第三章　Characters
字符

Internally, TEX represents characters by their (integer) character code. This chapter treats those codes, and the commands that have access to them.

在内部，TEX 用它们的（整数）字符码表示字符。本章讨论这些字符码以及能够访问它们的命令。

\char Explicit denotation of a character to be typeset.
显式表示要排版的字符。

\chardef Define a control sequence to be a synonym for a character code.
将控制序列定义为字符码的别名。

\accent Command to place accent characters.
用于放置重音字符的命令。

\if Test equality of character codes.
比较字符码是否相等。

\ifx Test equality of both character and category codes.
比较字符码和类别码是否都相等。

\let Define a control sequence to be a synonym of a token.
将控制序列定义为一个记号的别名。

\uccode Query or set the character code that is the uppercase variant of a given code.
查询或设置给定字符码的大写形式的字符码。

\lccode Query or set the character code that is the lowercase variant of a given code.
查询或设置给定字符码的小写形式的字符码。

\uppercase Convert the ⟨general text⟩ argument to its uppercase form.
将 ⟨general text⟩ 参数转换为大写形式。

\lowercase Convert the ⟨general text⟩ argument to its lowercase form.
将 ⟨general text⟩ 参数转换为小写形式。

\string Convert a token to a string of one or more characters.
将记号转换为一个或多个字符的字符串。

\escapechar Number of the character that is to be used for the escape character when control sequences are being converted into character tokens. IniTEX default: 92 (\).
当将控制序列转换为字符记号时，用于作为转义字符的字符的编号。在初始的 TEX 中，默认值为 92（\）。

3.1 Character codes
字符码

Conceptually it is easiest to think that TEX works with characters internally, but in fact TEX works with integers: the 'character codes'.

从概念上来说，最容易理解的是 TEX 内部使用字符，但实际上 TEX 使用的是整数：即“字符码”。

The way characters are encoded in a computer may differ from system to system. Therefore TEX uses its own scheme of character codes. Any character that is read from a file (or from the user terminal) is converted to a character code according to the character code table. A category code is then assigned based on this (see Chapter ??). The character code table is based on the 7-bit ascii table for numbers under 128 (see Chapter ??).

不同的计算机系统对字符的编码方式可能有所不同。因此，TEX 使用自己的字符码方案。从文件（或用户终端）读取的任何字符都会根据字符码表转换为字符码。然后，根据字符码分配类别码（参见第 ??章）。字符码表基于 7 位 ascii 表，在数字小于 128 时使用（参见第?? 章）。

There is an explicit conversion between characters (better: character tokens) and character codes using the left quote (grave, back quote) character ‘: at all

places where TEX expects a ⟨number⟩ you can use the left quote followed by a character token or a single-character control sequence. Thus both \count`a and \count``\a are synonyms for \count97. See also Chapter ??.

可以使用左引号（反引号）字符 对字符（更准确地说是字符记号）和字符码进行显式转换：在 TEX 需要一个 ⟨number⟩ 的任何地方，都可以使用左引号后跟一个字符记号或一个单字符控制序列。因此，\counta 和 \count``\a 都是 \count97 的同义词。详见第 ?? 章。

The possibility of a single-character control sequence is necessary in certain cases such as

在某些情况下，需要使用单字符控制序列，例如：

\catcode`\%=11 or \def\CommentSign{\char`\%}

which would be misunderstood if the backslash were left out. For instance

如果省略了反斜杠，则会引起误解。例如：

\catcode`%=11

would consider the =11 to be a comment. Single-character control sequences can be formed from characters with any category code.

会将 =11 视为注释。单字符控制序列可以由任何类别码的字符构成。

After the conversion to character codes any connection with external representations has disappeared. Of course, for most characters the visible output will 'equal' the input (that is, an 'a' causes an 'a'). There are exceptions, however, even among the common symbols. In the Computer Modern roman fonts there are no 'less than' and 'greater than' signs, so the input '<>' will give '<>' in the output.

在转换为字符码之后，与外部表示的任何联系都已消失。当然，对于大多数字符，可见的输出将与输入“相等”（即，a' 会输出为 a'）。然而，即使在常见的符号中也存在例外情况。在 Computer Modern Roman 字体中，没有“小于”和“大于”符号，因此输入 <>' 会在输出中显示为 <> '。

In order to make TEX machine independent at the output side, the character codes are also used in the dvi file: opcodes $n = 0 \ldots 127$ denote simply the instruction 'take character n from the current font'. The complete definition of the opcodes in a dvi file can be found in [?].

为了使 TEX 在输出方面具有机器无关性，字符码也用于 dvi 文件：操作码 $n = 0 \ldots 127$ 简单地表示“从当前字体中获取字符 n”。有关 dvi 文件中操作码的完整定义可以在 [?] 中找到。

3.2 Control sequences for characters 用于字符的控制序列

There are a number of ways in which a control sequence can denote a character. The \char command specifies a character to be typeset; the \let command introduces a synonym for a character token, that is, the combination of character code and category code.

有多种方式可以使用控制序列表示一个字符。命令 \char 指定要排版的字符；命令 \let 引入了一个字符记号的同义词，即字符编码和类别码的组合。

3.3 Denoting characters to be typeset: \char 用于表示要排版的字符：\char

Characters can be denoted numerically by, for example, \char98. This command tells TEX to add character number 98 of the current font to the horizontal list currently under construction.

字符可以通过数字进行表示，例如 \char98。此命令告诉 TEX 在当前正在构建的水平列表中添加当前字体的第 98 号字符。

Instead of decimal notation, it is often more convenient to use octal or hexadecimal notation. For octal the single quote is used: \char'142; hexadecimal uses the double quote: \char”62. Note that \char''62 is incorrect; the process that replaces two quotes by a double quote works at a later stage of processing (the visual processor) than number scanning (the execution processor).

除了十进制表示法，使用八进制或十六进制表示法通常更方便。八进制使用单引号：\char'142; 十六进制使用双引号：\char”62. 请注意，\char''62 是错误的；

将两个引号替换为一个双引号的过程发生在处理的较晚阶段（视觉处理器）而不是数字扫描阶段（执行处理器）。

Because of the explicit conversion to character codes by the back quote character it is also possible to get a 'b' – provided that you are using a font organized a bit like the ascii table – with \char`b or \char`\b.

由于反引号字符对字符编码进行了显式转换，因此也可以获取一个 b' – 假设您使用的字体的组织方式有点类似于 ascii 表 – 使用 \charb 或 \char`\b。

The \char command looks superficially a bit like the ^^ substitution mechanism (Chapter ??). Both mechanisms access characters without directly denoting them. However, the ^^ mechanism operates in a very early stage of processing (in the input processor of TEX, but before category code assignment); the \char command, on the other hand, comes in the final stages of processing. In effect it says 'typeset character number so-and-so'.

\char 命令在表面上看起来有点类似于 ^^ 替换机制（第 ?? 章）。这两种机制都是在不直接表示字符的情况下访问字符。然而，^^ 机制在处理的早期阶段（在 TEX 的输入处理器中，在类别码分配之前）进行操作；另一方面，\char 命令在处理的最后阶段出现。实际上，它的作用是“排版字符编号为某某的字符”。

There is a construction to let a control sequence stand for some character code: the command. The syntax of this is

还有一种构造可以让一个控制序列代表某个字符编码，即 命令。它的语法是

\chardef⟨control sequence⟩⟨equals⟩⟨number⟩,

where the number can be an explicit representation or a counter value, but it can also be a character code obtained using the left quote command (see above; the full definition of ⟨number⟩ is given in Chapter ??). In the plain format the latter possibility is used in definitions such as

其中数字可以是显式表示或计数器值，也可以是使用左引号命令获得的字符编码（参见上文；⟨number⟩ 的完整定义在第 ?? 章中给出）。在 plain 格式中，可以使用后一种可能性进行定义，例如：

\chardef\%=`\%

which could have been given equivalently as

这也可以等价地写为：

\chardef\%=37

After this command, the control symbol \% used on its own is a synonym for \char37, that is, the command to typeset character 37 (usually the per cent character).

执行此命令后，单独使用的控制符号 % 是 \char37 的同义词，即用于排版字符 37 的命令（通常是百分号字符）。

A control sequence that has been defined with a \chardef command can also be used as a ⟨number⟩. This fact is used in allocation commands such as \newbox (see Chapters ?? and ??). Tokens defined with \mathchardef can also be used this way.

使用 \chardef 命令定义的控制序列也可以用作 ⟨number⟩。在分配命令（如 \newbox）中使用了这个特性（见第 ?? 和第 ?? 章）。使用 \mathchardef 定义的记号也可以以这种方式使用。

3.3.1 Implicit character tokens: \let
隐式字符记号：\let

Another construction defining a control sequence to stand for (among other things) a character is \let:

另一种定义控制序列代表（除其他外）一个字符的构造是 \let：

\let⟨control sequence⟩⟨equals⟩⟨token⟩

with a character token on the right hand side of the (optional) equals sign. The result is called an implicit character token. (See page ?? for a further discussion of \let.)

其中等号（可选）右侧是一个字符记号。结果被称为隐式字符记号。（有关 \let 的进一步讨论，请参见第 ?? 页。）

In the plain format there are for instance synonyms for the open and close brace:

在 plain 格式中，例如对于左花括号和右花括号有以下同义词：

```
\let\bgroup={ \let\egroup=}
```

The resulting control sequences are called ‘implicit braces’ (see Chapter ??).

由此产生的控制序列被称为“隐式花括号”（参见第 ?? 章）。

Assigning characters by \let is different from defining control sequences by \chardef, in the sense that \let makes the control sequence stand for the combination of a character code and category code.

通过 \let 分配字符与通过 \chardef 定义控制序列不同，因为 \let 使得控制序列代表字符代码和类别码的组合。

As an example 例如，

```
\catcode`|=2 % make the bar an end of group
\let\b=|  % make \b a bar character
{\def\m{...}\b \m
```

gives an ‘undefined control sequence \m’ because the \b closed the group inside which \m was defined. On the other hand,

会报告“未定义的控制序列 \m”，因为 \b 关闭了 \m 定义所在的组。另一方面，

```
\let\b=| % make \b a bar character
\catcode`|=2  % make the bar character end of group
{\def\m{...}\b \m
```

leaves one group open, and it prints a vertical bar (or whatever is in position 124 of the current font). The first of these examples implies that even when the braces have been redefined (for instance into active characters for macros that format C code) the beginning-of-group and end-of-group functionality is available through the control sequences \bgroup and \egroup.

会留下一个未关闭的组，并打印一个竖线（或者当前字体位置的字符 124）。这两个示例表明，即使花括号已被重新定义（例如作为用于格式化 C 代码的宏的活动字符），组的开始和结束功能仍可通过控制序列 \bgroup 和 \egroup 实现。

Here is another example to show that implicit character tokens are hard to distinguish from real character tokens. After the above sequence

以下是另一个示例，以展示隐式字符记号很难与实际字符记号区分。在执行了上述代码之后，

```
\catcode`|=2 \let\b=|
the tests
\if\b|
and
\ifcat\b}
```

are both true.

两者都为真。

Yet another example can be found in the plain format: the commands

在 plain 格式中还有另一个示例：命令

```
\let\sp=^ \let\sb=_
```

allow people without an underscore or circumflex on their keyboard to make sub- and superscripts in mathematics. For instance:

允许没有下划线或插入符号的键盘的用户在数学公式中添加上标和下标。例如：

```
x\sp2\sb{ij}
```

gives x^2_{ij}

If a person typing in the format itself does not have these keys, some further tricks are needed:

如果在格式本身中输入的人没有这些键，就需要使用一些额外的技巧：

```
{\lccode`,=94 \lccode`.=95 \catcode`,=7 \catcode`.=8
\lowercase{\global\let\sp=, \global\let\sb=.}}
```

will do the job; see below for an explanation of lowercase codes. The ^^ method as it was in TEX version 2 (see page ??) cannot be used here, as it would require typing two characters that can ordinarily not be input. With the extension in TEX version 3 it would also be possible to write

这样就可以实现；有关小写代码的解释，请参见下文。在 TEX 版本 2 中的 ^^ 方法（参见第 ?? 页）无法在此处使用，因为它需要输入通常无法输入的两个字符。在 TEX 版本 3 中的扩展中，也可以写成：

```
{\catcode`\,=7
\global\let\sp=,,5e \global\let\sb=,,5f}
```

denoting the codes 94 and 95 hexadecimally.

其中分别表示十六进制代码 94 和 95。

Finding out just what a control sequence has been defined to be with \let can be done using \meaning: the sequence

通过 \let 找出控制序列的定义可以使用 \meaning：序列

\let\x=3 \meaning\x

gives 'the character 3'.

3.4 Accents
重音符号

Accents can be placed by the ⟨horizontal command⟩ \accent:

重音符号可以通过 ⟨ 水平命令 ⟩ \accent 来放置：

\accent⟨8-bit number⟩⟨optional assignments⟩⟨character⟩

where ⟨character⟩ is a character of category 11 or 12, a \char⟨8-bit number⟩ command, or a \chardef token. If none of these four types of ⟨character⟩ follows, the accent is taken to be a \char command itself; this gives an accent 'suspended in mid-air'. Otherwise the accent is placed on top of the following character. Font changes between the accent and the character can be effected by the ⟨optional assignments⟩.

其中 ⟨ 字符 ⟩ 可以是类别码为 11 或 12 的字符，一个\char⟨8 位数 ⟩ 命令，或一个\chardef 记号。如果没有这四种类型的 ⟨ 字符 ⟩ 之一，重音符号被视为一个\char 命令本身；这会得到一个“悬浮在空中”的重音符号。否则，重音符号将放置在紧随其后的字符上方。可以通过 ⟨ 可选赋值 ⟩ 来改变重音符号和字符之间的字体变化。

An unpleasant implication of the fact that an \accent command has to be followed by a ⟨character⟩ is that it is not possible to place an accent on a ligature, or two accents on top of each other. In some languages, such as Hindi or Vietnamese, such double accents do occur. Positioning accents on top of each other is possible, however, in math mode.

一个令人不愉快的事实是，\accent 命令后面必须跟一个 ⟨ 字符 ⟩，因此无法在连字上放置重音符号，也无法将两个重音符号叠放在一起。然而，在某些语言（如印地语或越南语）中，确实会出现双重重音符号。不过，在数学模式下，可以将重音符号叠放在一起。

The width of a character with an accent is the same as that of the unaccented character. TEX assumes that the accent as it appears in the font file is properly positioned for a character that is as high as the x-height of the font; for characters with other heights it correspondingly lowers or raises the accent.

带有重音符号的字符的宽度与无重音的字符相同。TEX 假设字体文件中的重音符号在正确的位置上，适合与字体的 x 高度一样高的字符；对于其他高度的字符，它相应地降低或提高重音符号的位置。

No genuine under-accents exist in TEX. They are implemented as low placed over-accents. A way of handling them more correctly would be to write a macro that measures the following character, and raises or drops the accent accordingly. The cedilla macro, \c, in plain TEX does something along these lines. However, it does not drop the accent for characters with descenders.

TEX 中不存在真正的下重音符号。它们被实现为位于底部的上重音符号。更正确处理它们的一种方法是编写一个宏，测量接下来的字符，并相应地提升或降低重音符号。在 plain TEX 中，附带的勾音宏 \c大致按照这个思路进行操作。然而，对于带有下行字符的字符，它不会降低重音符号的位置。

The horizontal positioning of an accent is controlled by \fontdimen1, slant per point. Kerns are used for the horizontal movement. Note that, although they are inserted automatically, these kerns are classified as explicit kerns. Therefore they inhibit hyphenation in the parts of the word before and after the kern.

重音符号的水平定位由\fontdimen1（每点的倾斜）控制。水平移动使用紧排。请注意，尽管它们是自动插入的，但这些紧排被归类为显式紧排。因此，它们会阻止单词的紧排出现在紧排之前和之后的部分。

As an example of kerning for accents, here follows the dump of a horizontal list.

作为重音符号紧排的一个示例，以下是水平列表的转储。

```
\setbox0=\hbox{\it \`l}
```

\showbox0

gives

\hbox(9.58334+0.0)x2.55554
.\kern -0.61803 (for accent)
.\hbox(6.94444+0.0)x5.11108, shifted -2.6389
..\tenit ^^R
.\kern -4.49306 (for accent)
.\tenit l

Note that the accent is placed first, so afterwards the italic correction of the last character is still available.

注意，重音符号首先放置，因此最后一个字符的斜体修正仍然可用。

3.5 Testing characters
测试字符

Equality of character codes is tested by \if:

使用 \if 可以测试字符码的相等性：

\if⟨token$_1$⟩⟨token$_2$⟩

Tokens following this conditional are expanded until two unexpandable tokens are left. The condition is then true if those tokens are character tokens with the same character code, regardless of category code.

在这个条件之后的记号会被展开，直到剩下两个不可展开的记号。如果这两个记号是具有相同字符码的字符记号，不考虑类别码，则条件为真。

An unexpandable control sequence is considered to have character code 256 and category code 16 (so that it is unequal to anything except another control sequence), except in the case where it had been \let to a non-active character token. In that case it is considered to have the character code and category code of that character. This was mentioned above.

不可展开的控制序列被认为具有字符码 256 和类别码 16（因此与除了另一个控制序列之外的任何内容都不相等），除非它是通过 \let 赋值给非活动字符记号。

在这种情况下，它被认为具有该字符的字符码和类别码。这在上面已经提到过。

The test \ifcat for category codes was mentioned in Chapter ??; the test

在第 ?? 章中已经提到了用于类别码的 \ifcat 测试；而测试

\ifx⟨token$_1$⟩⟨token$_2$⟩

can be used to test for category code and character code simultaneously. The tokens following this test are not expanded. However, if they are macros, TEX tests their expansions for equality.

可以用于同时测试类别码和字符码。这个测试之后的记号不会被展开。但是，如果它们是宏，TEX 会测试它们的展开是否相等。

Quantities defined by \chardef can be tested with \ifnum:

通过 \chardef 定义的字符可以使用 \ifnum 进行测试：

\chardef\a=`x \chardef\b=`y \ifnum\a=\b % is false

based on the fact (see Chapter ??) that ⟨chardef token⟩s can be used as numbers.

这是基于这样一个事实（参见第 ?? 章）：⟨chardef token⟩ 可以被视为数字。

3.6　Uppercase and lowercase
大写和小写

3.6.1　Uppercase and lowercase codes
大写和小写码

To each of the character codes correspond an uppercase code and a lowercase code (for still more codes see below). These can be assigned by

每个字符编码都对应一个大写码和一个小写码（有关更多编码，请参见下文）。可以通过以下方式进行赋值：

\uccode⟨number⟩⟨equals⟩⟨number⟩

and

\lccode⟨number⟩⟨equals⟩⟨number⟩.

In IniTEX codes `a..`z, `A..`Z have uppercase code `A..`Z and lowercase code `a..`z. All other character codes have both uppercase and lowercase code zero.

在 IniTEX 中，代码 a..z 和 A..Z 的大写码是 A..Z，小写码是 a..z。其他所有字符编码的大写码和小写码都为零。

3.6.2 Uppercase and lowercase commands 大写和小写命令

The commands \uppercase{...} and \lowercase{...} go through their argument lists, replacing all character codes of explicit character tokens by their uppercase and lowercase code respectively if these are non-zero, without changing the category codes.

命令 \uppercase{...} 和 \lowercase{...} 遍历其参数列表，用大写码和小写码替换所有显式字符记号的字符编码（如果它们非零），而不更改类别码。

The argument of \uppercase and \lowercase is a ⟨general text⟩, which is defined as

\uppercase 和 \lowercase 的参数是 ⟨general text⟩，其定义如下：

⟨general text⟩ ⟶ ⟨filler⟩{⟨balanced text⟩⟨right brace⟩

(for the definition of ⟨filler⟩ see Chapter ??) meaning that the left brace can be implicit, but the closing right brace must be an explicit character token with category code 2. TEX performs expansion to find the opening brace.

关于 ⟨filler⟩ 的定义，请参见第 ?? 章），这意味着左花括号可以是隐式的，但是右花括号必须是带有类别码为 2 的显式字符记号。TEX 进行展开以找到左花括号。

Uppercasing and lowercasing are executed in the execution processor; they are not 'macro expansion' activities like \number or \string. The sequence (attempting to produce \A)

大写和小写转换是在执行处理器中执行的；它们不像 \number 或 \string 那样是“宏展开”活动。序列（试图生成 \A）

\expandafter\csname\uppercase{a}\endcsname

gives an error (TEX inserts an \endcsname before the \uppercase because \uppercase is unexpandable), but

会报错（TEX 在 \uppercase 之前插入了 \endcsname，因为 \uppercase 是不可展开的），但是

\uppercase{\csname a\endcsname}

works.

可以正常工作。

As an example of the correct use of \uppercase, here is a macro that tests if a character is uppercase:

作为正确使用 \uppercase 的示例，下面是一个测试字符是否为大写的宏：

\def\ifIsUppercase#1{\uppercase{\if#1}#1}

The same test can be performed by \ifnum`#1=\uccode`#1.

可以使用 \ifnum#1=\uccode#1 执行相同的测试。

Hyphenation of words starting with an uppercase character, that is, a character not equal to its own \lccode, is subject to the \uchyph parameter: if this is positive, hyphenation of capitalized words is allowed. See also Chapter ??.

以大写字符开头的单词的断词处理，即字符不等于其自身的 \lccode，受到 \uchyph 参数的影响：如果该参数为正数，则允许对大写单词进行断词处理。另请参阅第 ?? 章。

3.6.3 Uppercase and lowercase forms of keywords
关键字的大写和小写形式

Each character in TEX keywords, such as pt, can be given in uppercase or lowercase form. For instance, pT, Pt, pt, and PT all have the same meaning. TEX does not use the \uccode and \lccode tables here to determine the lowercase form. Instead it converts uppercase characters to lowercase by adding 32 – the ascii difference between uppercase and lowercase characters – to their character code. This has some implications for implementations of TEX for non-roman alphabets; see page 370 of the TEX book, [?].

TEX 中的关键字，如 pt，可以以大写或小写形式给出。例如，pT、Pt、pt 和 PT 都具有相同的意义。在这里，TEX 不使用 \uccode 和 \lccode 表来确定小写形式。相反，它通过将大写字符的字符码增加 32（大写和小写字符之间的 ASCII 差异）来将大写字符转换为小写形式。这对于非罗马字母表的 TEX 实现有一些影响；请参阅《the TEX book》第 370 页 [?]。

3.6.4 Creative use of \uppercase and \lowercase 利用 \uppercase 和 \lowercase 进行创造性应用

The fact that \uppercase and \lowercase do not change category codes can sometimes be used to create certain character-code–category-code combinations that would otherwise be difficult to produce. See for instance the explanation of the \newif macro in Chapter ??, and another example on page ??.

\uppercase 和 \lowercase 不改变字符的类别码，这一事实有时可以用于创建某些字符码-类别码组合，否则可能很难产生。例如，参见第 ??章中对 \newif 宏的解释，以及第?? 页上的另一个示例。

For a slightly different application, consider the problem (solved by Rainer Schöpf) of, given a counter \newcount\mycount, writing character number \mycount to the terminal. Here is a solution:

对于稍微不同的应用，考虑以下问题（由 Rainer Sch”opf 解决）：给定计数器 \newcount\mycount，将编号为 \mycount 的字符写入终端。以下是一个解决方案：

```
\lccode`a=\mycount \chardef\terminal=16

\lowercase{\write\terminal{a}}
```

The \lowercase command effectively changes the argument of the \write command from ‘a’ into whatever it should be.

\lowercase 命令有效地将 \write 命令的参数从 ‘a’ 改变为它应该是的内容。

3.7 Codes of a character

字符的码

Each character code has a number of ⟨codename⟩s associated with it. These are integers in various ranges that determine how the character is treated in various contexts, or how the occurrence of that character changes the workings of TEX in certain contexts.

每个字符码都有一些与之关联的。这些是在各种范围内的整数，决定了字符在各种上下文中的处理方式，或者在特定上下文中出现该字符时如何改变 TEX 的工作方式。

The code names are as follows:

这些码名如下：

\catcode ⟨4-bit number⟩ (0–15); the category to which a character belongs. This is treated in Chapter ??.
⟨4-bit number⟩（0–15）；表示字符所属的类别码。详见第 ??章。

\mathcode ⟨15-bit number⟩ (0–"7FFF) or "8000; determines how a character is treated in math mode. See Chapter ??.
⟨15-bit number⟩（0–"7FFF）或 "8000；决定字符在数学模式中的处理方式。详见第??章。

\delcode ⟨27-bit number⟩ (0–"7 FFF FFF); determines how a character is treated after \left or \right in math mode. See page ??.
⟨27-bit number⟩（0–"7,FFF,FFF）；决定字符在数学模式中 \left 或 \right 之后的处理方式。详见第??页。

\sfcode integer; determines how spacing is affected after this character. See Chapter ??.
整数；决定字符之后的间距受到的影响。详见第?? 章。

\lccode, \uccode ⟨8-bit number⟩ (0-255); lowercase and uppercase codes – these were treated above.
[\lccode、\uccode] ⟨8-bit number⟩（0-255）；小写和大写代码 – 这些在上面已经介绍过。

3.8　Converting tokens into character strings

将记号转换为字符串

The command \string takes the next token and expands it into a string of separate characters. Thus

命令 \string 接受下一个记号并将其展开为一个由单独字符组成的字符串。因此，

\tt\string\control

will give \control in the output, and

将在输出中给出 \control，而

\tt\string$

will give $, but, noting that the string operation comes after the tokenizing,

将给出 $，但请注意，字符串操作在记号化之后进行，因此，

\tt\string%

will not give %, because the comment sign is removed by TEX's input processor. Therefore, this command will 'string' the first token on the next line.

不会给出 %，因为注释符号被 TEX 的输入处理器移除了。因此，这个命令将“字符串化”下一行的第一个记号。

The \string command is executed by the expansion processor, thus it is expanded unless explicitly inhibited (see Chapter ??).

\string 命令由展开处理器执行，因此除非明确禁止（详见第 ?? 章），否则它会被展开。

3.8.1 Output of control sequences 控制序列的输出

In the above examples the typewriter font was selected, because the Computer Modern roman font does not have a backslash character. However, TEX need not have used the backslash character to display a control sequence: it uses character number \escapechar. This same value is also used when a control sequence is output with \write, \message, or \errmessage, and it is used in the

output of \show, \showthe and \meaning. If \escapechar is negative or more than 255, the escape character is not output; the default value (set in IniTEX) is 92, the number of the backslash character.

在上述示例中选择了打字机字体，因为计算机现代罗马字体没有反斜杠字符。然而，TEX 并不一定使用反斜杠字符来显示控制序列：它使用字符编号 \escapechar。在使用 \write、\message 或 \errmessage 输出控制序列时，也会使用相同的值。并且在 \show、\showthe 和 \meaning 的输出中也会使用该值。如果 \escapechar 是负数或大于 255，则不会输出转义字符；默认值（在 IniTEX 中设置）为 92，即反斜杠字符的编号。

For use in a \write statement the \string can in some circumstances be replaced by \noexpand (see page ??).

在 \write 语句中，\string 在某些情况下可以被 \noexpand 替代（见第 ?? 页）。

3.8.2 Category codes of a \string
\string 的类别码

The characters that are the result of a \string command have category code 12, except for any spaces in a stringed control sequence; they have category code 10. Since inside a control sequence there are no category codes, any spaces resulting from \string are of necessity only space characters, that is, characters with code 32. However, TEX's input processor converts all space tokens that have a character code other than 32 into character tokens with character code 32, so the chances are pretty slim that 'funny spaces' wind up in control sequences.

\string 命令的结果字符的类别码为 12，除了字符串化的控制序列中的空格；它们的类别码为 10。由于在控制序列内部没有类别码，因此由 \string 生成的任何空格都是空格字符，即字符编码为 32 的字符。然而，TEX 的输入处理器会将所有字符编码不为 32 的空格记号转换为字符编码为 32 的字符记号，因此在控制序列中出现“奇怪的空格”的几率非常小。

Other commands with the same behaviour with respect to category codes as \string, are \number, \romannumeral, \jobname, \fontname, \meaning, and \the.

其他具有与 \string 相同类别码行为的命令包括 \number、\romannumeral、\jobname、\fontname、\meaning 和 \the。