第一章　The Structure of the TEX Processor
TEX 处理器的结构

This book treats the various aspects of TEX in chapters that are concerned with relatively small, well-delineated, topics. In this chapter, therefore, a global picture of the way TEX operates will be given. Of necessity, many details will be omitted here, but all of these are treated in later chapters. On the other hand, the few examples given in this chapter will be repeated in the appropriate places later on; they are included here to make this chapter self-contained.

本书将以涉及相对小而清晰的主题的章节来介绍 TEX 的各个方面。因此，在本章中，将给出 TEX 运行方式的整体图景。当然，这里有很多细节被省略了，但所有这些细节在后面的章节中都有涉及。另一方面，本章中给出的一些示例将在适当的地方再次出现；它们包含在这里是为了使本章内容自成一体。

1.1　Four TEX processors
四个 TEX 处理器

The way TEX processes its input can be viewed as happening on four levels. One might say that the TEX processor is split into four separate units, each one accepting the output of the previous stage, and delivering the input for the next stage. The input of the first stage is then the .tex input file; the output of the last stage is a .dvi file.

TEX 处理输入的方式可以看作是在四个层次上进行的。可以说 TEX 处理器被分为四个独立的单元，每个单元接受前一阶段的输出，并产生下一阶段的输入。第一阶段的输入是 .tex 输入文件；最后一阶段的输出是一个 .dvi 文件。

For many purposes it is most convenient, and most insightful, to consider these four levels of processing as happening after one another, each one accepting the completed output of the previous level. In reality this is not true: all levels are simultaneously active, and there is interaction between them.

对于许多情况，将这四个处理层次依次考虑是最方便和最有洞察力的，每个层次接受上一层次的完整输出。实际上，这并不完全正确：所有层次都是同时活动的，并且它们之间存在相互作用。

The four levels are (corresponding roughly to the 'eyes', 'mouth', 'stomach', and 'bowels' respectively in Knuth's original terminology) as follows.

这四个层次（与 Knuth 最初的术语“眼睛”、“嘴巴”、“胃”和“肠道”相对应）如下所示：

1. The input processor. This is the piece of TEX that accepts input lines from the file system of whatever computer TEX runs on, and turns them into tokens. Tokens are the internal objects of TEX: there are character tokens that constitute the typeset text, and control sequence tokens that are commands to be processed by the next two levels.
   输入处理器。这是 TEX 的一部分，它从 TEX 运行所在计算机的文件系统接受输入行，并将它们转换为记号。记号是 TEX 的内部对象：其中有构成排版文本的字符记号，还有作为下两个层次处理的命令的控制序列记号。
2. The expansion processor. Some but not all of the tokens generated in the first level – macros, conditionals, and a number of primitive TEX commands – are subject to expansion. Expansion is the process that replaces some (sequences of) tokens by other (or no) tokens.
   展开处理器。在第一层次生成的记号中，一些（但不是全部）– 宏、条件语句和一些原始 TEX 命令 – 可以进行展开。展开是将一些（序列的）记号替换为其他（或不替换）记号的过程。
3. The execution processor. Control sequences that are not expandable are executable, and this execution takes place on the third level of the TEX processor.

执行处理器。不可展开的控制序列是可执行的，并且这个执行过程发生在 TEX 处理器的第三层次上。

One part of the activity here concerns changes to TEX's internal state: assignments (including macro definitions) are typical activities in this category. The other major thing happening on this level is the construction of horizontal, vertical, and mathematical lists.

活动的一部分涉及到 TEX 的内部状态的更改：赋值（包括宏定义）是这一类别中的典型活动。在此层次上发生的另一项主要事务是构建水平、垂直和数学列表。

4. The visual processor. In the final level of processing the visual part of TEX processing is performed. Here horizontal lists are broken into paragraphs, vertical lists are broken into pages, and formulas are built out of math lists. Also the output to the dvi file takes place on this level. The algorithms working here are not accessible to the user, but they can be influenced by a number of parameters.

   视觉处理器。在处理的最后一层中，执行 TEX 的视觉处理。在这里，水平列表被分割为段落，垂直列表被分割为页面，并且数学列表被构建为公式。此外，输出到 dvi 文件也在此层次上进行。在此处工作的算法用户无法访问，但可以通过多个参数进行调整。

1.2 The input processor 输入处理器

The input processor of TEX is that part of TEX that translates whatever characters it gets from the input file into tokens. The output of this processor is a stream of tokens: a token list. Most tokens fall into one of two categories: character tokens and control sequence tokens. The remaining category is that of the parameter tokens; these will not be treated in this chapter.

TEX 的输入处理器是将输入文件中获取的字符转换为记号的 TEX 的一部分。该处理器的输出是一系列记号：一个记号列表。大多数记号属于以下两个类别之一：字符记号和控制序列记号。剩下的类别是参数记号；这些在本章中不进行讨论。

1.2.1 Character input
字符输入

For simple input text, characters are made into character tokens. However, TEX can ignore input characters: a row of spaces in the input is usually equivalent to just one space. Also, TEX itself can insert tokens that do not correspond to any character in the input, for instance the space token at the end of the line, or the \par token after an empty line.

对于简单的输入文本，字符被转换为字符记号。但是，TEX 可以忽略输入字符：输入中的多个空格通常等效为一个空格。此外，TEX 本身可以插入不对应任何输入字符的记号，例如行末的空格记号，或空行后的 \par 记号。

Not all character tokens signify characters to be typeset. Characters fall into sixteen categories – each one specifying a certain function that a character can have – of which only two contain the characters that will be typeset. The other categories contain such characters as {, }, &, and #. A character token can be considered as a pair of numbers: the character code – typically the ascii code – and the category code. It is possible to change the category code that is associated with a particular character code.

并非所有字符记号都表示要排版的字符。字符分为十六个类别 – 每个类别指定字符可以具有的某个功能 – 其中只有两个类别包含将要排版的字符。其他类别包含诸如 {, }, &, 和 # 等字符。字符记号可以被视为一对数：字符码（通常是 ascii 码）和类别码。可以更改与特定字符码相关联的类别码。

When the escape character (by default \ ) appears in the input, TEX's behaviour in forming tokens is more complicated. Basically, TEX builds a control sequence by taking a number of characters from the input and lumping them together into a single token.

当转义字符（默认为 \, ）出现在输入中时，TEX 形成记号的行为更加复杂。基本上，TEX 通过从输入中取出一些字符并将它们组合成一个单独的记号来构建控制序列。

The behaviour with which TEX's input processor reacts to category codes can be described as a machine that switches between three internal states: N, new line; M, middle of line; S, skipping spaces. These states and the transitions

between them are treated in Chapter ??.

TEX 的输入处理器对类别码的反应可以被描述为在三个内部状态之间切换的机器：N，新行；M，行中间；S，跳过空格。这些状态及其之间的转换在第 ?? 章中进行讨论。

1.2.2 Two-level input processing 两级输入处理

TEX's input processor is in fact itself a two-level processor. Because of limitations of the terminal, the editor, or the operating system, the user may not be able to input certain desired characters. Therefore, TEX provides a mechanism to access with two superscript characters all of the available character positions. This may be considered a separate stage of TEX processing, taking place prior to the three-state machine mentioned above.

实际上，TEX 的输入处理器本身就是一个两级处理器。由于终端、编辑器或操作系统的限制，用户可能无法输入某些期望的字符。因此，TEX 提供了一种机制，通过两个上标字符来访问所有可用的字符位置。这可以被视为 TEX 处理的一个单独阶段，在上述三状态机之前进行。

For instance, the sequence ^^+ is replaced by k because the ascii codes of k and + differ by 64. Since this replacement takes place before tokens are formed, writing \vs^^+ip 5cm has the same effect as \vskip 5cm. Examples more useful than this exist.

例如，序列 ^^+ 会被替换为 k，因为 k 和 + 的 ascii 码之差为 64。由于这种替换发生在记号形成之前，编写 \vs^^+ip 5cm 的效果与 \vskip 5cm 相同。存在比这更有用的示例。

Note that this first stage is a transformation from characters to characters, without considering category codes. These come into play only in the second phase of input processing where characters are converted to character tokens by coupling the category code to the character code.

注意，这个第一阶段是从字符到字符的转换，而不考虑类别码。只有在输入处理的第二阶段，字符才通过将类别码与字符码相结合，转换为字符记号时起作

用。

1.3　The expansion processor
展开处理器

TEX's expansion processor accepts a stream of tokens and, if possible, expands the tokens in this stream one by one until only unexpandable tokens remain. Macro expansion is the clearest example of this: if a control sequence is a macro name, it is replaced (together possibly with parameter tokens) by the definition text of the macro.

TEX 的展开处理器接受一系列记号，并在可能的情况下逐个展开这些记号，直到只剩下不可展开的记号为止。宏展开是最明显的例子：如果一个控制序列是宏名，它将被宏的定义文本（可能连同参数记号）替换。

Input for the expansion processor is provided mainly by the input processor. The stream of tokens coming from the first stage of TEX processing is subject to the expansion process, and the result is a stream of unexpandable tokens which is fed to the execution processor.

展开处理器的输入主要由输入处理器提供。来自 TEX 处理的第一阶段的记号流会经过展开处理，结果是一系列不可展开的记号，这些记号会被馈送给执行处理器。

However, the expansion processor comes into play also when (among others) an \edef or \write is processed. The parameter token list of these commands is expanded very much as if the lists had been on the top level, instead of the argument to a command.

然而，展开处理器还在处理（等等）\edef 或 \write 等命令时发挥作用。这些命令的参数记号列表的展开非常类似于列表位于顶层，而不是命令的参数中。

1.3.1　The process of expansion
扩展过程

Expanding a token consists of the following steps:

扩展一个记号 (token) 包括以下步骤：

1. See whether the token is expandable.
   检查记号是否可扩展。
2. If the token is unexpandable, pass it to the token list currently being built, and take on the next token.
   如果记号不可扩展，将其传递给当前正在构建的记号列表，并继续处理下一个记号。
3. If the token is expandable, replace it by its expansion. For macros without parameters, and a few primitive commands such as \jobname, this is indeed a simple replacement. Usually, however, TEX needs to absorb some argument tokens from the stream in order to be able to form the replacement of the current token. For instance, if the token was a macro with parameters, sufficiently many tokens need to be absorbed to form the arguments corresponding to these parameters.
   如果记号可扩展，用其扩展内容替换它。对于没有参数的宏和一些原始命令（如\jobname），这确实是一个简单的替换。然而，通常情况下，TEX 需要从流中吸收一些参数记号，以便能够形成当前记号的替换内容。例如，如果记号是带有参数的宏，需要吸收足够多的记号来形成与这些参数相对应的参数值。
4. Go on expanding, starting with the first token of the expansion.
   继续扩展，从扩展内容的第一个记号开始。

Deciding whether a token is expandable is a simple decision. Macros and active characters, conditionals, and a number of primitive TEX commands (see the list on page ??) are expandable, other tokens are not. Thus the expansion processor replaces macros by their expansion, it evaluates conditionals and eliminates any irrelevant parts of these, but tokens such as \vskip and character tokens, including characters such as dollars and braces, are passed untouched.

决定一个记号是否可展开是一个简单的决策。宏和活动字符、条件语句以及一些原始的 TEX 命令（参见第 ?? 页上的列表）是可展开的，其他的记号则不可展开。因此，展开处理器会用宏的展开部分替换宏，评估条件语句并消除其中的任何无关部分，但是像 \vskip 这样的记号和字符记号，包括美元符号和花括号等字符，会原样传递。

1.3.2 Special cases: \expandafter, \noexpand, and \the
特殊情况：\expandafter、\noexpand 和 \the

As stated above, after a token has been expanded, TEX will start expanding the resulting tokens. At first sight the \expandafter command would seem to be an exception to this rule, because it expands only one step. What actually happens is that the sequence

如上所述，在记号展开后，TEX 将开始展开所得到的记号。乍一看，\expandafter 命令似乎是一个例外，因为它只展开一步。实际上发生的是，序列

\expandafter⟨token$_1$⟩⟨token$_2$⟩

被替换为　　is replaced by

⟨token$_1$⟩⟨expansion of token$_2$⟩

and this replacement is in fact reexamined by the expansion processor.

而这个替换实际上会被展开处理器重新检查。

Real exceptions do exist, however. If the current token is the \noexpand command, the next token is considered for the moment to be unexpandable: it is handled as if it were \relax, and it is passed to the token list being built.

然而，确实存在一些真正的例外情况。如果当前记号是 \noexpand 命令，那么下一个记号在此刻被视为不可展开的：它被处理时就好像它是 \relax，并且它被传递给正在构建的记号列表。

例如，在宏定义中　　For example, in the macro definition

```
\edef\a{\noexpand\b}
```

the replacement text \noexpand\b is expanded at definition time. The expansion of \noexpand is the next token, with a temporary meaning of \relax. Thus, when the expansion processor tackles the next token, the \b, it will consider that to be unexpandable, and just pass it to the token list being built, which is the replacement text of the macro.

替换文本 \noexpand\b 在定义时被展开。\noexpand 的展开是下一个记号，并且临时具有 \relax 的含义。因此，当展开处理器处理下一个记号时，即 \b，它将被视为不可展开的，并且会直接传递给正在构建的记号列表，也就是宏的替换文本。

Another exception is that the tokens resulting from \the⟨token variable⟩ are not expanded further if this statement occurs inside an \edef macro definition.

另一个例外是，由 \the⟨token variable⟩ 产生的记号如果出现在 \edef 宏定义内部，则不会进一步展开。

1.3.3 Braces in the expansion processor 展开处理器中的花括号

Above, it was said that braces are passed as unexpandable character tokens. In general this is true. For instance, the \romannumeral command is handled by the expansion processor; when confronted with

前面提到，花括号作为不可展开的字符记号传递。一般来说，这是正确的。例如，\romannumeral 命令由展开处理器处理；当遇到

\romannumeral1\number\count2 3{4 ...

TEX will expand until the brace is encountered: if \count2 has the value of zero, the result will be the roman numeral representation of 103.

TEX 会展开直到遇到花括号为止：如果 \count2 的值为零，则结果将是数字 103 的罗马数字表示。

As another example,

作为另一个例子，

\iftrue {\else }\fi

is handled by the expansion processor completely analogous to

在展开处理器中的处理方式与

\iftrue a\else b\fi

The result is a character token, independent of its category.

完全类似。结果是一个字符记号，与其类别无关。

However, in the context of macro expansion the expansion processor will recognize braces. First of all, a balanced pair of braces marks off a group of tokens to be passed as one argument. If a macro has an argument

然而，在宏展开的上下文中，展开处理器会识别花括号。首先，一对平衡的花括号将一组记号标记为作为一个参数传递。如果一个宏带有参数，如

\def\macro#1{ ... }

one can call it with a single token, as in

可以使用单个记号调用它，例如

\macro 1 \macro \$

or with a group of tokens, surrounded by braces

或者使用由花括号括起来的记号组

\macro {abc} \macro {d{ef}g}

Secondly, when the arguments for a macro with parameters are read, no expressions with unbalanced braces are accepted. In

其次，在读取带参数的宏的实参时，不接受有不平衡花括号的表达式。在

\def\a#1\stop{ ... }

the argument consists of all tokens up to the first occurrence of \stop that is not in braces: in

中，实参包括从第一个不在花括号中的 \stop 开始的所有记号：在

\a bc{d\stop}e\stop

the argument of \a is bc{d\stop}e. Only balanced expressions are accepted here.

中，\a 的实参是 bc{d\stop}e。这里只接受平衡的表达式。

1.4　The execution processor
执行处理器

The execution processor builds lists: horizontal, vertical, and math lists. Corresponding to these lists, it works in horizontal, vertical, or math mode. Of these three modes ‘internal’ and ‘external’ variants exist. In addition to building lists, this part of the TEX processor also performs mode-independent processing, such as assignments.

执行处理器构建列表：水平列表、垂直列表和数学列表。与这些列表相对应，它可以在水平模式、垂直模式或数学模式下工作。这三种模式都有“内部”和“外部”变体。除了构建列表之外，TEX 处理器的这一部分还执行与模式无关的处理，例如赋值操作。

Coming out of the expansion processor is a stream of unexpandable tokens to be processed by the execution processor. From the point of view of the execution processor, this stream contains two types of tokens:

从展开处理器出来的是一系列不可展开的记号，由执行处理器处理。从执行处理器的角度来看，这个流包含两种类型的记号：

- Tokens signalling an assignment (this includes macro definitions), and other tokens signalling actions that are independent of the mode, such as \show and \aftergroup.
  表示赋值的记号（包括宏定义）和表示与模式无关的动作的其他记号，例如 \show 和 \aftergroup。

- Tokens that build lists: characters, boxes, and glue. The way they are handled depends on the current mode.
  构建列表的记号：字符、盒子和粘连。它们的处理方式取决于当前的模式。

Some objects can be used in any mode; for instance boxes can appear in horizontal, vertical, and math lists. The effect of such an object will of course still depend on the mode. Other objects are specific for one mode. For instance, characters (to be more precise: character tokens of categories 11 and 12), are intimately connected to horizontal mode: if the execution processor is in vertical mode when it encounters a character, it will switch to horizontal mode.

有些对象可以在任何模式下使用；例如盒子可以出现在水平、垂直和数学列表中。当然，这种对象的效果仍然取决于模式。其他对象只适用于特定的模式。例如，字符（更准确地说是类别为 11 和 12 的字符记号）与水平模式密切相关：如果执行处理器在垂直模式遇到字符，它会切换到水平模式。

类别码为 11 的字符被称为 “letter”（字母）。这个类别的字符包括所有的字母（a-z, A-Z）和某些特殊字符，比如 @ 在 LaTeX 中的类别码默认是 11。这些字符可以用于形成控制序列（命令名）的一部分。

类别码为 12 的字符被称为 “other”（其他）。这个类别的字符包括数字、标点符号和大多数特殊字符。这些字符被单独处理，不会形成控制序列的一部分。

助记,letter>l>1; 其他,t>two>2

—virhuiai

Not all character tokens signal characters to be typeset: the execution processor can also encounter math shift characters (by default $) and beginning/end of group characters (by default { and }). Math shift characters let TEX enter or exit math mode, and braces let it enter or exit a new level of grouping.

并非所有的字符记号都表示要排版的字符：执行处理器也可以遇到数学模式切换字符（默认为 $）和组的开始/结束字符（默认为 { and }）。数学切换字符让 TEX 进入或退出数学模式，而花括号则让它进入或退出新的分组级别。

One control sequence handled by the execution processor deserves special mention: \relax. This control sequence is not expandable, but the execution is to do nothing. Compare the effect of \relax in

执行处理器处理的一个控制序列值得特别提及：\relax。这个控制序列不可展开，但执行时不做任何操作。比较一下在以下两种情况下，\relax 的效果：

\count0=1\relax 2

with that of \empty defined by

\def\empty{}

in

\count0=1\empty 2

In the first case the expansion process that is forming the number stops at \relax and the number 1 is assigned; in the second case \empty expands to nothing, so 12 is assigned.

在第一种情况下，正在进行的展开过程在遇到 \relax 时停止，数字 1 被赋值；而在第二种情况下，\empty 展开为空，所以赋值为 12。

1.5 The visual processor 视觉处理器

TEX's output processor encompasses those algorithms that are outside direct user control: paragraph breaking, alignment, page breaking, math typesetting, and dvi file generation. Various parameters control the operation of these parts of TEX.

TEX 的输出处理器包括那些在直接用户控制之外的算法：段落分割、对齐、分页、数学排版和 dvi 文件生成。各种参数控制着这些部分的运行。

Some of these algorithms return their results in a form that can be handled by the execution processor. For instance, a paragraph that has been broken into lines is added to the main vertical list as a sequence of horizontal boxes with intermediate glue and penalties. Also, the page breaking algorithm stores its result in \box255, so output routines can dissect it. On the other hand, a math formula can not be broken into pieces, and, naturally, shipping a box to the dvi file is irreversible.

其中一些算法以可由执行处理器处理的形式返回其结果。例如，已经分割成行的段落被添加到主垂直列表中，作为一系列带有中间粘连和惩罚的水平盒子。此外，分页算法将其结果存储在 \box255 中，以便输出例程可以对其进行解析。另一方面，数学公式无法分割成多个部分，并且将盒子输出到 dvi 文件是不可逆转的。

1.6 Examples

示例

1.6.1 Skipped spaces
跳过的空格

Skipped spaces provide an illustration of the view that TEX's levels of processing accept the completed input of the previous level. Consider the commands

跳过的空格提供了一个说明，即 TEX 的处理层级接受上一层级的完成输入。考虑以下命令：

```
\def\a{\penalty200}
\a 0
```

This is not equivalent to

这与以下代码不/ 等价：

```
\penalty200 0
```

which would place a penalty of 200, and typeset the digit 0. Instead it expands to

后者会放置一个惩罚值为 200，并排版数字 0。相反，它展开为：

```
\penalty2000
```

because the space after \a is skipped in the input processor. Later stages of processing then receive the sequence

因为输入处理器会跳过 \a 后面的空格。然后，处理的后续阶段接收到以下序列：

```
\a0
```

1.6.2 Internal quantities and their representations
内部量及其表示形式

TEX uses various sorts of internal quantities, such as integers and dimensions. These internal quantities have an external representation, which is a string of characters, such as 4711 or 91.44cm.

TEX 使用各种内部量，如整数和尺寸。这些内部量具有外部表示形式，即由字符组成的字符串，例如 4711 或 91.44cm。

Conversions between the internal value and the external representation take place on two different levels, depending on what direction the conversion goes. A string of characters is converted to an internal value in assignments such as

内部值与外部表示形式之间的转换在两个不同的层次上进行，取决于转换的方向。在赋值语句中，字符字符串被转换为内部值，例如

\pageno=12 \baselineskip=13pt

or statements such as

或者在语句中，例如

\vskip 5.71pt

and all of these statements are handled by the execution processor.

所有这些语句都由执行处理器处理。

On the other hand, the conversion of the internal values into a representation as a string of characters is handled by the expansion processor. For instance,

另一方面，将内部值转换为字符字符串的表示形式是由展开处理器处理的。例如

\number\pageno \romannumeral\year
\the\baselineskip

are all processed by expansion.

都由展开处理器处理。

As a final example, suppose \count2=45, and consider the statement

作为最后一个例子，假设 \count2=45，考虑以下语句

\count0=1\number\count2 3

The expansion processor tackles \number\count2 to give the characters 45, and the space after the 2 does not end the number being assigned: it only serves as a delimiter of the number of the \count register. In the next stage of processing, the execution processor will then see the statement

展开处理器处理 \number\count2 以得到字符 45，而 2 后面的空格并不结束正在赋值的数字：它只作为 \count 寄存器的编号的分隔符。在下一阶段的处理中，执行处理器将看到语句

\count0=1453

and execute this.

并执行它。