

附录D 扩展 L^AT_EX

L^AT_EX之所以变得像现在这样普及，就是因为能对它进行重新开发：如果基本安装（核心）中缺少了某些功能，那么经验丰富的用户就可以通过写一个宏包来弥补这一点。在 L^AT_EX 2.09 流行的年代里，借助于计算机文件服务器，大批的宏包（那时称为 样式选项）得以普及。其中绝大多数现在与 L^AT_EX 2_ε 还是兼容的，其余的被进行了转化，当然还有更多的正在编写过程中。

本书的目标只限于标准 L^AT_EX 宏包，并不想过多涉及其它宏包内容；可以在 Goossens 等人的 *The L^AT_EX Companion* (1994 年) 一书中找到对其它一些宏包的介绍。我们在此只是对几方面的扩展进行大致的描述，包括多语言 L^AT_EX，PostScript 字体，图形的插入和彩色的使用。然后我们讲解一些基本安装的高级功能，例如宏包的档案集成化，某些半正式扩展，最后是 L^AT_EX 2_ε 的安装方法。

所有本附录列出的 L^AT_EX 宏包都可以在 D.5 节列出的地点中找到。

§D.1 国际化的 L^AT_EX

在 L^AT_EX 起初的版本中，有些命令中显式包含一些英文单词，例如 ‘Figure’ 和 ‘Bibliography’。这一事实与程序设计的一条良好规则发生冲突，即永不要显式地做任何事情。在欧洲，L^AT_EX 用户们很快就进行了改编，使其适合于自己的母语。相对于德语的改编，即宏包 `german.sty`，就是一组由维也纳技术大学的 H. Partl 收集的宏指令，这些宏指令是由不同的人员开发的，现在这个宏包已成为那些讲德语的 T_EX 用户组 (DANTE) 的标准。这个宏包中也包含一些处理其它语言（如法语和英语）的功能。下一节给出这部分的详情。

`german.sty` 宏包的关键就是对标准 L^AT_EX 文档样式（类）进行更新，输出中的显式英文单词已被名称命令代替。为了适用于所有语言，在欧洲用户间这些名称已经被标准化了。来自 Darmstadt 的 J. Schrod 对 L^AT_EX 版本进行了修改，得到了著名的 II^AT_EX，即国际化的 L^AT_EX。可以在 316 页上找到这些名称及其标准英文值。

到了 1991 年 12 月 1 日，这些名称特征成为 L^AT_EX 标准的一部分。它们代表良好程序开发的习惯，因为通过这种方法，即使是英语用户也可以很容易地改变某些标题，例如，把 ‘Abstract’ 改成 ‘Summary’，或者把 ‘Contents’ 改成 ‘Table of Contents’。

另外一种要应于多语言用法的重要工具是 T_EX 的 `\language` 计数器，通过它在 `initex` 上载的格式中可以保存不只一组的断词式样 (D.4 节)。通过把 `\language` 设成不同的值，可以激活不同的式样。只有 3.0 或更高版本的 T_EX 才具有这种功能。

§D.1.1 `german.sty` 文件

在 1987 年 10 月召开的第六届德语 \TeX 用户大会上, 通过了一组标准的 \TeX 和 \LaTeX 命令以应用于德语文档中。宏包 `german.sty` 是由维也纳技术大学的 H. Partl 建立起来的, 随后有很多人对它进行了扩展。最新版本是 2.5b, 发布日期是 1995 年 1 月 20 日, 现在是由 Stuttgart 大学的 Bernd Raichle 负责它的维护工作。这个宏包不仅打算在 $\text{\LaTeX} 2_{\epsilon}$ 下工作正常, 而且在 Plain \TeX 和 $\text{\LaTeX} 2.09$, 标准的字体编码以及 Cork 编码中都能发挥其正常功能。

我们在此并不打算对 `german.sty` 文件进行完整的说明, 因为在 C.3.3 节对 `esperant.sty` 的介绍已几乎包含了所有一般性语言改编的要点。事实上, `esperant.sty` 就是从 `german.sty` 中演化而来的, 可以认为是它的一个小型衍生宏包。在本附录中, 我们演示的是其中包含的相应于德语的其它功能。在下一节, 我们对 `french.sty` 宏包做同样的事情。

由于 `german.sty` 的开发是受国际化 \LaTeX 和多语言系统 `babel` 的启发, 因此在这一点上我们要额外注意。

在 `german.sty` 出现之前就存在着一个相应于德语的比较原始的小改编, 它由三条命令组成:

```
\catcode'\\"=\active \def"{\"} \def\3{\ss}
```

上述指令使得双引号 " 成为命令 (主动字符), 并把它定义成变音重音 (\"), 把 \3 定义成德语的双写 s, 即 *eszet*, ß。因此上面的指令只是简化了德语文本的输入。

双引号命令符号

现在的 `german.sty` 精心改进了双引号命令的定义, 使得

- 在计算机现代字体中的变音离字母的位置要比通常 \LaTeX 中的近, 例如 ö("o) 和 ö(\o);
- 双引号可以用在 `\verb` 命令和 `verbatim` 环境中;
- 有变音的单词也可以进行断词;
- 也可以用 "s 显示出 *eszet*; 但出于一致性的考虑, 原来的 \3 仍然有效, 但不鼓励这种用法。

双引号符号也可以实现德语中其它特殊功能, 特别是单词的断开:

"ck 在德语中, 有一个断词的规则, 那就是如果要在 ck 间断开, 那它就成为 k-k。这可以通过输入 "ck 来实现, 例如 Dru"cker。

"mm 另外一条规则就是当两个单词合写时, 那么至多只能有两个重复字母 (例如, *Schwimm + Meister = Schwimmeister*), 可是如果单词断点恰恰就在它们之间时, 省掉的字母就又要被加上 (*Schwimm-meister*)。在文本中可以输入 Schwi"mmeister) 做到这一点。当然也有可能存在其它的二 -

三字母情形。

- "- 建议断词点是用 "-" 表示的，这表示允许在此处进行自动断词，有时候通常的 L^AT_EX 命令 \- 没法做到这一点。
- "| 最后提一点，如果连写是不恰当的，可以用 "|" 断开，例如两个相邻单词间的字母 (*Auf* + *Lage* = *Auflage*，输入为 `Auf"|lage`，否则结果为 *Auflage*)。

引用记号

德语不使用英语中的‘引号’，它用的是 „鹅脚“ 形式的 ‚单双引号‘。生成这两组符号的命令分别是：\glqq 表示左双引号 („)，\grqq 表示右双引号 (“)，\glq 表示左单引号 (‚)，\grq 表示右单引号 (‘)。两个双引号命令可以缩写为 "‘ 和 "’。

也提供了法语中的 « 引号 »，它是利用 \flqq 和 \frqq 命令输入的。可以缩写为 "< 和 ">。而 < 单引号 > 是利用命令 \flq 和 \frq 生成的。这两者都经常出现在德语文本中。

标题和日期

国际化 L^AT_EX 中的名称命令也可用类似 316 页上的世界语方式，也就是利用命令 \captionsgerman，在德语中激活。可以用 \captionsenglish 命令切换回原来的模式；在 german.sty 中还有一条可用的命令 \captionsfrench，但是可以利用 french.sty (D.1.2 节) 得到更好的法语改编。

\today 会显示出当前日期，可以用声明 \dategerman, \dateaustrian, \dateUSenglish, \dateenglish 和 \datefrench 等进行重定义。其中两个英文形式的定义与 316 页上的一样。根据所选择的语言不同，\today 的结果形式也就不同。

语言选择

即使上载了 german.sty 宏包，也可以用利用 \selectlanguage 命令选择法语和英语，这条命令的参数值为 german, french, english, USenglish 或 austrian 中的一个。它通过把 \language 设成不同的值，而激活相应的断词式样。

§D.1.2 french.sty 文件

法语样式文件 french.sty 起初是由一批用法语编写文档的用户独立于 german.sty 而开发的。随后由 Bernard Gaulle 对它进行扩展与维护，Gaulle 把自己的全副精力奉献于使用法语的 T_EX 用户组 (GUIenberg) 中。当前版本是 3.32，发行日期为 1994 年 8 月 9 日。

我们在此讲解一些法语样式文件中可用的特殊命令，它们既可以用在 \TeX 中，也可以用在 \LaTeX 中。同德语时的情形一样，这样做的目的在于说明必要修改的实质以及问题的复杂性。

在德语样式 `german.sty` 中我们的重心是放在如何简化变音重音上，以及调整德语断词规则上，而与德语的情形不同，在 `french.sty` 中并没有特别的重音命令，而要考虑法语的断词以及缩写。这种修改可以组织起来，分为五种不同类型，每种类型都有两条命令，分别具有激活和关闭功能。概括命令 `\french` 会调用所有 5 条激活命令。

断词

利用 `\frenchhyphenation`，可以使 \TeX 切换到法语断词式样中，它是用 `initex` 上载到格式中的。这也就是说单词的断开是按照法语的规则进行的。利用相反的命令 `\nofrenchhyphenation`，就又切换到标准的式样中，即第 0 号语言，这通常就是英语。

标题和日期

`\frenchtranslation` 命令会调用 `\captionfrench`，以重定义在国际化 \LaTeX 中所有的名称命令。它也会把 `\today` 设成法语的形式。另外，还增加了一些名称命令，主要是针对于非标准信件类而引入的。另外还有一条与 `\abstract` 相似的命令 `\resume`，因为许多法语文章需要以英语和法语形式给出摘要 (*résumé*)。

命令 `\nofrenchhyphenation` 会关闭法语翻译。

标点符号

在法语的排版规则中，要求‘标点符号！？；：’前面有多余的空档。这个空档要比通常的单词间隔小；在源文本中，这些字符的前面应该是一个完整的空格（但请看下面的 `\untypedspaces`），当用了 `\frenchtypography` 命令后，这些空格是会自动减小的。 \TeX 命令 `\frenchspacing` 也同时被激活。

在法语中只有唯一的引号，在 \TeX 中 `« 形状 »` 与 \LaTeX 中 `« 形状 »` 不同。它是用 `<<` 和 `>>` 生成的，这与在 `german.sty` 中用 `\flqq` 和 `\frqq` 生成的引号有点儿差别。它们会在被包围文本两边插上额外的空档。被引用的文本也可以放在 `guillemets` 环境中。这样会在引用中每个段落的开始加上左引号。

另外还有许多可用的子选项，其中几个如下：

`\untypedspace` 即使在源文本中少掉了！？；：前面的空格，也会在它们前面插入额外的空档。

`\noTeXdots` 把 `\ldots` 和 `\dots` 命令改成显示法语模式的省略号 (...)，即三个紧相邻的句号，而不用英文中的方式 (...)。

`\todayguillemets` 在第二层次引用中的段落, 或者 `guillemets` 环境中的段落, 其前面都有左引号, 即与第一层次段落一样。到了今天, 这也是缺省的标准方式。

`\ancientguillemets` 在第二层次引用中的段落开始用的是右引号, 而不是左引号。这是古法语的风格。

`\noenglishquote` 把引号字符 ‘ 和 ’ 转化成重音符号 ` 和 ’ 。

`\noenglishdoublequotes` 把双引号 “和” 转化成左右引号。

`\idotless` 为重音 *i* 和 *ï* 生成没有点的 *i* 。

命令 `\nofrenchtypography` 就会关闭这些特殊的标点符号规则及子选项。

布局

利用 `\frenchlayout` 命令, 可以在 L^AT_EX 格式中进行如下几方面的改变。

- 所有的段落都要进行缩进, 即使一节开始的第一段也不例外。在 L^AT_EX 中通常第一段不进行缩进。
- 在 `itemize` 环境中每一层的 `\item` 标签都是横线。
- 每当开始新的部分, 都重设所有的章节命令。
- 定义了一个新的环境 `order`, 其类似于 `enumerate`, 但项的编号为 1^o 2^o 3^o ...。

可以用 `\nofrenchlayout` 命令去掉这些格式修改。

缩写

当调用了 `\frenchmacros` 时, 就会定义许多新的命令, 这些命令有助于显示常见的法语缩写。

`\ier 1\ier` 显示为 1^{er} 。

`\iere 1\iere` 显示为 1^{re} 。

`\ieme` 可以显示其它的基数, 例如 `3\ieme` 结果为 3^e 。

`\at` 显示 @ 。

`\bv` 利用 `\tt` 字体显示 l 。

`\chap` 显示 ^ 。

`\boi` 利用 `\tt` 字体显示反斜杠。

`\tilde` 显示 ~ 。

`\Numero` 显示 N^o 。

`\degres` 显示度数符号 ° 。

`\fup{χ}` 提升并缩小 χ , 例如 `M\fup{me}` 结果为 M^{me} 。

`\primo \secundo \tertio \quatro` 显示为 1^o 2^o 3^o 4^o。但后接右括号时, 圆圈会变得更小。

`\quando=n` 可以按上面的方式显示任何数值, 例如 `\quando={10}` 的结果为 10° 。

`\minMAJ{oe}` 在目录表和页眉中会自动根据大小写要求显示为 α 或 \mathfrak{E} 。也可以取其它的参数, 例如 `ae` 和 `i`。

可以用 `\nofrenchmacros` 关闭这些特殊命令。

语言选择

利用 `french.sty` 中的程序也可以切换回英语, 命令 `\english` 就可以做到这一点。随同的 `english.sty` 文件中包含了英文形式的标题名称以及相应的切换命令。

要使用已有断词式样的其它语言, 可以用 `\NouveauLangage[kaishu 数]{语言}` 进行定义。这条命令就会创建一条新命令 `\语言`, 它通过把 `\language` 记数器设成给定的 数 而切换到给定 语言。这时也会执行一条叫 `\语言TeX` 的命令, 以激活相应于该语言的其它功能, 必须另外定义这条命令。

§D.1.3 多语言 \LaTeX —babel 系统

在 `german.sty` 和 `french.sty` 宏包中都存在同样的一个问题, 虽然它们允许加进其它的语言, 但采用的都是不兼容的方式。因此在一个文档中就没办法同时使用这两种样式。

丹麦 PTT 研究实验室的 Johannes Braams 开发了 `babel` 系统。其主要目的就是提供一种标准方法, 在语言间进行切换, 所采用的是一种有弹性的方法, 来上载断词模式。已针对于 \TeX (2.x 和 3.x 版本) 和 \LaTeX (只有国际化版本和英文版本) 编写了这个系统。

在 312 页上我们列出了进行语言改编需要满足的三条要求: 对显式的英文标题进行翻译, 给出在相应语言中实现特定功能的简化命令, 选择相应的断词模式。为此, `babel` 系统增加了可以卸载特殊命令的功能, 以及对当前语言的测试功能。与某种语言有关的文件必须同 `babel` 选择命令的结构相匹配。在这个系统中也包含“方言”的概念, 也就是说某两种“语言”拥有同样的断词模式。例如德语与奥地利语, 以及英式英语与美式英语, 它们只是在日期格式上有点儿差别 (见 327 页)。

目前, 在 `babel` 宏包中包含如下语言的定义文件:

世界语 (Esperanto),

荷兰语 (Dutch), 英语 (English), 德语 (German),

法语 (French), 意大利语 (Italian), 葡萄牙语 (Portuguese), 西班牙语 (Spanish), 加泰罗尼亚语 (Catalan), 加利西亚语 (Galician), 罗马尼亚语 (Romanian),

丹麦语 (Danish), 挪威语 (Norwegian), 瑞典语 (Swedish),
法国布里多尼语 (Breton), 爱尔兰语 (Irish), 苏格兰语 (Scottish),
芬兰语 (Finnish), 匈牙利语 (Hungarian), 爱沙尼亚语 (Estonian),
捷克语 (Czech), 斯洛伐克语 (Slovak), 克罗地亚语 (Croatian), 斯洛
文尼亚语 (Slovene), Sorbian(大写与小写), 俄语 (Russian)(计划中), 波兰
语 (Polish),
土耳其语 (Turkish),
Bahasa.

在安装版本中会同时给出必需的定义文件。然而, 不同语言的断词模式却必须从另外的渠道取得。其中绝大多数文件, 并没有处理重音或拼写的特殊命令, 只是重定义了国际化 L^AT_EX 中的名称命令。

在 `babel` 的开发过程中, `german` 宏包具有很大的启示作用。然而, 由于 `\selectlanguage` 命令的用法稍有点儿不同, 因此必须为 `babel` 创建一个新的 `german.sty` 文件, 这个文件由同样的特殊命令集组成, 只是相对于 `babel` 稍做了点儿增加。同样地, 在 `babel` 中的法语文件被命名为 `francais.sty`, 以与 `french.sty` 区分开。

`babel` 文件

在 `babel` 安装中包含如下文件, 以实现所需要的目标:

`babel.def` 由运行 `babel` 所需要的一些基本宏组成, 必须被读入的第一个语言文件上载进来; 其余的宏就要么包含在格式文件中 (由 `hyphen.cfg` 上载), 要么从 `switch.def` 文件中读入; `babel.def` 确定当前状况, 并给出相应的反应;

`switch.def` 由其它的 `babel` 宏组成, 以备如果在 L^AT_EX 格式中没有定义这些宏时, 把它们读入进来;

`hyphen.cfg` 由与 `switch.def` 中相同的宏定义组成, 还包含一些运行 `initex` 时需要的宏定义; 如果在创建 L^AT_EX 2_ε 格式 (D.4 节) 时这个文件可用, 那么这些宏就会内建到格式中, 在运行时就不再需要 `switch.def`;

`babel.sty` 是主要的宏包文件, 它上载以选项形式给出的语言文件;

`language.dat` 由一组语言及相应的断词模式文件名组成; 在运行 `initex` 时, 这些文件会被 `hyphen.cfg` 文件读入进来; 为了特定的安装需要 (D.1.4 节), 这是唯一可以 (必须!) 进行编辑的文件;

`esperant.ldf...` 语言定义文件。

为了与原来的版本兼容, 还有一组 `.sty` 文件, 它们除了读入相应的 `.ldf` 文件外, 再不做任何事情。

调用 babel

在 \LaTeX 2_ϵ 中, `babel` 宏包是用指定的语言做为选项来上载的:

```
\usepackage[english,esperanto]{babel}
```

与之相应地, 语言名称也可以是全局选项, 如果有其它宏包也用语言做为选项, 这就是一种值得推荐的方法, 例如,

```
\documentclass[english,esperanto]{article}
\usepackage{babel,varioref}
```

在这两种情形中, 最后给出的名称相应的是马上被激活的语言。

在 1995 年 7 月 5 日发布的 3.5 版本 `babel` 系统中, 可以识别的做为选项的语言名称有:

american	danish	hungarian	scottish
austiran	dutch	italian	spanish
bahasa	english	lowersorbian	slovak
brazil	esperanto	magyar	slovene
brazilian	finnish	norsk	swedish
breton	francais	nynorsk	turkish
british	french	polish	uppersorbian
catalan	galician	portuges	
croatian	german	portuguese	
czech	germanb	romanian	

如果上载了 `babel` 宏包, 那么它所做的事情就是读入进来指定语言的宏包; 然后它们依次读入 `babel.def` 以及可能用到的 `switch.def` (如果在格式中没有相应的宏) 文件。

语言切换命令

通常用户不需要对 `babel` 内部操作有太多的了解。新高级用户命令是:

```
\selectlanguage{语言}
\foreignlanguage{语言}{文本}
\iflanguage{语言}{是文本}{否文本}
```

第一条命令把 `语言` 切换为当前语言, 而第二条命令用选定的 `语言` 设置一块文本, 在第三条命令中, 如果当前用的是指定 `语言`, 则执行 `是文本`, 否则执行 `否文本`。

另外, 在 `\language` 中包含当前选定语言的名称。最后, 任何语言名称都可以做为一个环境, 这样可以设置不同于正文语言的一大段文本。

当然, 每个语言定义文件也可以拥有自己的特殊命令, 以简化输入操作, 见 D.1.1 和 D.1.2 节中相应于德语和法语中的示例。

语言定义文件的内容

虽然并不需要知道切换机制的工作原理，我们这里还是做一简单刻划，供有兴趣者参考。

增加的两条 `babel` 内部命令是：

`\addlanguage{语言编号}` 以及

`\adddialect{语言编号 1}{语言编号 2}`

其中 `语言编号` 是一个内部的编号，它用来区分断词模式集合。`\addlanguage` 命令把其参数值设置成下一个可用的语言编号。在 `babel` 中 `语言编号` 的形式为 `\l@语言`；例如，`\l@english`。在 `initex` 执行时，会执行列在 `language.dat` 中每个语言名称对应的该类型命令。`\adddialect` 命令把一个参数的值设成与第二个参数的值相同，这样两者就可以使用同样的断词模式集合。例如，在 `english.ldf` 中就有一条命令 `\adddialect{\l@american}{\l@english}`。

语言定义文件中必须提供下述四条命令：

`\captions语言` 来重新定义类似于 `\tablename` 的名称命令；

`\date语言` 定义 `\today`；

`\extras语言` 定义所有与语言有关的命令；

`\noextras语言` 去掉所有与语言有关的命令。

如果这里的 `语言` 并不是具有预存贮在当前格式文件中的断词模式的一种语言（也就是说并没有定义 `\l@语言`），那就把它设成相应于第 0 号语言的“方言”。

最后再提一点，语言定义文件会调用 `\selectlanguage{语言}` 以激活该语言。其实现方法如下：

- 调用 `\language\l@语言` 以选择断词模式集合，
- 为了去掉可能存在的与语言有关的命令，调用 `\originalTeX`，
- 激活 `\caption语言`，`\date语言`，以及 `\extras语言` 命令，以调用与语言有关的名称，日期和命令，
- 把 `\originalTeX` 重定义为 `\noextras语言`，这样下次语言切换时就会去掉这里与 `语言` 有关的所有功能。

§D.1.4 使用 `\language` 命令

前面已提到过，在 `TeX` 的 3.0 版本或以后版本，可以在一个格式文件中保存不只一组断词模式。通过给 `\language` 计数器设成不同的值，可以在其中进行切换。

然而，还没有标准规定哪一种语言相应于哪一个编号。在 C.3.3 节中的 `esperant` 宏包，D.1.1 节中的 `german` 宏包都假定了某种顺序，但是在其它情形中就有可能行不通。`babel` 系统调用了一个相当可靠的过程，这个过程也被 `french` 所使用。

在格式创建时读入的文件 `language.dat`，由一组语言后加断词模式文件名组成。举例来说，如果 `language.dat` 包含

```
english    hyphen.tex
germanb    ghyphen.tex
français   fhyphen.tex
```

那么在第 0, 1 和 2 号语言中就会分别上载保存在 `hyphen.tex`, `ghyphen.tex` 和 `fhyphen.tex` 中的断词模式，而 `\l@english`, `\l@germanb`, `\l@français` 就分别定义成 0, 1, 2 号，以供 `\selectlanguage` 命令使用。因此 `français` 语言就直接与法语断词联系在一起。在使用法语的研究所中，有可能用的是 1 号语言，而不是 2 号，但这是无关紧要的。关键在于编号保存在格式中，并且可以通过标准的名称来引用。

在 `initex` 运行 (D.4 节) 时，通过提供 `hyphen.cfg` 文件，可以生成这样的格式。为了适应局部安装，所需要定制的唯一文件是 `language.dat`。

§D.2 \LaTeX 与 PostScript

做为打印和绘图语言而引入的 PostScript，已成为了一种打印质量良好的文件标准，这样的文件拥有足够的弹性，而且便于以电子版本传递到别处。现在的许多 DVI 驱动程序都可以把 \TeX 的 `.dvi` 输出转化成这种更具普遍性的输出格式。其中最出名的当属 `dvips` 了，这是一个由 Tomas Rokicki 开发的非经营性程序，可以从 CTAN 文件服务器上获取 (D.5 节)。

PostScript 可以为 \LaTeX 提供许多额外的功能，例如引入外部图形，彩色，放缩以及旋转。在 `dvips` 的安装中包含许多宏包以实现这些功能。然而，它们只在这种驱动程序中才可以使用；其它程序用的是不同宏包和命令。做为对 $\text{\LaTeX} 2_{\epsilon}$ 的一部分正式扩展，为此已经开发了一组标准命令，而把与驱动程序有关的代码隐藏在 `.def` 文件中。通过这种方法，对所有驱动程序，高级用户命令具有相当的一致性，而只需要利用 `graphics` 或 `color` 宏包的选项来指定驱动程序。

由于这些功能并不只限于 PostScript，因此我们在稍后的 D.3.3 节中再加以介绍。

§D.2.1 调用 PostScript 字体

利用 PostScript 驱动程序来输出 \LaTeX 文档，也就意味着可以使用许多 PostScript 字体，特别是现在 NFSS 成为 $\text{\LaTeX} 2_{\epsilon}$ 的一部分后更是如此。

为了简化对这些字体的选择，要找来一组宏包。在 CTAN 服务器 (见 353 页上的图表) 中 `psnfss` 目录中可以找到这些宏包。其由宏包文件以及所需的字体定义文件 `.fd` 组成，这些定义文件相应于 35 种标准 PostScript 字体，以

表 D.1: psnfss 宏包及相应字体

宏包	<code>\rmfamily</code>	<code>\sffamily</code>	<code>\ttfamily</code>
<code>times.sty</code>	TimesRoman	Helvetica	Courier
<code>palatino.sty</code>	Palatino	Helvetica	Courier
<code>newcent.sty</code>	NewCenturySchoolbook	AvantGarde	Courier
<code>bookman.sty</code>	Bookman	AvantGarde	Courier
<code>avant.sty</code>		AvantGarde	
<code>helvet.sty</code>		Helvetica	

及其它一些可用的商品化字体。这些宏包的用法相当简单；例如，`times.sty` 文件只是由三行文本组成，在 212 页上列出了其内容。所有这些宏包要做的事情就是重定义三个字体族 `\rmdefault`、`\sfdefault` 以及 `\ttdefault`。

我们在表 D.1 中列出了相应于 35 种标准字体的宏包，以及它们做出的三个赋值。（这 35 种字体包含了自己的斜体以及黑体变体。）

在 `psnfss` 目录中没有必需的字体尺寸文件 `.tfm`。这些文件位于 CTAN 的 `font/metrics` 目录中，每种字体相应于一个子目录。例如，TimesRoman 字体的尺寸文件在 `ptm` 子目录中。

这些字体利用了 T_EX 的虚拟字体机制，即 T_EX 看到的实际上只是假字体，这种字体并不存在。由于 T_EX 只是读入 `.tfm` 文件，因此这是完全行得通的。驱动程序接着就读入 `.vf` 文件，而不再读像素文件 `.pk` 或 `.pkl`。在虚拟字体文件中的指令告诉驱动程序如何生成每个字符：可以是绘制出来，也可以得自其它字体，或者进行变形。这也就是为什么当某一字体的 `slanted` 字体和小体大写变体并不存在时，可以用“粗糙”的方式得到 PostScript 变体。

甚至虚拟字体的字体布局也可以被重新设计。起初的 PostScript 字体所采用的编码框架并不遵从 CM 和 DC 布局（见 E.6 节），而虚拟字体则遵从这一布局。在计算机现代字体中，前 11 个位置是大写的希腊字母（见 368 页上的布局 1），而只有在 PostScript 符号字体中才能找到这些字母。`ptmr` 虚拟字体为了遵从这一点，采用的方法是从原始的 TimesRoman 和 Symbol 字体中选取这些字符。

为了正确地打印出来结果，我们必须从尺寸目录中得到所有的 `.tfm` 和 `.vf` 文件。

§D.3 其它的 L^AT_EX 附件

§D.3.1 DocStrip 工具

我们知道 T_EX 程序既是一个文本格式化语言，也可以进行程序设计，根

据这一点，我们可以利用它，开发一组实用程序进行文件操作。有了这样的程序，那么就会非常方便了：如果你有 \TeX ，那么你就可以执行这些程序。`DocStrip` 就是这样的一个程序，它由 Frank Mittelbach 开发，该程序是把文件中的注释行去掉。

粗粗一看，你可能会问，为什么要做这样一件事呢？因为注释总是被忽略的。在 \LaTeX 的原始发行版本中，Leslie Lamport 所给出的支持文件中有相当丰富的注释，其甚至比代码还要长几倍。在当时，存储空间和速度都是非常珍贵的，因此为了实用的需要，他同时提供了没有注释的同样文件。为了以后参考，可以把有注释的版本存贮在其它地方。Mittelbach 的想法就是创建一个程序，为了紧致存贮和加快上载速度，而精简掉那些注释行。

根据这个简单的概念，我们要对 `DocStrip` 进行扩展，给它增加另外两个功能：根据处理时的选项，可以有选择地抑制或包含某些代码行；可以把多个文件，也称为模块，组合成单个文件。这也就是说，在一个源文件中可以存放有几个不同的 \LaTeX 宏包，一个宏包文件也可以利用不同的成分来构造起来。事实上，主要的 $\text{\LaTeX} 2_{\epsilon}$ 生成文件 `latex.ltx` 就是由 30 多个单独文件构成的，而标准的类文件以及大多数公共代码就是利用其选项文件，从一个名为 `classes.dtx` 的文件中提取的。

只有利用 D.3.2 节中的集成文档系统，我们才能看到 `DocStrip` 现在所具有的最重要应用。然而，它还有其它用途，例如在 B.3.2 节列出的通用参考文献样式文件。它有点儿类似于 Unix 系统中的 C 编译器和 `makefile` 工具，只是这里要运行在 \TeX 中，从而可以方便地移植到其它系统中。

交互执行

对一个文件执行 `DocStrip` 的最简单方法就是用 \TeX 调用它 (也可用 \LaTeX , 但 \TeX 速度更快)，例如，

```
tex docstrip
```

这样就会得到如下反馈：

```
*****
* This program converts documented macro-files into fast *
* loadable files by stripping off (nearly) all comments! *
*****
*****
* First type the extension of your input file(s): *
\infileext=
*****
```

一种反应就是给出输入扩展名 (通常为 `.dtx`)；然后依次被要求给出输出扩展名，所需选项，最后是被处理文件的基本名。接着就开始执行。

这种方法有一些局限性，因为输入和输出文件有相同的基本名，只是扩展名不同而已，而且加到输出文件中开始与结尾时的注释是固定的。运行这个工具的更富有弹性的方法是采用批处理作业的形式。

利用批处理作业来执行

DocStrip的批处理作业就是一个文件，其由该工具识别的指令组成。例如，假设在 C.3.1 节示例的 `fullpage` 宏包被放到一个名为 `fullpage.dtx` 的档案源文件中，真正的宏包文件是 `fullpage.sty`，可以用选项 `package` 提取出来；那么批处理文件 `fullpage.ins` 就应该如下所示：

```
\def\batchfile{fullpage.ins}
\input docstrip

\preamble
This is a stripped version of the original file.
\endpreamble

\postamble
This is the end of the stripped file.
\endpostamble

\generateFile{fullpage.sty}{f}{\from{fullpage.dtx}{package}}
```

前两行是至关重要的：`\batchfile` 定义为包含指令的文件名称；接着上载 `docstrip.tex` (`\input` 命令的写法必须完全与这里给出的一样)，接着就读入并执行指定的文件，并停下来！剩下的指令行当 DocStrip 读入 `\batchfile` 时就被处理过了。事实上，它们可以一起位于另外一个没有这里前两行内容的文件中；唯一的要求是 `\batchfile` 必须精确定义为包含指令的那个文件名。

`\preamble` 和 `\postamble` 命令使我们可以在开始与结尾处包含解释性的注释。

主要命令是 `\generateFile`，它用输出文件名作为第一个参数值。第二个参数值为 `t` 或 `f`，告诉 DocStrip 如果已经有一个与输出文件同名的文件存在时，是否给出一条警告消息。这里 `f` 表示 `<false>`，不给出警告消息。第三个参数值由一条或多条 `\from` 命令组成，其两个参数值表示输入文件的名称以及有选择编码时的选项。

这样生成的 `fullpage.sty` 文件内容如下：

```
%% This is file 'fullpage.sty', generated
%% on <1994/10/15> with the docstrip utility (2.2h).
```

```
%%
%% The original source files were:
%%
%% fullpage.dtx (with options: 'package')
%% This is a stripped version of the original file.
\NeedsTeXFormat{LaTeX2e}
. . . . .
\addtolength{\topmargin}{-1in}
%% This is the end of stripped file.
```

也可以用一个主批处理作业, 处理单个作业, 此时主作业用 `\batchinput` 输入小作业, 而不能用 `\input`。

去掉文本行的规则

`\generateFile` 命令对输入文件进行转化, 逐行生成输出文件, 所遵从的规则如下:

1. 以单个 % 开始的行都要被去掉。;
2. 而以两个 % 开始的行则保持不变;
3. 对于 `%<opt>` (或 `%<+opt>`) 开始的行, 如果这里的 `opt` 是选定选项中的一个, 就把余下的文本送到输出文件中; 否则就把该行去掉;
4. `%<!opt>` 或 `%<-opt>` 开头的行, 如果 `opt` 不在选定选项中, 则把余下文本送到输出文件中; 否则就把该行去掉;
5. 对于位于 `%<*opt>` 和 `%</opt>` 之间的所有行, 会根据 `opt` 是否是选定的选项, 来确定是否保留这些行。

选项可以进行逻辑组合, 求反以及分组:

```
a&b      ( a and b);
a|b      ( a or b);
!a       ( not a);
(a|b)&c   (c and one of a or b)
```

关于 `DocStrip` 的详情, 可以阅读用 \LaTeX 处理 `docstrip.dtx` 后得到的文档。

§D.3.2 建立 \LaTeX 代码的档案

为软件产品准备充分的档案是相当重要的, 其一方面给用户提供了使用手册, 另一方面也给出了程序员开发代码的细节。在起初的 \LaTeX 样式, 以及基本的 `latex.tex` 文件中, Leslie Lamport 都给出了相当丰富的注释, 但其中只是明了的普通文本。而 Frank Mittelbach 的 `doc` 宏包才第一次可以给源代码生成复杂的, 集成化的档案。

所谓集成化档案,就是指在单个源文件中描述性语言与代码很好地揉合在一起。因此需要进行两次处理,一次提取出真正的代码(D.3.1 节的 DocStrip 工具),另一次显示档案(doc 宏包)。这个宏包的基本想法为注释实际上也是通常的 L^AT_EX 文本,它具有某些额外的功能,可以自动索引,并跟踪程序执行的记录。其自身的代码用一种特殊的 `verbatim` 环境形式表示。

档案是通过用 L^AT_EX 运行一个特殊的驱动程序来生成的。相对于一个名为 `fullpage.dtx` 的源文件,该驱动程序的最简形式应为

```
\documentclass{article}
\usepackage{doc}
\begin{document}
  \DocInput{fullpage.dtx}
\end{document}
```

`\DocInput` 命令读入指定的文件,但它首先把 `%` 字符的功能从“注释”变为“什么也不做”。这也就是说在 `fullpage.dtx` 中所有的注释行都变成要被处理的实际文本了。

原来,源文件的后缀为 `.doc`,它是一个 `.sty` 文件,但用了特殊的 `doc` 宏包命令。我们可以把该文件重命名为 `.sty`,作为宏包文件使用。随着 L^AT_EX 2_ε 标准的出现, `doc.sty` 和 DocStrip 成为基本安装中的一部分,这时不能再任选扩展名了。现在源文件标志为 `.dtx`,充当档案驱动文件角色。也就是说我们只需要用 L^AT_EX 处理这个文件就可以得到档案。在可以使用宏包 `.sty` 文件之前,必须用 DocStrip 从源文件中提取出来。

我们下面相应于列在 C.3.1 节中的 `fullpage` 宏包,用 `fullpage.dtx` 源文件为例,来演示 `doc` 宏包是如何使得 `.dtx` 文件中的“注释”成为有用文本等功能。与其它情形一样,通过处理 `doc.dtx` 可以得到一个相当完整的使用手册。

描述部分

档案由两部分组成: 描述 就是提供给用户的手册, 代码 就是软件产品工作原理的详细介绍,由代码行组成。可以通过在导言中调用下述命令来抑制代码部分,只显示出描述部分:

```
\OnlyDescription
```

在描述部分可以使用的特殊命令有:

```
\DescribeMacro{\宏名}
```

```
\DescribeEnvironment{环境名}
```

它们要放在文本开头,演示新的高级命令(宏)或环境。这两条命令做两件事:把宏或环境的名称放在该处的页边注中(阅读时便于检索),并在索引中插入一项条目。

由于在档案中经常要用到 \verb 命令以显示源文本，因此可以用下面的命令给出其缩写形式：

$\text{\MakeShortVerb{\ 字符}}$ 和 $\text{\DeleteShortVerb{\ 字符}}$

其首先把给定字符变成 \verb 字符的简写，然后再恢复其原来用法。例如，在调用了 $\text{\MakeShortVerb{\|}}$ 之后， \|mycom| 就会显示出 \mycom 。（上载了 shortverb 宏包后，在任何文档中都可以使用这两条命令，这个宏包实际上就是从 doc 宏包提取出来的。）

如果已经关闭了注释字符 $\%$ 的功能，那么该如何向档案文本中添加注释呢？一种方法就是利用 \TeX 的条件语句 \iffalse 构成一个模块，即块注释，例如：

```
% \iffalse
%   These lines are ignored even when the
%   percent character is inactive
% \fi
```

另一种方法就是用 \^A 代替 $\%$ ，这也是 doc 的一项特殊功能。

描述部分是用下述命令标志结束的：

$\text{\StopEventually{结束文本}}$

其中结束文本要位于文章的尾部；如果只显示出描述部分，那么就会紧接着显示结束文本，然后结束档案。

代码部分

在代码部分应该包含的是更专门的材料，平常的用户不会对它有多大的兴趣。此处可以使用的特殊命令有

```
\begin{macro}{\ 宏名} 文本与代码 \end{macro}
\begin{environment}{\ 环境名} 文本与代码 \end{environment}
```

这两条命令也都会插入一条页边注，并在索引中生成一项条目。它们也组织可能存在的 \changes 命令，见下面的介绍。

在代码部分最重要的环境是 macrocode ，它以 verbatim 形式显示其内容，并且可以选择是否加上代码行编号。这个环境的形式多少有点儿特别：

```
UUUU\begin{macrocode}
代码行
UUUU\end{macrocode}
```

这里在 $\text{\end{macrocode}}$ 前面的四个空格是必需的；而 \begin 前面的空格则是可有可无，但为了对称起见，最好还是加上。这一环境会统计其内部所有反斜杠的数目，供后面校验和测试使用，为找到的每条命令名称生成一项索引。因此它并单单只是一个 verbatim 环境！

代码部分的结尾部分应该是

\Finale

这条命令进行校验和测试，显示来自于 `\StopEventually` 中保存的 结束文本。有可能在其后还有其它文本，只有当描述部分和代码部分都输出后才会显示它们。

宏的索引和修改的记录

在 `doc` 宏包中是通过上面列出的两条 `\Describexxx` 命令和两个环境，在索引中自动插入条目。另外，所有出现的命令都要生成索引。然而，只有在导言中使用了

\CodelineIndex 或者 \PageIndex

中一条时，索引功能才会有效。其中第一条命令用所位于的代码行编号来引用被索引命令，而第二行命令用的则是页码。对于后者，代码行没有编号，除非使用了 `\CodelineNumbered` 声明。

由于在代码中并不是所有命令都需要进行索引，如果处理的是标准 L^AT_EX 和 T_EX 部分命令时更时如此，命令

\DoNotIndex{命令名称清单}

通常放在开头部分，而且可以重复使用，以剔除不需要生成索引的命令。

对代码中的命令自动生成索引，会显著减慢处理速度。一旦生成了索引文件 `.idx`，那么后面的运行就不必再做这件事了（除非对代码又进行了修改）。命令

\DisableCrossrefs

会关闭生成索引的功能，但它可以被前面的 `\EnableCrossrefs` 否决，后一条命令就是抑制关闭命令的。

`MakeIndex` 程序 (8.4 节) 根据 `.idx` 数据文件，生成索引文本，为此必须用特殊的索引样式 `gind.ist` 执行：

```
makeindex -s gind.ist 文件名
```

索引样式文件 `gind.ist` 是从 `doc.dtx` 中提取出来的。

为了显示索引，命令

\PrintIndex

就放在希望索引所处的地方。其通常是 `\StopEventually` 中 结束文本 的一部分。只有在两次 L^AT_EX 处理之间运行了 `MakeIndex` 程序，才能得到一个更新后的索引。

对软件修改的记录可以通过在档案任何地方插入下述命令来生成：

\changes{版本}{日期}{文本}

插入的地方既可以是描述部分，也可以是代码部分。为了生成修改历史清单，可以调用命令

\RecordChanges

它必须放在导言中。这就使得修改条目放在汇总文件中, 然后可以用 `MakeIndex` 进行处理:

```
makeindex -s gglo.ist -o 文件名.gls 文件名.glo
```

(索引样式文件 `gglo.ist` 也是从 `doc.dtx` 中提取出来的。) 这样修改历史就会显示在档案中

```
\PrintChanges
```

所位的地方, 它通常也是 结束文本 的一部分。在 `\changes` 命令中的文本是有顺序的, 首先是版本号, 然后是其所处的宏或环境的名称,

测试完整性

如果要把源文件送到电子网络上, 那么其就有可能被破坏或截短。为此可以进行两项测试。在档案靠近开头的地方 (必须是在代码部分前面), 放上

```
\Checksum{数}
```

这样就会统计在 `macrocode` 环境中所有的反斜杠总数, 然后 `\Finale` 命令就会把所得总和与这里给定的 数 进行比较。如果 数 = 0, 那么真正的总和就会显示在终端上; 否则, 如果总和不等于 数, 就会显示出一条错误消息, 同时给出这两个值。

另外一个测试检查如果经过用不同字符集下的计算机系统传输后, 字符集是否被破坏。

```
\CharacterTable
```

```
{Upper-case \A\B\C\D\E\F\G\H. . .
```

```
. . . . .
```

```
. . . . . Tilde ~}
```

这里的参数值必须精确地与 `doc` 所要求的一样 (除有作用的 % 和多重空格外)。应当直接从 `doc.sty` 或者 `doc.dtx` 文件中复制这部分内容。

获取文件信息

在 `doc` 的 $\LaTeX 2_{\epsilon}$ 版本中有一个新功能, 那就是命令

```
\GetFileInfo{文件名}
```

这条命令要利用在 `\Providesxxx` 命令中给出的区别指定文件的可省发行信息来定义 `\filename`, `\filedate`, `\fileversion` 和 `\fileinfo`(C.2.1 节)。想法就是在 `.dtx` 文件中这些信息应该出现一次, 而且只需要一次, 但我们应该能知道它, 以显示在文章的标题中。这些 `\filexxx` 命令就可以实现此目的。发行信息的形式必须为 日期, 空白, 版本, 空白, 文本。

ltxdoc 类

有一个名为 `ltxdoc` 的特殊类可用来帮助运行 `doc` 宏包。它用 `doc` 宏包启动 `article` 类, 然后调用下述命令

```

\AtBeginDocument{\MakeShortVerb{\|}}
\CodelineNumbered
\DisableCrossrefs

```

并定义其它一些有用的命令，以帮助建立档案。详情见处理 `ltxdoc.dtx` 后的描述部分。它也有局部配置文件：如果有 `ltxdoc.cfg`，就把它读入进来。这可以把纸张尺寸或其它格式化选项传递给 `article`，用 `\AtBeginDocument` 方法调用 `\OnlyDescription`，等等。

.dtx 示例文件

为了说明上述功能的用法，我们下面列出 `fullpage.dtx` 源文件的一部分。开始几行确认的是在接受了恰当识别命令后，所有能从其中提取出来的文件（`.sty` 宏包和 `.drv` 档案驱动程序）。日期和版本信息只需要出现一次，但会送到所有提取出来的文件中。

```

% \iffalse      (This is a meta-comment)
%% Copyright (c) 1994 Patrick W. Daly
\NeedsTeXFormat{LaTeX2e}
%<*dtx>
\ProvidesFile      {fullpage.dtx}
%</dtx>
%<package>\ProvidesPackage{fullpage}
%<driver>\ProvidesFile{fullpage.drv}
% \fi
%\ProvidesFile{fullpage}
          [1994/02/15 1.0 (PWD)]

```

最后那条 `\ProvidesFile{fullpage}` 命令使得 `\GetFileInfo` 可以具有正确的作用；而前一条同样的命令没有任何内容，只是吸收 `.dtx` 文件被直接读入时的信息行。

接下来，给出驱动程序部分。当用 L^AT_EX 直接处理文件时，这是它所见到的内容。

```

%\iffalse
%<*driver>
\documentclass[a4paper,11pt]{article}
\usepackage{doc}
\EnableCrossrefs
\RecordChanges
\CodelineIndex
\begin{document}

```

```

\DocInput{fullpage.dtx}
\end{document}
%</driver>
%\fi

```

下面给出校验和以及不用给出索引的命令清单，然后就是文章的开始。

```

% \Checksum{73}
% \DoNotIndex{\addtolength,\boolean,\ExecuteOptions,\ifthenelse}
% \DoNotIndex{\newboolean,\newlength,\ProcessOptions}
% \DoNotIndex{\RequirePackage,\setboolean,\setlength}
% \changes{1.0}{1994 Feb 15}{Initial version}

% \GetFileInfo{fullpage}
% \title{\bfseries A Package to Set Margins to Full Page}
% \author{Patrick W. Daly}
% \date{This paper describes package \texttt{\filename}\\
%       version \fileversion, from \filedate}
% \maketitle
% \MakeShortVerb{\|}
%
% \section{Purpose}
% To set a uniform margin of one inch or 1.5~cm on all four
% . . . . .

```

下面跳到描述部分的结尾，代码部分的开头。

```

% \StopEventually{\PrintIndex\PrintChanges}
%
% \section{The Coding}
% The first thing is to read in the \texttt{ifthen} package,
% if it is not already there.
% \begin{macrocode}
%<*package>
\RequirePackage{ifthen}
% \end{macrocode}
%
% \begin{macro}{\DeclareOption}
% \begin{macro}{\FP@margin}
% Define the options with help of the length |\FP@margin|. The
% options |in| and |cm| select the actual margin size.

```

```
% \begin{macrocode}
\newlength{\FP@margin}
\DeclareOption{in}{\setlength{\FP@margin}{1in}}
\DeclareOption{cm}{\setlength{\FP@margin}{1.5cm}}
% \end{macrocode}
% \end{macro}
```

在文件的结尾部分结束了代码部分，并调用 `\Finale`。

```
. . . . .
\setlength{\topmargin}{\FP@margin}
\addtolength{\topmargin}{-1in}
%</package>
% \end{macrocode}
% \end{macro}\end{macro}
% \Finale
```

§D.3.3 图形与彩色

正如在 D.2 节指出的那样，T_EX 的 PostScript 驱动程序的出现，使得我们可以扩展 L^AT_EX 的能力，增加新的功能，诸如对外部图形文件的引入，放缩和旋转，以及彩色。由于这些驱动程序中的每一个都倾向于有自己的宏包和用户接口命令，因此我们必须把 L^AT_EX 文档与某一个特定的驱动程序紧密关联在一起。

`graphics` 和 `color` 宏包为所有的驱动程序提供了一组相当一致的命令，而专门化的代码是放在特定的 `.def` 文件中，可以用这些宏包的选项来上载。因此只需要改变一下选项，就可以切换到另一个驱动程序；而正文并不需要进行修改。

它们所定义的命令可以是其它一些宏包的构造模块，这些新的宏包可以提供（一定程度上的）更轻松自在的语法，以实现上述功能。只要这些新的宏包是基于 `graphics` 和 `color` 的，那么它们就应当同样与所有支持的驱动程序兼容。

可以在 CTAN 服务器的 `graphics` 目录 (353 页上的图表) 中找到这些宏包。它们是由 Sebastian Rahtz 和 David Carlisle 提供的。

现在支持的驱动程序 (截止 1994 年) 有

<code>dvi2ps</code>	<code>dvipsone</code>	<code>dviwindo</code>	<code>textures</code>
<code>dvialw</code>	<code>dvitops</code>	<code>emtex</code>	
<code>dvilaser</code>	<code>dvitps</code>	<code>oztex</code>	
<code>dvips</code>	<code>dviwin</code>	<code>pubps</code>	

但是只有 `dvips` 和 `dvipsone` 才具有完备的功能，并且经过了测试。而其它

一些驱动程序可能有引入功能，但是也就只此而已。现在这一切还都在发展过程中，因此要想知道最新的情形，应查阅当前的资料。

有两个宏包可以使用，一个是比较基本的 `graphics`，另一个则是扩展了的 `graphicx`，它上载并使用前者的宏。独立的 `color` 宏包提供了处理颜色的宏包。这三个宏包上载时都可以有一个驱动程序名称做为选项，例如，

```
\usepackage[dvips]{graphics,color}
```

可以通过 `graphics.cfg` 和 `color.cfg` 文件进行局部配置，因为这两个文件如果存在，是会被上述调用读入的。

除了驱动程序名称外，上载 `graphics` 宏包时还可以使用其它一些选项：

`draft`(用 `final` 反过来) 使得不必真的进行引入，而只是留下空白；

`hidescale` 在要进行放缩的地方留下空白；

`hiderotate` 在要进行旋转的地方留下空白。

可以用这些选项得到一个草稿，尤其是如果预览程序不支持图形功能时更要如此。与此类似，`color` 宏包也识别选项

```
monochrome
```

为了校样的需要，它把所有的彩色命令转化成黑白色。

引入外部图形

我们的问题就是把其它某个程序生成的图形文件包含到文档中，做为插图使用。有可能需要对其进行放缩或者旋转 90° 。我们希望要利用剪刀和胶水进行的操作改由计算机实现。

基本命令为

```
\includegraphics[llx,ly][urx,ury]{文件名}
```

其中 `llx`, `ly` 是包围引入图形的 包络盒 的左下角坐标，而 `urx`, `ury` 则是其右上角坐标。也就是说，它们确定的是下剪刀的地方。可以给定单位 (如 `[3cm,2in]`)，可是如果不给定单位，就假定是大点 (bp，每英寸 72 bp)。如果只给出了一个可省参数，那它就是指右上角，而左下角假定为 `[0,0]`。

如果没有给出包络盒的坐标，那么驱动程序要从其它途径来获取该信息，具体视图形文件的类型而定。例如，对于非常流行的 *encapsulated PostScript* 文件 (扩展名为 `.eps`)，包络盒信息就是从图形文件自身提取出来的。

利用 `\includegraphics*`，可以对图形进行剪切，只有指定范围的内容被包含进来。

放缩

在引入图形时，用的是其本来大小。为了放缩图形，可以用下面两条命令：

```
\scalebox{h_scale}[v_scale]{文本}
```

这条命令对 文本 内容应用水平和竖直放缩因子；如果没有给出 `v_scale`，那么它就等于 `h_scale`；

```
\resizebox{h_length}{v_length}{文本}
```

调整插图，使之具有给定的水平和竖直尺寸；如果有一个长度为！，那么要对两个尺寸进行同样的放缩。星号形式的命令可以使 `v_length` 表示盒子的高度与深度之和，而不仅仅只是高度。对于上述两条命令，文本 内容都可以是 `\includegraphics` 命令。

反射

盒子中的内容可以用下述命令进行水平反射：

```
\reflectbox{文本}
```

旋转

盒子中的内容相对于基线左端点进行旋转，采用的命令为

```
\rotatebox{角}{文本}
```

其中 角 以度数为单位，而旋转是反时针方向的。

graphicx 宏包

如果我们选用的是 `graphicx`，而不是 `graphics` 宏包，那么在引入和旋转方面就可以用不同的接口。

```
\includegraphics[bb=llx lly urx ury, \angle=角, width=h_length,
height=v_length, scale=因子, clip=true/false,
draft=true/false]{文件名}
```

其中的关键词顺序是无关紧要的，而且也不需要给出所有的。其绝大多数的含义是自明的，只是 `clip=` 等于 `<true>`，表示要进行剪切；而 `draft` 意味着并不真的引入文件，而是为其留下适当的空白。也可以用命令的星号形式，这时 `clip` 的值为 `<true>`。

`\rotatebox` 也用这些关键词进行了类似地重定义。

rotating 宏包

由 Sebastian Rahtz 和 Leonor Barroca 设计的 `rotating` 宏包尝试给出在某些程度上相对简化了的进行旋转的接口。它定义了

```
\begin{sideways} 文本 \end{sideways}
\begin{turn}{角} 文本 \end{turn}
\begin{rotate}{角} 文本 \end{rotate}
\turnbox{角}{文本}
```


这里 `sideways` 把文本旋转 90° ，`turn` 则可以旋转任意角度。`rotate` 环境与 `\turnbox` 命令是等价的：进行了旋转，但是所处的盒子为零尺寸，这样其内容会与周围文本重叠。

epsfig 宏包

由 Sebastian Rahtz 所开发的 `epsfig` 宏包，不但更新了原来的 (2.09) 版本，而且利用 `graphics` 命令重新实现了 Rokicki 的 `epsf` 宏包。这对于那些习惯于其语法的用户是非常有用的。在 `epsf` 中，语法为

```
\epsfysize=y_size 或 \epsfxsize=x_size
```

```
\epsf[llx lly urx ury]{文件名}
```

如果没有给出包络盒坐标，那就从引入文件中提取相应信息。

在 `epsfig` 宏包中定义的引入命令，也是利用关键词来输入参数：

```
\epsfig{file=文件名, height=高度, width=宽度, clip=, silent=,
        rotate=角, bblx=llx, bblly=lly, bburx=urx, bbury=ury}
```

为了与旧版本兼容，还有一条 `\psfig` 命令，它与上面的命令含义相同。缺省情形中 `epsfig` 宏包自动调用 `dvips` 驱动程序。

上述不同形式的旋转和引入命令表明现在没有一个定形的标准语法；然而，所有上述命令的基础都是 `graphics` 宏包，因此其应当在所有的驱动程序中都会正常工作。或者至少说，它们不会使驱动程序崩溃。

颜色

颜色的指定可以采用已定义的名称，也可以用如下形式

```
[模型]{定义}
```

其中 模型 可以取的值为 `rgb`(红 (red), 绿 (green), 蓝 (blue)), `cmk`(青 (cyan), 品红 (magenta), 黄 (yellow), 黑 (black)), `gray`, 或者 `named`。而 定义 则是一串从 0 到 1 的数，表示模型中每个分量的大小。因此 `[rgb]{1,0,0}` 定义的是红色，`[cmk]{0,0,1,0}` 定义的是黄色。而 `gray` 模型只有一个值。`named` 模型是用于可识别颜色内部名称的驱动程序，例如 `dvips` 知道 68 种颜色。打开 `dvips.def`，就会知道有哪些名称可以使用。

一种颜色可以如下定义

```
\definecolor{名称}{模型}{定义}
```

这样 名称 就可以用到下面所有的颜色命令中。对所有的驱动程序，有些颜色是自动预定义好了的：`red`, `green`, `blue`, `yellow`, `cyan`, `magenta`, `black`, `white`。

在下面的颜色命令中，颜色定义就是某一定义好的颜色，如 `{blue}`，或者 `[模型]{定义}`，如 `[rgb]{0,1,0}`。

`\pagecolor` 颜色定义 为当前页和后续页设置背景颜色；

`\color` 颜色定义 为一个声明, 把后续文本设置成给定颜色;
`\textcolor` 颜色定义{文本} 用给定颜色设置其 文本 参数;
`\colorbox` 颜色定义{文本} 把参数放到一个盒子中, 并以给定颜色做为其背景;
`\fcolorbox` 颜色定义 1 颜色定义 2{文本} 类似于 `\colorbox`, 只是用 颜色定义 1 做为框线的颜色, 包围背景色为 颜色定义 2 的盒子; 这两个颜色定义必须同为名称, 或者同样的模型, 对这后一种情形, 只需给出一次模型名称;
`\normalcolor` 切换到在导言结束时激活的颜色。因此在导言中用 `\color` 颜色命令, 可以改变整篇文档的标准颜色。其等价于字体选择中的命令 `\normalfont`。

§D.3.4 其它有用的宏包

L^AT_EX3 项目组的成员个人也编写了许多宏包, 为了便于使用, 这些宏包收集在一个名为 `tools` 的特殊目录中 (见 353 上的图表)。其中绝大多数是从 L^AT_EX2.09 时代就开始创办的, 现在已更新到了 L^AT_EX2_ε。

我们这里扼要介绍一下其中某些宏包能做什么事; 要想详细了解, 只要用 L^AT_EX 处理其 `.dtx` 文件就可以了。

`array` 是对标准 L^AT_EX 中 `tabular` 和 `array` 环境 (4.8.1 节) 的重新实现, 增加了许多功能。除了等价于 `\parbox[t]{宽度}` 的列定义 `p{宽度}`, 还有 `m{宽度}` (类似于 `\parbox{宽度}`) 和 `b{宽度}` (类似于 `\parbox[b]{宽度}`), 这三个定义都用给定的 宽度 做为列的宽度, 只是各列相对于顶行, 中间和底行对齐。也可以用 `>{声明}` 和 `<{声明}` 在一列的开始和结尾处插入声明, 可以用这种方式来包含一个字体声明, 作用于列中所有行, 或者在数学模式中进出。也可以定义用户自己的列类型。

`dcolumn` 需要 `array` 宏包, 在 `tabular` 表格中可以用小数点对齐。

`delarray` 需要 `array` 宏包, 能在 `array` 环境外面加上大的括号符号, 这样可以生成各种形式的矩阵。

`hhline` 需要 `array` 宏包, 在表格的水平直线输入方面更富有弹性。

`longtable` (不需要 `array` 宏包, 但认识其功能) 使我们可以创建长达几页的表格, 中间自动进行分页。

`tabularx` 定义了 `tabularx` 环境, 其类似于 `tabular*`, 生成具有希望宽度的表格, 但只是伸展列宽, 而不是伸展列间距。

`afterpage` 允许存贮代码, 并插入在当前页的尾部。

`enumerate` 重新实现了 `enumerate` 环境, 可以用一个可省参数值确定每个 `\item` 生成数字的样式。

`fileerr.dtx` 解开这个文件, 会生成一组小文件, 可以用来回答 T_EX 中文件

不存在的情形；它使得反应类似于通常的错误信息；文件名称为 `h.tex`, `e.tex`, `s.tex` 和 `x.tex`；因此，回答 `x`(回车) 会上载最后那个文件，从而结束输入。

`fontsmpl` 是一个打印希望了解的字体示例的宏包。

`ftnright` 在两列模式中把脚注放在列的右端。

`indentfirst` 对章节的第一段进行缩进，通常并不进行这种操作。

`layout` 定义了命令 `\layout`，它生成当前页面格式的示意图，显示出各个参数的值。

`multicol` 给出了 `multicol` 环境，它是 `\twocolumn` 命令的扩充，把其文本以指定数目的列显示出来，而且在开始和结尾部分不开始新页，各列长度均衡。

`showkeys` 显示出所有的由 `\label` 和 `\bibitem` 定义，并由 `\ref`, `\pageref` 和 `\cite` 使用的交叉索引关键词；它们以边注和行间注释的形式出现，只适用于草稿，以检查交叉索引的情况。

`somedefs` 根据 `\usepackage` 中选项，在宏中只允许定义选定的命令。

`theorem` 推广了 `\newtheorem` 命令，使之成为更富有弹性的“定理型”环境。

`varioref` 定义了 `\vref` 和 `\vpageref`，其类似于 `\ref` 和 `\pageref`，它们会检测被引用对象是否就在前一页，如果是这样的话，就显示出类似于“on the previous page”这样的文本；实际显示的文本可以有两种变化。

`verbatim` 重新实现了 `verbatim` 环境，避免了长文本时的内存溢出；而且还定义了命令 `\verbatiminput`，以输入一个文件，并且按字面显示文本，还有一个 `comment` 环境，以生成源文件中的注释块。

`xr` 利用这个宏包可以用 `\ref` 交叉索引另外文档中的 `\label` 命令。

`xspace` 这个工具可以更正命令名称吞掉后接空格的问题；因此如果命令 `\PS` 的定义为

```
\newcommand{\PS}{\PostScript\xspace}
```

那么 `\PS file` 用法就完全可以，没有必要用 `_` 或 `\{` 结束 `\PS` 命令。

§D.4 \LaTeX 2_ε 的安装

在发行的 \LaTeX 2_ε 安装文件组中有一些操作指南，其中既有一般性的，也有针对特定操作系统的。后者所在文件的扩展名为 `.txt`，而基本名就是 \TeX 供应者的名称，如 `emtex.txt`, `oztex.txt`，而一般性指南放在一个名为 `install.txt` 的文件中。在进行操作之前，应首先仔细阅读这些指南。

所需要的文件，有可能是压缩的也有可能是未压缩的。压缩文件由大量的 `.dtx` 文件组成，其为源文件和档案文件的集成。通过用 `initex` 执行批处理作业文件 `unpack.ins` 可以解开上述文件。这样就会得到必需的 `.cls`, `.sty`,

.clo 和其它文件。其也构造出来基础文件 latex.ltx，在生成格式文件时需要这个文件。

如果你所拥有的文件集中已经有了 latex.ltx，那就是说其已经被解开了。这就非常方便，因为视所用的计算机不同，解压缩操作可以化费 5 分钟到 3 小时的时间。但是另一方面，用网络传递压缩文件就会化费较少的时间。

接下来一步就是用 initex 执行 latex.ltx 文件生成 L^AT_EX 2_ε 格式 latex。然而，在做这之前，要考虑许多结构特征。在处理过程中有多处要读入特定的文件，但是如果有基本名相同，扩展名为 .cfg 的文件存在，就会上载这个文件。这种方法可以让我们按自己的条件或意愿配置最终的格式。可能的配置文件有：

texsys.cfg 提供了为老版本的 T_EX 或者某些独特版本的 T_EX 增加相当与机器有关的调整能力；可以在特定的 .txt 或者 ltdirchk.dtx 文件中找到相应信息。

fonttext.cfg 如果这个文件存在，就会取代 fonttext.ltx 文件而被上载；

其定义了处理时可用的字体；可以在 fontdef.dtx 中找到详情。

fontmath.cfg 如果这个文件存在，就会取代 fontmath.ltx 文件而被上载；

其等价于数学字体中的 fonttext.cfg。

preload.cfg 如果这个文件存在，就会取代 preload.ltx 文件而被上载；其

确定在格式中预先上载哪种字体；可以从 preload.dtx 中提取其它的 preload 文件，其每一个都可以重命名成 .cfg，从而实现它。

hyphen.cfg 如果这个文件存在，就会取代 hyphen.ltx 文件而被上载；其指

定了断词模式以及赋值；默认情形中，在 hyphen.tex 中的模式被上载到第 0 号语言中；在 babel 系统中提供了如此的配置文件 (D.1.4 节)。

一旦建立起了一个配置文件，并且把它放在 T_EX 的读取位置，用 initex 执行 latex.ltx 就会生成格式文件 latex.fmt。这个文件必须放在读取格式文件的位置，其它的类，选项和宏包也要放在 T_EX 读这些文件的地方。最后要定义 L^AT_EX 处理命令，这样可以用 latex 格式调用 T_EX，例如，

```
tex &latex
```

注意：L^AT_EX 2.09 的格式名称为 lplain；因此格式名称自身就揭示了要被激活的 L^AT_EX 版本。

§D.4.1 更新 L^AT_EX

计划一年更新两次 L^AT_EX，分别在 6 月份和 12 月份进行。在这些时间，就必须用新的 .dtx 文件生成 latex.ltx 或者直接得到该文件。在两次更新之间的重要修改是在 ltpatch.ltx 文件中进行补正的，这个文件要被 latex.ltx 文件读入。通过这种方法，我们需要得到补丁文件，并从已有的 latex.ltx 重新生成格式文件。

§D.5 \LaTeX 文件的来源

得到扩展 \LaTeX 软件的最简单方法就是利用各种教育机构支持的网络服务器。这些服务器也提供即时更新的标准 \LaTeX 安装版本, \TeX 扩展, 在各种 PC 上运行 \TeX (和 \LaTeX) 的程序, 不同打印机的驱动程序, 其它语言的断词模式, 等等。它们是 \TeX 和 \LaTeX 衷心用户的无穷宝库。

在美国、英国和德国有 \TeX 和 \LaTeX 的三个主要网络服务器, 其构成了全面的 \TeX 文件网络 (Comprehensive \TeX Archive Network, CTAN)。它们不但随时更新, 而且它们之间还要定期交换信息。这些系统都是基于 Unix 的, 可以用文件传输协议 (File Transfer Protocol, FTP) 访问。

国家	IP 地址	IP 数
美国	<code>pip.shsu.edu</code>	192.92.115.10
英国	<code>ftp.tex.ac.uk</code>	134.151.79.32
德国	<code>ftp.dante.de</code>	129.206.100.192

要想连接到这些地址, 只需要输入 `ftp 地址` 就可以了; 经过几秒钟的时间, 就可以用

```
user anonymous
```

登录, 这时密码为你自己的 FTP 地址, 当然任意文本都是可以的。现在主机就会给出如下提示

```
ftp>
```

表示它已经可以接受 FTP 命令了。用 `bye` 或 `quit` 命令结束登录。

可以用 `dir` 得到目录清单, 这些目录通常还有自己的子目录, 其名称表示其可能包含的内容。INDEX 文件对每个子目录的目的进行了简单的介绍。相应于 \LaTeX 的子目录可以在 `macros/latex` 中找到。

在所有服务器上的目录结构是一样的。图 D.1 表示的是目录树的主要部分。

可以在 `base` 或 `unpacked` 子目录中找到最新版本的 \LaTeX 宏包, 对于后一个目录, 文件已经被解开, `latex.ltx` 是现成的。由其它用户奉献出来的 \LaTeX 扩展和一般样式文件位于子目录 `contrib/supported` 中, 其中每个来源都放在单独一个子目录中。由 \LaTeX 3 项目组提供的扩展可以在 `packages` 子目录中找到, 其中包含 `babel`, `graphics`, `tools` 等等。

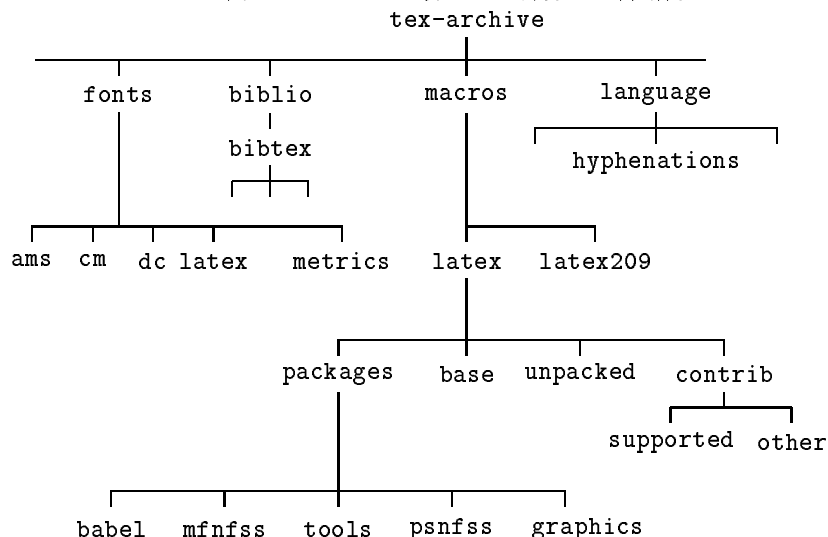
要想把文件复制到自己的计算机上, 可以用 FTP 命令

```
get 文件名 . 扩展名
```

要想一次复制多个文件, 就需要用 `mget` 命令, 而用 `*` 做为文件指定中的通配符。

有许多文件已经用特殊的压缩程序进行了压缩, 因此不能作为 ASCII 文件, 而必须做为二进制文件来传输。可以用输入 FTP 命令 `binary` 来做到这

图 D.1: CTAN 服务器上的部分目录树结构



一转换，调用地方就是在相应的 `get` 或 `mget` 命令之前。对于文本文件，就必须切换回 ASCII，因为 ASCII 文件在二进制模式中通常无法正确地传输。

另外，任一目录的所有内容可以传输到一个存档 / 压缩系统中，方法就是先恰好移到该目录上方，并给出如下调用

<code>get 目录名.tar</code>	相应于 Unix 磁带存储格式
<code>get 目录名.tar.Z</code>	相应于 Unix 的 <code>tar</code> ，压缩后的
<code>get 目录名.tar.gz</code>	相应于 Unix 的 <code>tar</code> ，GNU <code>zip</code>
<code>get 目录名.zip</code>	相应于 <code>zip</code> 压缩
<code>get 目录名.zoo</code>	相应于 <code>zip</code> 存档

例如，`get tools.zip` 会把整个 `tools` 目录作为 `zip` 文件进行传输。自然地，接受者在自己机器上必须有相应的解开程序。

§D.5.1 T_EX组织

有大量的组织致力于 T_EX和 L^AT_EX想法与程序的传播。这些用户组织也为那些无法使用网络的用户提供必须的文件。

可以从 CTAN 文件服务器上的一个名单中得到各种组织的地址。