尺寸必须在某个地方进行指定。这是用如下命令完成的：

$\text{\textbackslash}$ `DeclareMathSizes`{*正文* }{*数学文本* }{*上下标* }{*双重上下标* }

这里的四个参数值都是数值，给出尺寸的点数。当正常文字体的尺寸是 *正文*pt 时，`\textstyle` 就会是 *数学文本* 尺寸，`\scriptstyle` 是 *上下标* 尺寸，而 `\scriptscriptstyle` 是 *双重上下标* 尺寸。例如，

`\DeclareMathSizes{10}{10}{7}{5}`

所有的 `\Declare...` 和 `\Set...` 命令都只能在导言中调用。

### §8.5.7 处理属性值

在程序设计中，有时要利用属性的当前值，而并不知道它们的值是多少。这些值实际上保存在如下内部命令中的：

<code>\f@encoding</code>	<code>\f@shape</code>	<code>\tf@size</code>
<code>\f@family</code>	<code>\f@size</code>	<code>\sf@size</code>

`\f@series      \f@baselineskip    \ssf@size`

永远不能直接改变这些命令的值。然而，可以测试它们，以确定它们是否具有某个值。由于其名称中都包含字符 `@`，因此它们只能用在类或宏包文件中，而不能直接用在主文档文件中（附录 C）。

### §8.5.8 定义 NFSS 中的字体

在 NFSS 中，如果需要在文档中指定字体，那么就先给出所需的属性，然后调用 `\selectfont`。这样确定的字体属性集合是如何与附录 E 中所讲的特定外部字体名称联系起来的呢？这是通过字体定义命令做到的，它通常保存在扩展名为 `.def` 和 `.fd` 的文件中。

首先利用声明

`\DeclareFontEncoding{ 编码 }{ 正文集合 }{ 数学集合 }`

建立一个叫 `编码` 的新编码属性；无论何时选择这种编码中的一个字体，就会执行 `正文集合`，以重定义重音命令或其它与编码有关的事情；同样地，对于这种编码中的每个数学字母表都会执行调用 `数学集合`。也可以如下定义默认的 `正文集合` 和 `数学集合`：

`\DeclareFontEncodingDefaults{ 正文集合 }{ 数学集合 }`

可以用这条命令声明一般的文本和数学模式设置，而更具体的，在后面执行的设置是位于 `\DeclareFontEncoding` 中的。

如果对应于指定的属性，不存在任何字体，

`\DeclareFontSubstitution{ 编码 }{ 族 }{ 序列 }{ 形状 }`

声明出要被取代的属性值；取代是按照从 `形状`，`序列`，然后 `族` 的顺序进行的；`编码` 从不会被取代。如果这样还找不到对应字体，那么

`\DeclareErrorFont{ 编码 }{ 族 }{ 序列 }{ 形状 }{ 尺寸 }`

就确定出最终迫不得已时使用的字体。

指定编码框架中的一个新族是用如下命令建立的：

`\DeclareFontFamily{ 编码 }{ 族 }{ 选项 }`

这里的 `option` 是一组命令，每当选择该族和编码中的一种字体时就会被执行。

把字体属性与外部字体名称关联的主要字体定义声明是：

`\DeclareFontShape{code}{family}{series}{shape}`  
`{font_def}{option}`

这里的 `选项` 是另外的命令，每当选择其中一种字体时就会执行它。

字体指定 包含一系列尺寸 / 字体联系，每一个联系都是由一个尺寸对、一个函数、一个可省参数和一个字体参数值组成。例如，

`\DeclareFontShape{OT1}{cmr}{m}{n}`  
`{ <5> <6> <7> <8> <9> <10> <12> gen * cmr`

```

<10.95> cmr10
<14.4> cmr12
<17.28> <20.74> <24.88> cmr17}{\}

```

就说明 **cmr** 族的中间序列、正常形状成员用外部字体 **cmr5** ... **cmr12** 表示 5–12pt 的尺寸，用 **cmr10** 放大到 10.95 以接近 11pt 的尺寸，等等。如果指定的尺寸不存在，就会用特定限度内最接近的尺寸。

尺寸部分由一个或多个放在尖括号内的表示尺寸 (pt) 的数字组成。括号内也可以包含范围，如 **<-10>** 就表示所有小于 10pt 的尺寸，而 **<10-14>** 表示从 10pt 到小于 14pt 的尺寸，而 **<24->** 表示 24pt 或更高。可能的函数有：

**(empty)** (上面例子中的第二、三、四行) 上载指定的字体，并放缩到要求的点数尺寸；如果在字体名称前面有位于中括号内的可省参数值，那它就是一个额外的放缩因子：

**<11> [.95] cmr10** 就会上载 **cmr10** 并放缩到 11pt 的 95% ；

**gen \*** (上面例子中的第一行) 把点数尺寸加到字体参数值后，生成字体名称；

**<12> gen \* cmr** 上载 **cmr12** ；

**sub \*** 用不同的字体代替，这种字体的属性在形式为 族/序列/形状 的字体参数值中给出；

**<-> sub \* cmtt/m/n** 当没有字体具有要求的属性时，最好用这种字体；会在监视器和抄本文件中给出一条信息；

**subf \*** 类似于空函数，但是如果上载了一种显式取代字体时会给出一条警告信息；

**fixed \*** 以正常尺寸上载指定字体，忽略尺寸部分；如果给出了可省参数值，那这个参数值就是字体要放缩到的点数尺寸，如

**<10> fixed \* [11] cmr12** 当要求的是 10pt 时，就在 11pt 尺寸下上载 **cmr12** 。

上面所有函数都可以前缀一个 **s**(表示无声)，以禁止显示在屏幕上的信息。

因此 **sub \*** 的无声形式为 **ssub \***，无声的空函数是 **s \***。

现在给出另一个例子，考虑黑斜打字机字体的定义，因为在计算机现代字体集中没有这种字体：

```

\DeclareFontShape{OT1}{cmtt}{bx}{it}{\}
<-> ssub * cmtt/m/it }{\}

```

这会把所有尺寸 (**<->**) 的字体（无声地）用中间斜体打字机属性取代。这就是那些由 **\DeclareFontShape** 命令确定的相关属性的字体。

字体定义命令也可以在宏包文件中调用，甚至在文档中也可以使用。然而，通常的过程是先把每条 **\DeclareFontEncoding** 命令存贮在一个名为 **编码enc.def** (例如 **OT1enc.def** 对应于 **OT1** 编码) 的文件中，然后把命令 **\DeclareFontFamily** 和 **\DeclareFontShape** 放在一个文件中，其名称由编



码加上族标识, 后缀 `.fd` 组成。例如, 编码 `OT1` 和族 `cmr` 的形状定义可以在 `OT1cmr.fd` 文件中找到。当选择的编码和族组合并没有定义,  $\text{\LaTeX}$  就会尝试找到相应的 `.fd` 文件做为输入。因此并不需要显式地输入这个文件, 因为需要时可以自动调入。

然而, 事先声明编码是重要的。如果在当前格式中并不知道编码, 就必须调用 `\DeclareFontEncoding`, 方法是或者显式地调用, 或者上载 编码 `enc.def` 文件。例如, 做到这一点的方法就是调用已存在的宏包 `fontenc`:

```
\usepackage[OT2,T1]{fontenc}
```

这里所期望的编码列在中括号做为选项, 这组选项就是当前的编码。

普通用户从来不必担心这些问题。然而,  $\text{\LaTeX}$  程序开发者将会发现这些事情是相当容易的。例如, 在 NFSS 中安装 PostScript 字体是相当平凡的。一些常用的 PostScript 字体已经在它们自己的 `.fd` 文件做为单独的族定义好了; 例如 `OT1ptm.fd` 就把 PostScript 的新罗马字体与一个名为 `ptm` 的族关联起来。激活这些字体的宏包是在名为 `times.sty` 的文件中, 其中主要包含如下几行:

```
\renewcommand{\rmdefault}{ptm}
\renewcommand{\sfdefault}{phv}
\renewcommand{\ttdefault}{pcr}
```

这就使得新罗马 `ptm` 成为默认罗马字体族, 用 `\rmfamily` 命令调用, 而 Helvetica `phv` 成为默认的 sans serif 字体族 (用 `\sffamily` 调用), Courier `pcr` 为默认的打字机字体族 (用 `\ttfamily` 激活)。

另外两个关于 NFSS 使用的例子是华盛顿大学的 Cyrillic 字体 (E.7 节) 和 Cork 编码中的扩展 DC 字体。这两个字体都可以在文档中只要选择另一个编码 `OT2` (对应于 Cyrillic), `T1` (对应于 DC 字体) 就可以激活, (这些编码必须首先进行声明, 例如利用上面说明的 `fontenc` 宏包。)

### §8.5.9 编码命令

在  $\text{\LaTeX}$  中, 必须用命令来得到特殊的字符和重音, 例如 `\O` 显示出 Scandinavian 字母  $\text{\O}$ 。这个字符在字体表格中的位置有编码有关 (在 `OT1` 中是第 31 个字符, 而在 `T1` 中是第 216 个字符), 因此当编码改变了时, 需要重定义所有这样的符号命令。这可以借助于某种编码命令来完成, 这些命令通常与 `\DeclareFontEncoding` 命令一起位于 编码 `enc.def` 文件中。

为了定义一个在不同编码下功能不同的命令, 可以用:

```
2ε \ProvideTextCommand{\命令}{编码}[参数个数]
      [可省参数]{定义}
2ε \DeclareTextCommand{\命令}{编码}[参数个数]
      [可省参数]{定义}
```

其行为同 `\providecommand` 和 `\newcommand` 命令一样，只是只有当 编码 被激活时 `\命令` 具有这里给定的 定义。因此对每种编码 `\命令` 就具有不同的定义。

$\boxed{2\epsilon}$  `\DeclareTextSymbol{\命令}{编码}{位置}`

定义 `\命令` 为当 编码 被激活时字体中给定 位置 的字符。

$\boxed{2\epsilon}$  `\DeclareTextAccent{\命令}{编码}{位置}`

定义 `\命令` 为一个重音命令，当 编码 被激活时使用位于给定 位置 处的字符作为重音符号。

$\boxed{2\epsilon}$  `\DeclareTextComposite{\命令}{编码}{字母}{位置}`

$\boxed{2\epsilon}$  `\DeclareTextCompositeCommand{\命令}{编码}{字母}{定义}`

定义命令 `\命令` 及后接的单个 字母 的行为是显示在字体给定 位置 处的字符或者执行 定义。在 T1 编码中这些声明是相当有用的，这时许多重音字母就是单独一个符号。因此 `\'e` 在字母 e 上放尖重音（字符 19），而在 OT1 中它显示的是位置 233 处的字符。这个行为是如下得到的：

`\DeclareTextAccent{\'}{OT1}{19}`

`\DeclareTextComposite{\'}{T1}{e}{233}`

相应于给定编码的这条命令必须已经有定义，方法是用 `\DeclareTextAccent` 或者 `\DeclareTextCommand`；对后一种情形，必须把它定义成具有单个参数值的命令。

上面所有定义命令都生成相应于指定编码中的新命令。如果这里定义的命令在其它编码中调用，会给出错误信息。可以为未指定的编码给出默认定义：

$\boxed{2\epsilon}$  `\DeclareTextCommandDefault{\命令}[参数个数][可省参数]{定义}`

$\boxed{2\epsilon}$  `\ProvideTextCommandDefault{\命令}[参数个数][可省参数]{定义}`

$\boxed{2\epsilon}$  `\DeclareTextAccentDefault{\命令}{编码}`

$\boxed{2\epsilon}$  `\DeclareTextSymbolDefault{\命令}{编码}`

这里前两条命令创建适用于所有未指定编码的默认定义，而后两条命令声明把哪种编码取为默认值。

注意：`\DeclareTextCompositeCommand`、`\Provide...` 和默认命令是于 1994 年 12 月 1 日加到  $\text{\LaTeX} 2\epsilon$  中的。因此任何使用这些命令的文件应包含 `\NeedsTeXFormat{LaTeX2e}[1994/12/01]`。

## §8.6 输入代码 $\boxed{2\epsilon}$

在不同的计算机系统上有可能直接向输入文件中输入一些 2.5.5-2.5.7 节的特殊符号，而且在用文本编辑器准备  $\text{\LaTeX}$  文件时这些符号会显示在屏幕上。有些  $\text{\TeX}$  实现版本确实可以把这些特殊符号转化成对应的命令。不幸的是这些特殊字符并没有标准代码，因此这样的文件是与系统有关的。例如，

特殊字母  $\text{\AE}$  是 ISO-Latin1 代码方案的第 198 位字符, 而在 IBM PC 扩展系统中却是第 146 位。

`inputenc` 软件包可以解决这个与系统有关的问题。其调用方式为

```
\usepackage[ 编码 ]{inputenc}
```

这样就依照代码方案 `编码` 来定义第 128–255 位输入字符的代码。`编码` 可能取的值为 `ascii`(没有扩展字符), `latin1`, `latin2` (ISO-Latin1 和 2 代码), `cp437`, `cp850` (IBM 代码页中第 437 页和 850 页) 以及 `applemac` (Apple Macintosh 代码)。

在文件 `编码.def` 中用如下命令定义了扩展字符:

```
\DeclareInputText{ 位置 }{ 文本 } 或者
```

```
\DeclareInputMath{ 位置 }{ 数学 }
```

这样就把 `文本` 或 `数学` 的代码指定给给定 `位置` 的字符。例如, `latin1.def` 中包含

```
\DeclareInputText{198}{\AE}
```

这样就把输入的第 198 位字符翻译成  $\text{\LaTeX}$  命令 `\AE`。

这也就是说为了得到 the rôle of the æther, 可以直接输入 `the rôle of the æther` (不必用 `the r\^o\le of the {ae}ther`), 这样即使同样的输入, 在不同的计算机上可能结果完全不一样。

## §8.7 特殊符号的替代 2ε

浏览一下 CM (第 368 页) 和 DC 的字体布局方案, 那你就会发现有很多符号, 是不能直接用通常的命令输入的。例如,  $\text{\textit{i}}$  和  $\text{\textit{j}}$  认为是输入文件 `!'` 和 `?'` 的连写, 其它特殊符号类似。有一些符号对应的命令只能用在数学模式中。

利用连写生成符号的一个问题是有些特殊字体可能并不支持, 即使在相应的位置上有这个符号。

任何符号都可以通过 `\symbol` 命令直接使用, 这条命令要求布局位置做为参数值, 但实际生成的字符与字体编码有关。一些 `\text...` 命令可以用来直接显示所有这些符号, 而与当前编码无关。

命令	符号	替代
连写		
<code>\textemdash</code>	—	---
<code>\textendash</code>	–	--
<code>\textexclamdown</code>	¡	!‘
<code>\textquestiondown</code>	¿	?‘
<code>\textquotedblleft</code>	“	‘‘
<code>\textquotedblright</code>	”	’’
<code>\textquoteleft</code>	‘	‘
<code>\textquoteright</code>	’	’
数学符号		
<code>\textbullet</code>	•	<code>\$_bullet\$</code>
<code>\textperiodcentered</code>	·	<code>\$_cdot\$</code>
杂类		
<code>\textvisiblespace</code>	□	<code>\verb*+ +</code>

另外还有:

`\textcompwordmark` 表示分开连写的不可见字符

`\textcircled{字符}` 用圆把字符包起来, 如 ⊗。

`\textsuperscript{字符}` 生成当前文本 (不是数学) 字体中的上标。

## §8.8 其它标准文件

基本的 L<sup>A</sup>T<sub>E</sub>X 格式包含了需要生成通常文档所需的绝大多数功能。然而还存在许多其它功能, 在某些应用中是不可缺少的, 可以用 `\usepackage` 命令 (3.1.2 节) 把它们包含进来。各个方面的用户编写个上百个这样的宏包, 可能通过计算机网格得到这些宏包 (D.5 节); 还有一些宏包是由 L<sup>A</sup>T<sub>E</sub>X3 项目组的成员开发的, 与标准安装版本一起发行; 最后要指出, 有些宏包就是标准安装版本的一部分。

### §8.8.1 特殊文档

在标准安装版本中有许多特殊的‘文档’, 这些文件的后缀为 `.tex`。它们有:

`small.tex`<sup>2.09</sup> L<sup>A</sup>T<sub>E</sub>X 2.09 中一个简短的示例文档;

`sample.tex`<sup>2.09</sup> L<sup>A</sup>T<sub>E</sub>X 2.09 中一个较长的示例文档;

`small2e.tex`<sup>2ε</sup> L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 中一个简短的示例文档;

`sample2e.tex`<sup>2ε</sup> L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 中一个较长的示例文档;

`labl1st.tex` (8.3.1 节) 显示出所有的交叉引用表的翻译及对应页码; 它会交

交互式地询问要列出的文档名称;

`idx.tex` (8.3.3 节) 显示出文档中所有的 `\index` 项, 以两列方式同时显示出页码; 它也要交互式地询问要列出的文档名称;

`testpage.tex` 测试文本在页面上的位置; 这也是一个打印机驱动器的检测, 以确保页边界正确, 放缩显示恰当; 在  $\text{\LaTeX} 2.09$  中, 这个文档只适用于美国信纸, 而在  $\text{\LaTeX} 2_{\epsilon}$  中, 它会交互式地询问所希望的纸张尺寸;

`nfssfont.tex`<sup>2 $\epsilon$</sup>  这个文档显示字体表格, 示例文本以及其它各种检测; 它会交互式询问字体的名称; `\help` 命令会显示出所有可用的命令;

`docstrip.tex`<sup>2 $\epsilon$</sup>  (D.3.1 节) 如其说这是一个文档, 不如说是一个程序; 它是从文档源文件中解开得到操作文件的基本工具; 它不但去掉注释, 而且会根据不同的选项加上替代编码。

### §8.8.2 其它类<sup>2 $\epsilon$</sup>

除了我们已经讨论的标准类 (`book`, `report`, `article`, `letter`) 外, 在标准安装版本还有如下类:

`proc` 生成会议学报的照像制版拷贝; 它是 `article` 类的两列模式的变体; 它包含额外的一条命令 `\copyrightspace`, 用以在第一列的底部留下显示版权信息的空间; (在  $\text{\LaTeX} 2.09$  中, 这是 `article` 样式的一个选项, 在  $\text{\LaTeX} 2_{\epsilon}$  中也可以这样用, 但这只是为了兼容的需要);

`slides` (8.10 节) 相应于  $\text{\LaTeX} 2.09$  中的 `SL $\text{\TeX}$` , 用来生成演讲材料;

`ltxdoc` (D.3.2 节) 生成类和宏包文件的文档。

### §8.8.3 标准宏包

那些成为  $\text{\LaTeX}$  标准安装版本一部分的宏包, 要考虑得相当广泛, 这样我们就可以期望总是可以用它。而其它的宏包, 即使是  $\text{\LaTeX} 3$  项目组的, 也都只是供选择应用的。

有些标准宏包我们在其它地方已做了描述。标准宏包有:

`alltt` 给出了 `alltt` 环境, 它非常类似于 `verbatim` 环境, 只是 `\{` 和 `\}` 的行为与通常的一样; 即在文本中的命令要被执行, 而不是照字样显示;

`bezier`<sup>2.09</sup> (6.4.9 节) 提供了在 `picture` 环境中画二次曲线的能力; 这个功能现在内植于  $\text{\LaTeX} 2_{\epsilon}$  中, 只是为了兼容原来的文档, 而提供了一个空文件;

`doc`<sup>2 $\epsilon$</sup>  (D.3.2 节) 提供了建立  $\text{\LaTeX}$  宏包文档的特殊功能;

`exscale`<sup>2 $\epsilon$</sup>  允许以不同尺寸显示的数学符号也根据在 `\documentclass` 命令中的字体尺寸选项进行放缩; 通常这些符号的尺寸是固定的, 与文本的基本尺寸无关;

`fontenc`<sup>2 $\epsilon$</sup>  (8.5.8 节, 第 212 页) 通过上载 `.def` 文件来声明字体编码; 编码

的名称列在选项中;

`graphpap`<sup>[2ε]</sup> (6.5.6 节) 给出了命令 `\graphpaper`, 以生成 `picture` 环境中的坐标网格;

`ifthen` (7.3.5 节) 允许文本或命令的定义与各种条件的状态有关; 我们可以测试 Boolean 值开关的状态, 计数器或长度的值, 或者某一命令的定义文本;

`inputenc`<sup>[2ε]</sup> (8.6 节) 允许在制作 L<sup>A</sup>T<sub>E</sub>X 文档时特殊字符的输入采用的是与系统无关的编码方案;

`latexsym`<sup>[2ε]</sup> 用 `lasy` 字体 (E.5.1 节) 上载 11 个特殊的 L<sup>A</sup>T<sub>E</sub>X 符号, 这些符号现在并不是内植于 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 中; 在兼容模式中这个文件是自动读入的;

`makeidx` (8.4 节) 给出了当用 `MakeIndex` 程序生成关键词索引时可能用到的命令;

`newlfont`<sup>[2ε]</sup> 重定义了两字母声明, 使得它们的行为与第一版的 NFSS 中的一样; 这些字体声明只改变一个属性; 即 `\bf` 类似于 `\bfseries`, 其余类似;

`oldlfont`<sup>[2ε]</sup> 定义两字母 (如 `\bf`) 声明的行为同在 L<sup>A</sup>T<sub>E</sub>X 2.09 中文本和数学模式中的一样; 这些声明现在并不内植于 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 中, 但所有的标准类中都有其它义; 保留这个宏包只是处于兼容性的考虑;

`pict2e`<sup>[2ε]</sup> (6.5.6 节) 通过去掉直线长度、粗细、斜角以及圆的半径等方面的限制, 来增强某些 `picture` 环境命令的功能; 它利用了与驱动器有关的功能;

`showidx` (8.3.3 节) 使得 `\index` 项显示为边注;

`shortvrb`<sup>[2ε]</sup> 提供了 `\MakeShortVerb` 和 `\DeleteShortVerb`, 这两条命令用来标记 (和删掉) 某一特定字符的功能与 `\verb` 功能, 用以显示源文本。这个字符前缀反斜杠做为命令的参数。因此如果进行了如下调用:

```
\MakeShortVerb{\|}
```

那么 `| \cmd{} |` 等价于 `\verb| \cmd{} |`, 结果都是 `\cmd{};`

`syntonly`<sup>[2ε]</sup> 提供了命令 `\syntonly`, 在导言中调用它, 不把结果输入到 `.dvi` 文件中, 但会给出错误和警告信息; 这样会比正常的情形快四倍;

当并不急于得到输出时, 这种方法可以用来检验处理过程;

`Tienc`<sup>[2ε]</sup> 把 DC 字体编码设为标准;

`tracefmt`<sup>[2ε]</sup> 是一个检验 NFSS 字体的对话式工具; 在 `\usepackage` 命令中可以用下面这些选项控制其行为:

`errorshow` 不在屏幕上显示所有的警告和提示信息, 但会送到抄本文件中; 在屏幕上只显示错误信息;

`warningshow` 在屏幕上显示警告和错误信息; 这种形式与完全没有用这个宏包很相像;

**infoshow** (默认选项) 在显示器上显示字体选择信息, 通常这些信息只是写到抄本文件中;

**debugshow** 在每次改变字体, 都会向抄本文件中写入相当多的信息; 这样会得到一个很大的抄本文件;

**pausing** 把所有的警告信息都转化为错误信息, 这样会使得处理暂时中止, 等待用户的反馈;

**loading** 显示对外部字体的上载。

#### §8.8.4 附属软件

在可以找到 L<sup>A</sup>T<sub>E</sub>X 的网络服务器 (D.5 节) 上有很多平行的目录, 其中包含相当多的独立于标准安装版本之外的附属软件。这些软件都属于半正式状态。稍后在附录 D 中会描述其中一些软件对标准 L<sup>A</sup>T<sub>E</sub>X 的功能扩展。

**amslatex** 是一组由美国数学学会提供的用于高级数学排版的宏包。

**babel** 提供了 L<sup>A</sup>T<sub>E</sub>X 中的多语言支持, 在 D.1.3 中对它进行了描述。

**graphics** 使得可以利用一些宏包, 来进行一些图形处理: 插入、旋转和处理外部图形和图示, 以及彩色。

**mfnfss** 提供在 NFSS 中安装其它 METAFONT (位图) 字体的宏包 (见下面的例子)。

**psnfss** 提供在 NFSS 中安装特定 PostScript 类型 1 字体的软件包, 见 D.2.1 中的说明。

**tool** 是一组由 L<sup>A</sup>T<sub>E</sub>X3 项目组成员编写的实用宏包, 在 D.3 节中有描述。

上面宏包中大多数都有完整的文档, 在 D.3.2 中描述了如何利用文档化源代码的技术。把源文件在 L<sup>A</sup>T<sub>E</sub>X 中进行处理, 以得到手册和 / 或代码的详细解释, 而运行安装文件可以从源文件中得到各种文件 (.sty, .fd 等等)。

#### §8.8.5 捐献的宏包

并不是只有 L<sup>A</sup>T<sub>E</sub>X3 项目组的成员在开发 L<sup>A</sup>T<sub>E</sub>X 类和宏包文件。在网络服务器 (D.5 节) 上有成千上万个已公开的个人宏包。其中大多数很有用的宏包在 *The L<sup>A</sup>T<sub>E</sub>X Companion* (Goossens 等著, 1994 年) 一书中有介绍。至于时间方面, 这些宏包大多数是为 L<sup>A</sup>T<sub>E</sub>X2.09 编写, 但在 L<sup>A</sup>T<sub>E</sub>X2<sub>ε</sub> 中功能也相当不错。当然我们希望作者们能很快把它们更新到新的版本。

可以编写自己的类和宏包的能力 (在附录 C 中详细介绍) 是 L<sup>A</sup>T<sub>E</sub>X 的巨大能量之一。与其期望初始发明者把文本处理中每件事都做得很好, 不如实际用户针立自己专业需要, 寻找方法, 并使得其它用户也可以应用自己的这一方法。

## §8.9 各种 L<sup>A</sup>T<sub>E</sub>X 文件

在 L<sup>A</sup>T<sub>E</sub>X 的处理过程中要用到各种各样的文件：有些是从中读取信息，而有些是新创建的，存贮供下一次运行使用的信息。这些文件的名称都是由两部分组成：

基本名. 扩展名

对于每个 L<sup>A</sup>T<sub>E</sub>X 文档，都有一个主文件，当调用 L<sup>A</sup>T<sub>E</sub>X 程序时只需用它的基本名。其扩展名通常为 `.tex`，而其它文件可以用 `\input` 或 `\include` 命令来读入。（如果它们有另外的扩展名，那就必须显式地给出了。）

在 L<sup>A</sup>T<sub>E</sub>X 运行期间创建的文件通常都具有与主文件相同的基本名，而只是扩展名不同，具体与文件的功能有关。如果主文件中包含 `\include` 命令，那么就会创建另外一个同被包含的 `.tex` 文件基本名一样的文件，扩展名为 `.aux`。

其中有些文件每次运行 L<sup>A</sup>T<sub>E</sub>X 时都会被创建，而其它的文件只有当调用了特定 L<sup>A</sup>T<sub>E</sub>X 命令时才会出现，如 `\tableofcontents` 或 `\makeindex` 命令。对于 `.aux` 文件以及那些通过特殊 L<sup>A</sup>T<sub>E</sub>X 命令生成的文件，都可以通过在导言中包含下面这条命令来抑制其生成：

`\nofiles`

如果文档频繁被订正和重处理，那么这条命令就相当有用，因为这时特殊文件中的信息还没有定案，因此不必管它。在 T<sub>E</sub>X 层次上创建的文件总是会生成的。

其它的扩展名描述的文件用来包含格式信息、其它指令（编码）或数据库。这些文件名称中的基本名并不与任何文档有关。这种类型的一个例子是 `article.cls` 文件定义了 `article` 类。

本节余下部分是一个列表，包含 L<sup>A</sup>T<sub>E</sub>X 文本扩展名和其在 L<sup>A</sup>T<sub>E</sub>X 处理过程中作用的一个简短说明：

**.aux** 这是由 L<sup>A</sup>T<sub>E</sub>X 生成的辅助文件，包含交叉引用信息以及其它生成目录表和其它列表所需的命令。除了每次用 `\include` 命令读入一个文件时会为每个文件创建一个 `.aux` 文件，也会为主文件创建（重建）一个 `.aux` 文件。

当用 L<sup>A</sup>T<sub>E</sub>X 第一次处理一个文档时，并没有 `.aux` 文件，因此交叉引用命令并没有作用。这时会向屏幕上显示如下信息：

No file 主文件名.aux

`.aux` 文件是由 `\begin{document}` 命令读入的。当其被读入时，它们的名称会以下面的形式显示在终端屏幕上：

(主文件名.aux)(从属文件名 1.aux) ...

每次运行 L<sup>A</sup>T<sub>E</sub>X 进行处理时都会生成新的 `.aux` 文件。相应于主文件的新



.aux 文件是从 `\begin{document}` 命令开始的,而那些相应于 `\include` 命令的文件是由相应的 `\include` 命令初始化的,当读完文件时被关闭。  
主 .aux 文件是由 `\end{document}` 命令关闭的。

可以在导言中给出命令 `\nofiles` 来不生成辅助性 .aux 文件。

- .bbl 这个文件不是由  $\text{\LaTeX}$  创建的,而是由程序  $\text{\BibTeX}$  生成的。它与主文件有相同的基本名。实际上  $\text{\BibTeX}$  只是从 .aux 文件中读取信息。 .bbl 文件是由 `\bibliography` 命令读入到第二次  $\text{\LaTeX}$  处理过程中的,从而生成参考文献列表。
- .bib 参考文献数据库的扩展名为 .bib 。  $\text{\BibTeX}$  读取这些文件以提取生成 .bbl 文件所需的信息。其基本名描述了数据库信息,而不必与任何文件相关。数据库的名称是在文本中用 `\bibliography` 命令包含进来的。
- .blg 这是来自于  $\text{\BibTeX}$  运行时的抄本文件。它与输入文件有相同的基本名。
- .bst 这是参考文献样式文件,它是  $\text{\BibTeX}$  的输入,用以确定参考文献的格式。 .bst 文件的名称在正文中是用 `\bibliographystyle` 命令包含进来的。
- .cfg <sup>2ε</sup> 有些类允许局部把纸张设置成其它尺寸,方法是把这些定义放到一个与类有相同基本名的 .cfg 文件中。 `ltxdoc` 类就是这种类型的类 (D.3.2 节)。
- .clo <sup>2ε</sup> 这是  $\text{\LaTeX 2}_\epsilon$  中的类选项文件,由可能不只适用于一个类的特定选项的编码组成。并没有特殊命令来输入它。
- .cls <sup>2ε</sup> 这是  $\text{\LaTeX 2}_\epsilon$  类文件,定义文档的全局格式。它是由 `\documentclass` 命令读入的。 .cls 文件也可以用 `\LoadClass` 命令来输入。
- .def <sup>2ε</sup> 各种类型的额外定义文件都具有 .def 的扩展名。例如 `T1enc.def` (定义 T1 编码)和 `latex209.def` (保持  $\text{\LaTeX 2}_\epsilon$  和  $\text{\LaTeX 2.09}$  中的兼容性)。
- .drv <sup>2ε</sup> 不用直接处理 .dtx 文件以得到档案,而是处理 .drv 驱动器文件。档案通常是用 `DocStrip` 从 .dtx 文件中提取的。而现在这种方法的好处在于,当为了满足不同的页面尺寸,或者其它要求时,用户可以按手册的指导或者利用代码的解释来编辑这些驱动程序。(在早先的版本中,这时得到档案文件的唯一方法。)
- .dtx <sup>2ε</sup> 在  $\text{\LaTeX}$  安装中有档案化的源文件。如果直接处理 .dtx 文件,就会显示出档案文件。这个文件是一个解释性的手册和 / 或对代码的详细解释。通过利用 `DocStrip` 处理 .dtx 文件可以得到类、宏包、.ltx 、.fd 或其它文件 (D.3.1 节)。
- .dvi 这是  $\text{\TeX}$  的输出文件,其由有格式的与输出打印机无关的被处理后的文本组成,因此称之为‘与设备无关’文件。每次运行  $\text{\TeX}$  处理文件时都会生成这个文件,不能用 `\nofiles` 禁止生成它。当这个文件被关闭时,会在屏幕上显示如下信息:

Output written on 主文件名.dvi(*n* pages, *m*bytes).

当有错误出现时, 没有向 .dvi 文件中写入任何信息, 这时会显示如下信息:

No pages of output

必须用打印机驱动程序进一步处理 .dvi 文件, 以把它转化为所用输出设置 (打印机) 上的特殊命令。

- .fd <sup>2ε</sup> 这个文件包含 NFSS 命令, 这些命令把外部字体名称与字体属性关联起来 (8.5.8 节), 这个文件被称为字体定义文件。其基本名由编码和族标识组成, 如 OT1cmr.fd。当第一次使用给定族的一个字体时, 就会自动地读入相应 .fd 文件 (如果存在的话)。
- .fdd <sup>2ε</sup> .fd 文件的档案化文件具有扩展名 .fdd, 它只是一种特殊类型的 .dtx 文件。为了得到档案文件, 可以处理它, 或者用 DocStrip 提取 .fd 文件。
- .fmt 这个扩展名属于一种 T<sub>E</sub>X 格式, 它是用特殊的程序 initex(1.2.1 节) 创建的。为了快速上载在格式文件中以紧致编码存贮基本指令。L<sup>A</sup>T<sub>E</sub>X 也就是 T<sub>E</sub>X 运行时使用格式 latex.fmt<sup>2ε</sup> 或者 lplain.fmt<sup>2.09</sup>。(Plain T<sub>E</sub>X 利用的格式文件是 plain.fmt。)
- .glo 只有当导言中有指令 \makeglossary 时, 才会生成这个文件。它具有与主文件相同的基本名, 整个文件是由源文本中的 \glossary 命令生成的 \glossaryentry 命令组成。如果在导言中有 \nofiles 命令, 那么即使用了 \makeglossary, 也不会生成这个文件。
- .gls 这个文件就是汇总中的 .ind 文件。它也是由 MakeIndex 程序生成的, 以 .glo 文件做为输入。这里的输入和输入文件扩展名都必须显式给出。不存在标准的 L<sup>A</sup>T<sub>E</sub>X 命令用来读取 .gls 文件。在 doc 宏包中用它来记录改变历史。
- .idx 只有当导言中有指令 \makeindex 时才会生成这个文件。它与主文件有相同的基本名, 只是由源文本中的 \index 命令生成的 \indexentry 命令组成。如果在导言有 \nofiles 命令, 那么即使用了 \makeindex 命令, 也不会生成这个文件。
- .ilg 这是运行 MakeIndex 时的抄本文件。与输入文件有相同的基本名。
- .ind 这个文件由 MakeIndex 程序生成, 以 .idx 文件为输入。它是由 makeidx 软件包中定义的 \printindex 命令读入的。其由一个 theindex 环境组成, 这是利用输入文件得到每个项的。
- .ins <sup>2ε</sup> 为了便于在 .dtx 文件上运行 DocStrip, 就在安装文件中放一些必须的命令 (和选项)。只要简单地 (利用 T<sub>E</sub>X 或 L<sup>A</sup>T<sub>E</sub>X) 处理 .ins 文件, 就可以从中提取出所需指令。
- .ist 这是索引样式文件, 由为 MakeIndex 提供的样式设置组成。基本名反映

出样式的性质，与任一文本文件没有关系。对于 doc 宏包，为指定的索引格式提供了 gind.ist 和 gglo.ist 文件。

- .lis 在有些系统上这就是 .log 文件的替身。
- .lof 这个文件由插图列表的信息组成。其行为与 .toc 文件完全一样，只是它是由 \listoffigures（不是 \tableofcontents）命令打开的。
- .log 这个文件由 T<sub>E</sub>X 处理过程中的协议组成，也就是说显示在终端上的信息以及当出错时，只有对 T<sub>E</sub>X 专家才有用的其它信息，都会写到这个文件中。即使用了 \nofiles，也会生成这个文件。当该文件被关闭时，会在终端上显示出如下信息：

Transcript written on 主文件名.log

- .lot 这个文件由表格列表信息组成。它同 .toc 文件的行式完全一样，只是它是由 \listoftables 命令打开的，不是用的 \tableofcontents 命令。
- .ltx <sup>2ε</sup>在 I<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 中，某些输入给 initex 程序的文件具有扩展名 .ltx。这些文件建立起生成 latex.fmt 格式的基础。这个文件的主文件名称为 latex.ltx。
- .sty 这是 I<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 中的宏包文件，用 \usepackage 调入。宏包是由定义新功能或修改已存在功能的其它命令组成。其中不能包含任何可显示的文本。也可以用命令 \RequirePackage 使它成为另一个宏包的输入。

在 I<sup>A</sup>T<sub>E</sub>X 2.09 中，.sty 文件既是样式选项文件（宏包），也是主样式文件（与类等价）。这是两种截然不同的性质，但是它们却具有相同的扩展名。它们是由 \documentstyle 命令读入的。

- .tex 包含用户源文本的文件应该具有扩展名 .tex。每个 I<sup>A</sup>T<sub>E</sub>X 文档都至少要有有一个 .tex 文件。如果只有一个 .tex 文件，那么它就也是决定其它所有文件基本名的主文件。如果在源文本中有 \input 或 \include 命令，那么在文档中就会有其它基本名不同，但后缀仍是 .tex 的文件。主文本就是调用 I<sup>A</sup>T<sub>E</sub>X 程序时用到的名称。其中有最高层次的 \input 命令，而且 \include 命令也只能出现在主文件中。
- .toc 该文件由目录表信息组成。如果 .toc 文件存在，那么 \tableofcontents 命令就会从其中读入信息，然后输出目录表。同时，打开一个新的 .toc 文件，写入当前运行过程所有新的目录信息。该文件由 \end{document} 命令关闭，因此如果在 I<sup>A</sup>T<sub>E</sub>X 运行完毕之前通过其它方法终止的话，那么就会失去 .toc 文件。

只有文档中包含 \tableofcontents 命令，而且没有调用 \nofiles 命令时，才会生成 .toc 文件。它与主文件具有相同的基本名。

## §8.10 报告材料的准备 (S<sub>Li</sub>T<sub>E</sub>X)

L<sup>A</sup>T<sub>E</sub>X 2.09 提供了与自己平行的称为 S<sub>Li</sub>T<sub>E</sub>X 的版本, 用来生成报告材料, 如彩色的幻灯片或视图。这个特殊的 L<sup>A</sup>T<sub>E</sub>X 变体利用了不同的字体集, 这比字母通常的书籍样式更适合于进行投影。然而, 这需要另一种 T<sub>E</sub>X 格式 (1.2.1 节), 因此原来版本的 L<sup>A</sup>T<sub>E</sub>X 把某个字体集显式地程序设计在格式文件中。因此用 S<sub>Li</sub>T<sub>E</sub>X 以示与通常的 L<sup>A</sup>T<sub>E</sub>X 的区别。运行的是 S<sub>Li</sub>T<sub>E</sub>X, 而不是 L<sup>A</sup>T<sub>E</sub>X, 实际上差别就在于前者用的是 `splain` 格式, 而后者用的是 `lplain` 格式来调用 T<sub>E</sub>X 程序。

在 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 中, 利用新字体选择方案, 在一个 L<sup>A</sup>T<sub>E</sub>X 格式中替换所有的字体再不会有任何问题。因此 S<sub>Li</sub>T<sub>E</sub>X 再不只是与 L<sup>A</sup>T<sub>E</sub>X 相似, 但独立之外的事物, 而且也不需要再给另外一个名称了。因此我们用 `slides` 类生成报告材料。在 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 中该类同其它类的选择方式完全一样, 而在 L<sup>A</sup>T<sub>E</sub>X 2.09 中, `slides` 样式只能在 S<sub>Li</sub>T<sub>E</sub>X 中选用, 而且事实上它也只接受这一种样式。

从此当我们用 S<sub>Li</sub>T<sub>E</sub>X 这一名称时, 它只是相对于 L<sup>A</sup>T<sub>E</sub>X 2.09 而言的, 而它与 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 中的 `slides` 类有相当大的差别。

### §8.10.1 `slides` 类

我们所谓的报告材料, 就是那些要投影给观众看的幻灯片或者视图。我们当然可以利用通常的 L<sup>A</sup>T<sub>E</sub>X 命令编写这些材料, 但是如果得到适当的字体尺寸, 如何安排覆盖, 确保文本不会出现不期望的分页, 书籍样式的字体是否适合于投影方面存在一些问题。L<sup>A</sup>T<sub>E</sub>X 的类 `slides` 就是尝试解决这些问题的。

在这个类中用的是一组不同的字体集, 其中的小写字母比通常的字体要大一些, 而且基本尺寸更大。这当然会限制了一页 / 片上的文本量, 但是这对做报告材料就非常好。这里的文本应局限于关键词和简略的句子。满满的一页普通文本即使投影在屏幕上, 也不会有观众去看。

投影片也应该利用彩色。从这一点上来看, S<sub>Li</sub>T<sub>E</sub>X 就更应该更新了。当在八十年代中期人们创造出第一个 L<sup>A</sup>T<sub>E</sub>X 时, 还没有理想价格的彩色打印机可以使用。一种变通的方法是利用彩色层, 也就是每种颜色的文本 (用黑白色) 单独打印出来, 然后把它复印到透明片上, 接着叠加起来就可以了。到了今天, 有更好的可以直接生成彩色投影片的方法。虽然对彩色的支持并不是 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 核心的一部分, 但是宏包 `color` 却可以针对某些驱动器提供彩色命令 (D.3.3 节)。在 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 中任何地方都可以用这个宏包, 并不是只能用在 `slides` 类中的。

即使没有彩色的功能, `slides` 在利用特殊字体和其它优势, 生成黑白视图方面也是非常有用的。通常 L<sup>A</sup>T<sub>E</sub>X 的格式命令大多都可以使用, 只有分页

和章节命令例外。在下面一节中描述只属于 `slides` 的特殊功能。

在  $\text{\LaTeX 2}_\epsilon$  中并没有提供彩色层命令，除非你用的是兼容模式才可以用。

### §8.10.2 制作幻灯片的环境

生成幻灯片的源文件在结构上同通常文档类似，只是这里用的是 `slides` 类：

```
 $\LaTeX$ \documentclass{slides}
```

  导言文本

```
\begin{document}
```

  幻灯片文本

```
\end{document}
```

如果希望得到真正的彩色（不是黑白加彩色膜），那么可以如下在导言中加入 `color` 宏包：

```
 $\LaTeX$ \usepackage[dvips]{color}
```

因为这个软件包并不是专为 `slides` 类定义的，因此我们在 D.3.3 节对它进行讲述。

在导言中可以包含某些全局性规定，如像通常那样改变纸张尺寸或选择纸张样式。在这里不能包含任何可显示的材料。

任何出现在 `\begin{document}` 命令之后，但在下面将要描述的特殊幻灯片环境之前的文件，都输入到一个没有编号的首页上，它位于所有幻灯片的前面。可以用它做为幻灯片的封面。

在组织幻灯片不同部分时，有三个环境可以使用：主幻灯片自身，可能用的覆盖，和对幻灯片的注解。

#### 幻灯片

幻灯片或视图是用如下环境生成的：

```
 $\LaTeX$ \begin{slide} 文本和命令 \end{slide}
```

`slide` 环境中的内容可以是任何所期望的文本。注意 `slides` 类使用的是自己字符字体集。正常尺寸中的标准字体大致相当于  $\text{\LaTeX}$  的 `\LARGE` 尺寸的 sans serif 字体 `\sffamily`。可以使用 4.1 节中的通常字体命令和声明，以及第 4 章中的其它显示和列表环境。然而，在环境中不能出现分页，因为整个文本希望在一页（幻灯片或视图）上结束。后面的 `slide` 环境会顺序编号的。如果一页出现了溢出，给显示出警告信息。

可以使用来自于 `color` 环境的彩色命令，前提条件是这个软件包已经被调入。这样就会直接输出彩色，会不必需要以往  $\text{\LaTeX}$  中的彩色膜。

#### 覆盖

所谓 覆盖 就是幻灯片的补充，它可以放在幻灯片上面，以弥补某些不

足。基本想法就是在报告时通过稍后才填上某些关键词，或者能够替换某些文本，来创造出某种悬念。

`slides` 类是利用 `overlay` 环境生成覆盖的，其作用方式同 `slide` 环境完全一样，只是编号是前面幻灯片的子编号。因此跟在第 6 号幻灯片后面的覆盖编号为 6-a, 6-b 等等。

<sup>25</sup> `\begin{overlay}` 文本和命令 `\end{overlay}`

生成视图的 `slide` 环境若要有覆盖，那它应在相应的 `overlay` 环境之前。幻灯片和覆盖环境中应包含相同的文本，只是某些部分用下面的声明以不可见的墨水显示：

`\invisible` 和 `\visible`

这两条命令的作用方式与字体声明一样。可以把它们放在大括号 `{}` 内以限制其作用范围，也可以用于整个环境。通常在 `slide` 环境中只会使某几个单词不可见，而在相应的 `overlay` 环境中，在开头处放上 `\invisible` 声明，而只有幻灯片中看不见的部分才是可见的。具体见 227 页的样例。

覆盖的一种应用就是提供替换文本。例如我们可以显示一张成本估计的表格，然后把图示放在覆盖上。通过交换这些覆盖，那么只要进行某些程序修改，所得到的新数可能适合于同样的表格。

## 注解

在报告的过程中，经常需要在进行实际的投影幻灯片时引用一些关键词或其它注解。`slides` 类提供了 `note` 环境，可以生成这样的给报告人的提示。

`\begin{note}` 文本 `\end{note}`

同 `overlay` 一样，注解也是相对于前面的幻灯片进行子编号的，只是这个用的是数字，不是字母。例如，接在第 4 号幻灯片后面，注解的编号为 4-1, 4-2, 等等。

### §8.10.3 其它功能

#### 页面样式

L<sup>A</sup>T<sub>E</sub>X 命令 `\pagestyle{样式}` 也可以用在 `slides` 类中。可以使用下面这些样式：

**plain** 所有的幻灯片、覆盖和注解的编号都是显示在右下角。

**headings**

同 **plain** 一样，只是如果选择了 **clock** 选项 (见下)，就会在注解的左下角显示一个时间标记。这也是默认的页面样式。

(**headings** 和 **plain** 样式在 SLiTeX 中有很大的差别，前者这时还会绘制对齐标志。)

**empty** 打印出来的页面上没有页码 (或者对齐标志)。

在通常的 L<sup>A</sup>T<sub>E</sub>X 中, `\pagestyle` 命令是在导言中调用的, 从而对整个文档都有效。单个页面可以利用 `\thispagestyle` 命令给出不同的样式, 这条命令只对当前一页有效。在上面环境内部不能用这条命令, 而 `\pagestyle` 命令却可以用在 `slide`, `overlay` 和 `note` 环境外面。

### 时间注解

在 `\documentclass` 的选项中可以选 `clock`; 它激活如下两条命令:

`\settime{秒数}`

`\addtime{秒数}`

这会以分钟为单位在注解的底部显示出时间。通过这种方法, 可以提醒报告人在该处可以做怎样程度的讲述。时间标志的值是用上面那两条命令进行设置和增加的, 这里的参数值以秒为单位。记时器的初始值为 0。

### 有选择地处理幻灯片

如果幻灯片文件要处理很多的 `slide` 环境, 而且有可能用户只想改变其中很少一部分幻灯片, 这样就没有必要再全部重新输出一遍。这可以用下面的命令做到这一点:

`\onlyslides{幻灯片清单}`

把这条命令放在导言中。这里的幻灯片清单表示升序的一组幻灯片编号, 例如, 2, 5, 9-12, 15, 用来指定要处理的幻灯片或幻灯片范围。

不存在的幻灯片编号也可以出现在幻灯片清单中。比如说在幻灯片文件中只有 20 张幻灯片, 那么 `\onlyslides{1,18-999}` 命令将会使得只有第 1 号和 18-20 号幻灯片被处理。属于这些幻灯片的覆盖也同时被输出。

最后, 在导言中的命令

`\onlynotes{注解清单}`

使得只有列在注解清单中的注解才会被输出。假设第 5 号幻灯片有三个与之相关联的 `note` 环境, 那么 `\onlynotes{5}` 将会使得只有页码为 5-1, 5-2 和 5-3 的注解被输出。

`\onlyslides` 和 `\onlynotes` 命令同在 8.1.2 节中描述的 `\includeonly` 命令在功能上很相像。也可以同 8.1.3 节中描述的通过利用 `\typein` 命令一样, 实现幻灯片和注解的交互式选择:

`\typein[\slides]{Which slides to do?}`

`\onlyslides{\slides}`

在处理过程中会在屏幕上显示出 ‘Which slides to do?’ 的信息, 这时用户就可以输入所期望处理的幻灯片。对 `\onlynotes` 也可以类似操作。

### 幻灯片文件样例

下面给出一个使用 `slides` 类的输入文件例子, 其中包含了首页、覆盖和

注解, 也使用了 `clock` 选项。同时在图 8.1 中显示了四页的输出结果。

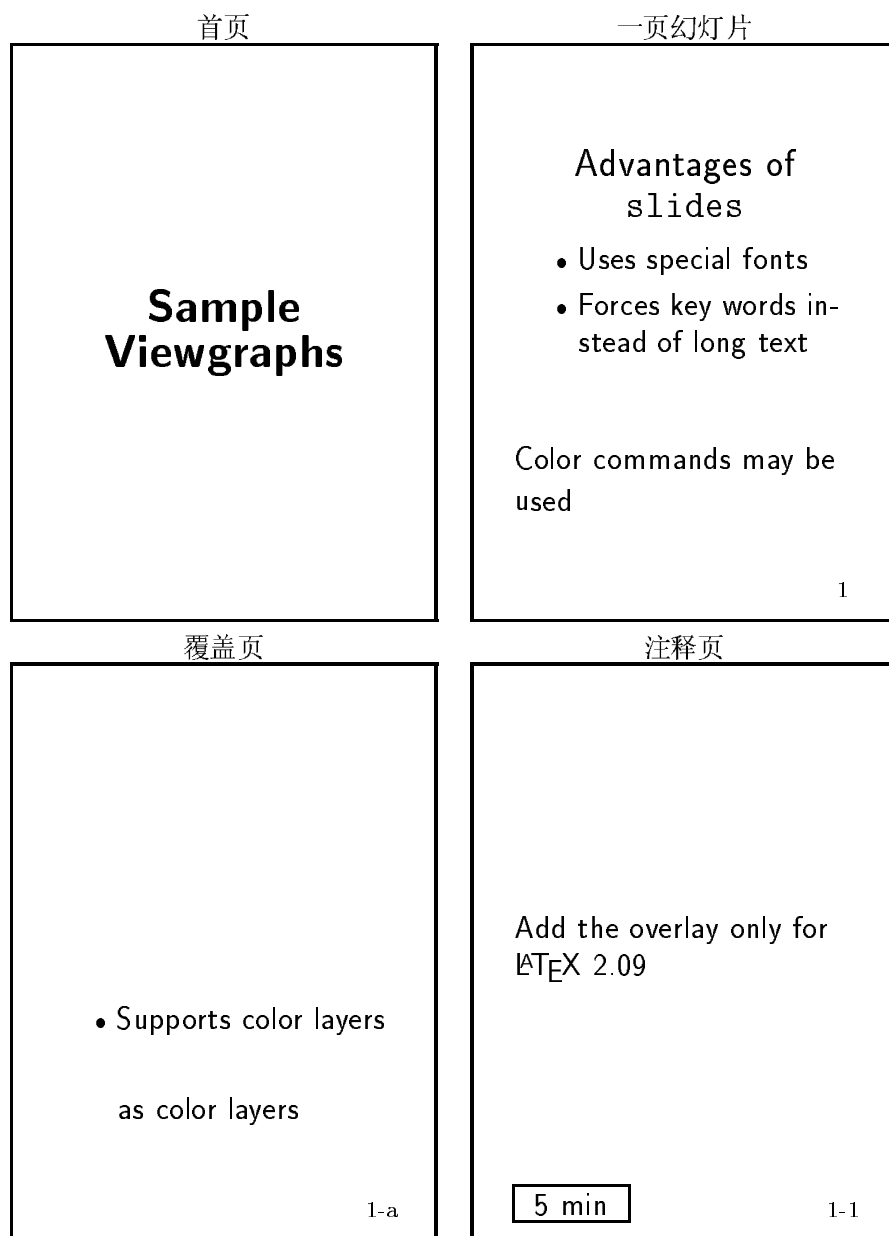


图 8.1: 幻灯片文件样例

```
\documentclass[a4paper, clock]{slides}
\begin{document}
\begin{center}\Large\bfseries
Sample Viewgraphs
```

<http://202.38.68.78/texguru>

Email: [texguru@263.net](mailto:texguru@263.net)



```
\end{center}

\begin{slide}
\begin{center}\large Advantages of \texttt{slides}\end{center}

\begin{itemize}
\item Uses special fonts
\item Forces key words instead of long text
\invisible
\item Supports color layers
\visible
\end{itemize}

Color commands may be used {\invisible as color layers}

\end{slide}

\begin{overlay}
\invisible
\begin{center}\large Advantages of \texttt{slides}\end{center}

\begin{itemize}
\item Uses special fonts
\item Forces key words instead of long text
\visible
\item Supports color layers
\invisible
\end{itemize}

Color commands may be used {\visible as color layers}
\end{overlay}

\addtime{300}
\begin{note}
Add the overlay only for \LaTeX~2.09
\end{note}
```

\end{document}