The xparse package
Document command parser

The LaTeX Project* virhuiai@qq.com 翻译

Released 2022-01-12

The xparse package provides a high-level interface for producing document-level commands. In that way, it is intended as a replacement for the LaTeX 2ε `\newcommand` macro. However, xparse works so that the interface to a function (optional arguments, stars and mandatory arguments, for example) is separate from the internal implementation. xparse provides a normalised input for the internal form of a function, independent of the document-level argument arrangement.
xparse 包提供了一个高级接口，用于生成文档级命令。因此，它旨在取代LaTeX 2ε的 `\newcommand` 宏。然而，xparse 的工作方式是将函数的接口（例如可选参数、星号和必选参数）与内部实现分离开来。xparse 提供了一个标准化的输入，用于函数的内部形式，独立于文档级参数的安排。

At present, the functions in xparse which are regarded as "stable" are:
目前，xparse 中的被认为是“稳定”的函数有：

- `\NewDocumentCommand`
  `\RenewDocumentCommand`
  `\ProvideDocumentCommand`
  `\DeclareDocumentCommand`
- `\NewDocumentEnvironment`
  `\RenewDocumentEnvironment`
  `\ProvideDocumentEnvironment`
  `\DeclareDocumentEnvironment`
- `\NewExpandableDocumentCommand`

*E-mail: latex-team@latex-project.org

\RenewExpandableDocumentCommand
\ProvideExpandableDocumentCommand
\DeclareExpandableDocumentCommand

- \IfNoValue(TF)
- \IfValue(TF)
- \IfBoolean(TF)

with the other functions currently regarded as "experimental". Please try all of the commands provided here, but be aware that the experimental ones may change or disappear.
其他功能目前被视为“实验性”。请尝试这里提供的所有命令，但请注意实验性的命令可能会更改或消失。

1 Specifying arguments
指定参数

Before introducing the functions used to create document commands, the method for specifying arguments with xparse will be illustrated. In order to allow each argument to be defined independently, xparse does not simply need to know the number of arguments for a function, but also the nature of each one. This is done by constructing an *argument specification*, which defines the number of arguments, the type of each argument and any additional information needed for xparse to read the user input and properly pass it through to internal functions.
在介绍用于创建文档命令的函数之前，我们将演示使用 xparse 指定参数的方法。为了允许每个参数都独立定义，xparse 不仅需要知道函数的参数数量，还需要知道每个参数的性质。这是通过构建一个参数规范来实现的，它定义了参数的数量，每个参数的类型以及 xparse 读取用户输入并正确传递到内部函数所需的任何其他信息。

The basic form of the argument specifier is a list of letters, where each letter defines a type of argument. As will be described below, some of the types need additional information, such as default values. The argument types can be divided into two, those which define arguments that are mandatory (potentially raising an error if not found) and those which define optional arguments. The mandatory types are:
参数说明符的基本形式是一个字母列表，其中每个字母定义一个参数类型。如下所

述，一些类型需要额外的信息，例如默认值。参数类型可以分为两类，一类定义必需的参数（如果未找到可能引发错误），另一类定义可选参数。必需的类型包括：

`m` A standard mandatory argument, which can either be a single token alone or multiple tokens surrounded by curly braces `{}`. Regardless of the input, the argument will be passed to the internal code without the outer braces. This is the xparse type specifier for a normal TEX argument.
标准的必须参数，可以是单个标记或被大括号 `{}` 包围的多个标记。无论输入是什么，参数都将被传递给内部代码，但外部大括号将被去除。这是 xparse 的一种类型指示符，用于普通的 TEX 参数。

`r` Given as `r`⟨*token1*⟩⟨*token2*⟩, this denotes a "required" delimited argument, where the delimiters are ⟨*token1*⟩ and ⟨*token2*⟩. If the opening delimiter ⟨*token1*⟩ is missing, the default marker `-NoValue-` will be inserted after a suitable error.
给定为 `r`⟨*token1*⟩⟨*token2*⟩，表示一个"必需"的定界参数，其中定界符是 ⟨*token1*⟩ 和 ⟨*token2*⟩。如果缺少开头的定界符 ⟨*token1*⟩，则会在适当的错误后插入默认标记 `-NoValue-`。

`R` Given as `R`⟨*token1*⟩⟨*token2*⟩`{`⟨*default*⟩`}`, this is a "required" delimited argument as for `r`, but it has a user-definable recovery ⟨*default*⟩ instead of `-NoValue-`.
给定为 `R`⟨*token1*⟩⟨*token2*⟩`{`⟨*default*⟩`}`，这是一个"必需"的定界参数，就像 `r` 一样，但它有一个可由用户定义的恢复 ⟨*default*⟩，而不是 `-NoValue-`。

`v` Reads an argument "verbatim", between the following character and its next occurrence, in a way similar to the argument of the LATEX 2_ε command `\verb`. Thus a `v`-type argument is read between two identical characters, which cannot be any of `%`, `\`, `#`, `{`, `}` or ␣. The verbatim argument can also be enclosed between braces, `{` and `}`. A command with a verbatim argument will produce an error when it appears within an argument of another function.
以"原样输出"的方式读取参数，介于以下字符和其下一次出现之间，类似于 LATEX 2_ε 命令 `\verb` 的参数。因此，`v` 类型的参数在两个相同的字符之间读取，这些字符不能是 `%`、`\`、`#`、`{`、`}` 或 ␣。原样输出的参数也可以被大括号 `{` 和 `}` 包围。带有原样输出参数的命令将在另一个函数的参数中出现时产生错误。

`b` Only suitable in the argument specification of an environment, it denotes the body of the environment, between `\begin{`⟨*environment*⟩`}` and `\end{`⟨*environment*⟩`}`. See Section 1.6 for details.
仅适用于环境的参数规范，表示环境的主体部分，介于 `\begin{`⟨*environment*⟩`}`

和 \end{⟨*environment*⟩} 之间。有关详细信息，请参阅第 1.6 节。

The types which define optional arguments are:
定义可选参数的类型有：

o A standard LaTeX optional argument, surrounded with square brackets, which will supply the special -NoValue- marker if not given (as described later).
一个标准的 LaTeX 可选参数，用方括号括起来，如果没有给出值，将提供特殊的 -NoValue- 标记（稍后会描述）。

d Given as d⟨*token1*⟩⟨*token2*⟩, an optional argument which is delimited by ⟨*token1*⟩ and ⟨*token2*⟩. As with o, if no value is given the special marker -NoValue- is returned.
给定为 d⟨*token1*⟩⟨*token2*⟩，一个可选参数，由 ⟨*token1*⟩ 和 ⟨*token2*⟩ 分隔。与 o 类型一样，如果没有给出值，则返回特殊标记 -NoValue-。

O Given as O{⟨*default*⟩}, is like o, but returns ⟨*default*⟩ if no value is given.
给定为 O{⟨*default*⟩}，与 o 类似，但如果没有给出值，则返回 ⟨*default*⟩。

D Given as D⟨*token1*⟩⟨*token2*⟩{⟨*default*⟩}, it is as for d, but returns ⟨*default*⟩ if no value is given. Internally, the o, d and O types are short-cuts to an appropriated-constructed D type argument.
给定为 D⟨*token1*⟩⟨*token2*⟩{⟨*default*⟩}，与 d 类型一样，但如果没有给出值，则返回 ⟨*default*⟩。在内部，o、d 和 O 类型是一个适当构造的 D 类型参数的快捷方式。

s An optional star, which will result in a value \BooleanTrue if a star is present and \BooleanFalse otherwise (as described later).
一个可选的星号，如果存在，则结果为 \BooleanTrue，否则为 \BooleanFalse（稍后会描述）。

t An optional ⟨*token*⟩, which will result in a value \BooleanTrue if ⟨*token*⟩ is present and \BooleanFalse otherwise. Given as t⟨*token*⟩.
一个可选的 ⟨*token*⟩，如果 ⟨*token*⟩ 存在，则结果为 \BooleanTrue，否则为 \BooleanFalse。给定为 t⟨*token*⟩。

e Given as e{⟨*tokens*⟩}, a set of optional *embellishments*, each of which requires a *value*. If an embellishment is not present, -NoValue- is returned. Each embellishment gives one argument, ordered as for the list of ⟨*tokens*⟩ in the argument specification. All ⟨*tokens*⟩ must be distinct. *This is an experimental*

type.
给定为 `e{`⟨*tokens*⟩`}`，一组可选的装饰，每个装饰都需要一个值。如果装饰不存在，则返回 `-NoValue-`。每个装饰都给出一个参数，按参数规范中 ⟨*tokens*⟩ 列表的顺序排列。所有的 ⟨*tokens*⟩ 必须是不同的。这是一种实验性类型。

`E` As for `e` but returns one or more ⟨*defaults*⟩ if values are not given: `E{`⟨*tokens*⟩`}{`⟨*defaults*⟩`}`. See Section 1.5 for more details.
与 `e` 类似，但如果没有给出值，则返回一个或多个 ⟨*defaults*⟩`:E{`⟨*tokens*⟩`}{`⟨*defaults*⟩`}`。有关更多详细信息，请参见第 1.5 节。

Using these specifiers, it is possible to create complex input syntax very easily. For example, given the argument definition '`s o o m O{default}`', the input '`*[Foo]{Bar}`' would be parsed as:
使用这些标识符，可以非常容易地创建复杂的输入语法。例如，给定参数定义 `s o o m O{default}`，输入 `*[Foo]{Bar}` 将被解析为：

- `#1` = `\BooleanTrue`
- `#2` = `Foo`
- `#3` = `-NoValue-`
- `#4` = `Bar`
- `#5` = `default`

whereas '`[One][Two]{}[Three]`' would be parsed as:
而 '`[One][Two]{}[Three]`' 将被解析为：

- `#1` = `\BooleanFalse`
- `#2` = `One`
- `#3` = `Two`
- `#4` =
- `#5` = `Three`

Delimited argument types (`d`, `o` and `r`) are defined such that they require matched pairs of delimiters when collecting an argument. For example
定义了分隔的参数类型（`d`、`o` 和 `r`），它们要求在收集参数时需要匹配的分隔符对。例如：

```
\NewDocumentCommand{\foo}{o}{#1}
\foo[[content]] % #1 = "[content]"
\foo[[]         % Error: missing closing "]"
```

Also note that `{` and `}` cannot be used as delimiters as they are used by TEX as grouping tokens. Implicit begin- or end-group tokens (*e.g.*, `\bgroup` and `\egroup`) are not allowed for delimited argument tipes. Arguments to be grabbed inside these tokens must be created as either `m`- or `g`-type arguments.
还要注意的是，`{` 和 `}` 不能用作分隔符，因为它们被 TEX 用作分组标记。隐式的起始或结束分组标记（例如，`\bgroup` 和 `\egroup`）不能用于分隔参数类型。要在这些标记内获取的参数必须创建为 `m`- 或 `g`-类型参数。

Within delimited arguments, non-balanced or otherwise awkward tokens may be included by protecting the entire argument with a brace pair
在指定的参数范围内，可以通过用花括号保护整个参数来包含非平衡或其他不方便的标记。

```
\NewDocumentCommand{\foobar}{o}{#1}
\foobar[{[}]          % Allowed as the "[" is 'hidden'
```

These braces will be stripped only if they surround the *entire* content of the optional argument
只有当这些大括号包围整个可选参数的内容时，它们才会被移除。

```
\NewDocumentCommand{\foobaz}{o}{#1}
\foobaz[{abc}]          % => "abc"
\foobaz[ {abc}]          % => " {abc}"
```

Two more characters have a special meaning when creating an argument specifier. First, `+` is used to make an argument long (to accept paragraph tokens). In contrast to LATEX 2ε's `\newcommand`, this applies on an argument-by-argument basis. So modifying the example to '`s o o +m O{default}`' means that the mandatory argument is now `\long`, whereas the optional arguments are not.
在创建参数说明符时，另外两个字符具有特殊含义。首先，`+` 用于使参数变长（接受段落标记）。与 LATEX 2ε的 `\newcommand` 相比，这是基于逐个参数应用的。因此，将示例修改为“`s o o +m O{default}`”意味着强制性参数现在是 `\long`，而可选参数不是。

Secondly, the character `>` is used to declare so-called "argument processors", which can be used to modify the contents of an argument before it is passed to the macro definition. The use of argument processors is a somewhat advanced topic, (or at least a less commonly used feature) and is covered in Section 3.2.
其次，字符 `>` 用于声明所谓的“参数处理器”，它可以用于在将参数传递给宏定义之

前修改参数的内容。使用参数处理器是一个相对高级的主题（或者至少是一个不太常用的特性），这个主题在第 3.2节中有所涉及。

When an optional argument is followed by a mandatory argument with the same delimiter, xparse issues a warning because the optional argument could not be omitted by the user, thus becoming in effect mandatory. This can apply to `o`, `d`, `O`, `D`, `s`, `t`, `e`, and `E` type arguments followed by `r` or `R`-type required arguments, but also to `g` or `G` type arguments followed by `m` type arguments.
当一个可选参数后面紧跟着一个带有相同分隔符的必选参数时，xparse 会发出警告，因为用户不能省略可选参数，实际上变成了必选参数。这适用于 `o`，`d`，`O`，`D`，`s`，`t`，`e` 和 `E` 类型的参数，后面跟着 `r` 或 `R` 类型的必选参数，也适用于 `g` 或 `G` 类型的参数后面跟着 `m` 类型的参数。

As xparse is also used to describe interfaces that have appeared in the wider LaTeX 2_ε eco-system, it also defines additional argument types, described in Section 1.8: the mandatory types `l` and `u` and the optional brace group types `g` and `G`. Their use is not recommended because it is simpler for a user if all packages use a similar syntax. For the same reason, delimited arguments `r`, `R`, `d` and `D` should normally use delimiters that are naturally paired, such as `[` and `]` or `(` and `)`, or that are identical, such as `"` and `"`. A very common syntax is to have one optional argument `o` treated as a key–value list (using for instance l3keys) followed by some mandatory arguments `m` (or `+m`).
由于 xparse 也用于描述出现在更广泛的 LaTeX 2_ε 生态系统中的接口，它还定义了附加的参数类型，如第1.8 节所述：必需类型 `l` 和 `u`，以及可选的大括号组类型 `g` 和 `G`。这些类型的使用并不建议，因为如果所有的包都使用类似的语法，对用户来说会更简单。出于同样的原因，定界参数 `r`、`R`、`d` 和 `D` 通常应该使用天然成对的定界符，例如 `[` 和 `]` 或 `(` 和 `)`，或者使用相同的定界符，例如 `"` 和`"`。一个非常常见的语法是有一个可选的参数 `o`，被视为键值列表（例如使用 l3keys），后面跟着一些必需参数 `m`（或 `+m`）。

1.1 Spacing and optional arguments
间距和可选参数

TeX will find the first argument after a function name irrespective of any intervening spaces. This is true for both mandatory and optional arguments. So `\foo[arg]` and `\foo␣␣␣[arg]` are equivalent. Spaces are also ignored when collecting arguments up to the last mandatory argument to be collected (as it must exist). So after

TEX 会在函数名称后找到第一个参数，无论有没有空格。这适用于必需参数和可选参数。因此，`\foo[arg]` 和 `\foo␣[arg]` 是等效的。当收集参数时，空格也被忽略，直到收集到最后一个必需参数（因为它必须存在）。因此，在收集到最后一个必需参数之后，以下内容：

```
\NewDocumentCommand \foo { m o m } { ... }
```

the user input `\foo{arg1}[arg2]{arg3}` and `\foo{arg1}␣␣[arg2]␣␣␣{arg3}` will both be parsed in the same way.
用户输入的 `\foo{arg1}[arg2]{arg3}` 和 `\foo{arg1}␣[arg2]␣{arg3}` 将以相同的方式解析。

The behavior of optional arguments *after* any mandatory arguments is selectable. The standard settings will allow spaces here, and thus with
在任何必选参数之后的可选参数的行为是可选择的。标准设置允许在此处使用空格，因此：

```
\NewDocumentCommand \foobar { m o } { ... }
```

both `\foobar{arg1}[arg2]` and `\foobar{arg1}␣[arg2]` will find an optional argument. This can be changed by giving the modified `!` in the argument specification:
`\foobar{arg1}[arg2]` 和 `\foobar{arg1}␣[arg2]` 均会寻找可选参数。可以通过在参数规范中添加修改后的 `!` 来改变这种情况：

```
\NewDocumentCommand \foobar { m !o } { ... }
```

where `\foobar{arg1}␣[arg2]` will not find an optional argument.
`\foobar{arg1}␣[arg2]` 不会找到可选参数。

There is one subtlety here due to the difference in handling by TEX of "control symbols", where the command name is made up of a single character, such as "`\\`". Spaces are not ignored by TEX here, and thus it is possible to require an optional argument directly follow such a command. The most common example is the use of `\\` in amsmath environments. In xparse terms it has signature
这里有一个微妙之处，因为 TEX 处理“控制符号”（即命令名称由单个字符组成，如“`\\`”）的方式不同。在这里，TEX 不会忽略空格，因此可能需要直接在此类命令后面跟随一个可选参数。最常见的例子是在 amsmath 环境中使用`\\`。在 xparse 术语中，它的签名为：

```
\DeclareDocumentCommand \\ { !s !o } { ... }
```

1.2 Required delimited arguments
必需的定界参数

The contrast between a delimited (D-type) and "required delimited" (R-type) argument is that an error will be raised if the latter is missing. Thus for example
定界型（D 类型）和“必需的定界”（R 类型）参数之间的区别在于，如果缺少后者，将会引发错误。例如：

```
\NewDocumentCommand {\foobaz} {r()m} {}
\foobaz{oops}
```

will lead to an error message being issued. The marker -NoValue- (r-type) or user-specified default (for R-type) will be inserted to allow error recovery.
将导致发出错误消息。标记 -NoValue-（r-类型）或用户指定的默认值（对于 R-类型）将被插入以允许错误恢复。

1.3 Verbatim arguments
抄录参数

Arguments of type v are read in verbatim mode, which will result in the grabbed argument consisting of tokens of category codes 12 ("other") and 13 ("active"), except spaces, which are given category code 10 ("space"). The argument is delimited in a similar manner to the LaTeX 2_ε \verb function, or by (correctly nested) pairs of braces.
类型为 v 的参数以抄录模式读取，因此所得到的参数由类别码为 12（“其它”）和 13（“活动”）的记号组成，除空格外，空格则被赋予类别码为 10（“空格”）。该参数的定界方式与LaTeX 2_ε的\verb函数或正确嵌套的一对大括号类似。

Functions containing verbatim arguments cannot appear in the arguments of other functions. The v argument specifier includes code to check this, and will raise an error if the grabbed argument has already been tokenized by TeX in an irreversible way.
包含抄录（verbatim）参数的函数不能出现在其他函数的参数中。v 参数说明符包括用于检查这一点的代码，并且如果抓取的参数已经被 TeX 不可逆地分解为记号，则会引发错误。

By default, an argument of type v must be at most one line. Prefixing with + allows line breaks within the argument.

默认情况下，类型为 v 的参数最多只能有一行。在参数前加上 + 可以允许在参数内换行。

Users should note that support for verbatim arguments is somewhat experimental. Feedback is therefore very welcome on the LaTeX-L mailing list.
用户应该注意，对于逐字逐句的参数支持还处于实验阶段。因此，对于 LaTeX-L 邮件列表上的反馈非常欢迎。

1.4 Default values of arguments
参数的默认值

Uppercase argument types (O, D, …) allow to specify a default value to be used when the argument is missing; their lower-case counterparts use the special marker -NoValue-. The default value can be expressed in terms of the value of any other arguments by using #1, #2, and so on.
大写的参数类型（O，D，等等）允许在参数缺失时指定默认值；它们的小写版本使用特殊标记 -NoValue-。默认值可以使用 #1、#2 等来表示任何其他参数的值。

```
\NewDocumentCommand {\conjugate} { m O{#1ed} O{#2} } {(#1,#2,#3)}
\conjugate {walk}            % => (walk,walked,walked)
\conjugate {find} [found]    % => (find,found,found)
\conjugate {do} [did] [done] % => (do,did,done)
```

The default values may refer to arguments that appear later in the argument specification. For instance a command could accept two optional arguments, equal by default:
默认值可能指后面出现在参数规范中的参数。例如，一个命令可以接受两个可选参数，默认情况下相等：

```
\NewDocumentCommand {\margins} { O{#3} m O{#1} m } {(#1,#2,#3,#4)}
\margins {a} {b}              % => {(-NoValue-,a,-NoValue-,b)}
\margins [1cm] {a} {b}        % => {(1cm,a,1cm,b)}
\margins {a} [1cm] {b}        % => {(1cm,a,1cm,b)}
\margins [1cm] {a} [2cm] {b}  % => {(1cm,a,2cm,b)}
```

Users should note that support for default arguments referring to other arguments is somewhat experimental. Feedback is therefore very welcome on the LaTeX-L mailing list.

用户应注意，默认参数引用其他参数的支持有些实验性质。因此，非常欢迎在 LaTeX-L 邮件列表上提供反馈意见。

1.5 Default values for “embellishments” 关于“装饰”的默认值

The E-type argument allows one default value per test token. This is achieved by giving a list of defaults for each entry in the list, for example:
E 类型参数允许每个测试令牌设置一个默认值。这可以通过为列表中的每个条目提供默认值列表来实现，例如：

```
E{^_}{{UP}{DOWN}}
```

If the list of default values is *shorter* than the list of test tokens, the special -NoValue- marker will be returned (as for the e-type argument). Thus for example
如果默认值列表比测试标记列表更短，则将返回特殊的-NoValue-标记（就像 e-类型参数一样）。例如：

```
E{^_}{{UP}}
```

has default UP for the ^ test character, but will return the -NoValue- marker as a default for _. This allows mixing of explicit defaults with testing for missing values.
对于测试字符 ^，默认为 UP，但对于 _，默认将返回 -NoValue- 标记。这使得可以将显式默认值与测试缺失值混合使用。

1.6 Body of an environment 环境的主体

While environments \begin{⟨*environment*⟩} … \end{⟨*environment*⟩} are typically used in cases where the code implementing the ⟨*environment*⟩ does not need to access the contents of the environment (its “body”), it is sometimes useful to have the body as a standard argument.
尽管在实现⟨环境⟩时通常不需要访问环境的内容（即其“主体”），但在某些情况下，将主体作为标准参数是有用的。环境\begin{⟨*environment*⟩} … \end{⟨*environment*⟩}通常用于这种情况。

This is achieved in xparse by ending the argument specification with b. The approach taken in xparse is different from the earlier packages environ or newenviron: the body

of the environment is provided to the code part as a usual argument `#1`, `#2` etc., rather than stored in a macro such as `\BODY`.
在 xparse 中，通过在参数规范的末尾添加 `b`，实现了这一点。xparse 采取的方法不同于早期的包 environ 或 newenviron：环境的主体作为常规参数 `#1`，`#2` 等提供给代码部分，而不是存储在诸如`\BODY` 之类的宏中。

For instance
例如

```
\NewDocumentEnvironment { twice }
  { O{\ttfamily} +b }
  {#2#1#2} {}
\begin{twice}[\itshape]
  Hello world!
\end{twice}
```

typesets "Hello world!*Hello world!*".

The prefix `+` is used to allow multiple paragraphs in the environment's body. Argument processors can also be applied to `b` arguments.
前缀`+` 用于在环境主体中允许多个段落。也可以将参数处理器应用于 `b` 参数。

By default, spaces are trimmed at both ends of the body: in the example there would otherwise be spaces coming from the ends the lines after `[\itshape]` and `world!`. Putting the prefix `!` before `b` suppresses space-trimming.
默认情况下，正文两端的空格会被删除：例如，在 `[\itshape]` 和 `world!` 之后的行末会有空格。在 `b` 前加上前缀 `!` 可以抑制空格修剪。

When `b` is used in the argument specification, the last argument of `\NewDocumentEnvironment`, which consists of an ⟨*end code*⟩ to insert at `\end{`⟨*environment*⟩`}`, is redundant since one can simply put that code at the end of the ⟨*start code*⟩. Nevertheless this (empty) ⟨*end code*⟩ must be provided.
当在参数规范中使用 `b` 时，`\NewDocumentEnvironment` 的最后一个参数，包括在`\end{`⟨*environment*⟩`}` 处插入的⟨*end code*⟩是多余的，因为可以直接将该代码放在⟨*start code*⟩的末尾。然而，必须提供这个 (空的)⟨*end code*⟩。

Environments that use this feature can be nested.
使用此功能的环境可以嵌套。

Users should note that this feature is somewhat experimental. Feedback is therefore

very welcome on the LaTeX-L mailing list.
用户应注意，此功能有些实验性质。因此，在 LaTeX-L 邮件列表上非常欢迎反馈。

1.7 Starred environments
带星号的环境

Many packages define environments with and without * in their name, for instance tabular and tabular*. At present, xparse does not provide specific tools to define these: one should simply define the two environment separately, for instance
许多包在其名称中定义了带有和不带有 的环境，例如 tabular 和 tabular*。目前，xparse 没有提供定义这些环境的特定工具：应该分别定义这两个环境，例如

```
\NewDocumentEnvironment { tabular } { o +m } {...} {...}
\NewDocumentEnvironment { tabular* } { m o +m } {...} {...}
```

Of course the implementation of these two environments, denoted "..." in this example, can rely on the same internal commands.
当然，在这个例子中标记为"..."的这两个环境的实现可以依赖于相同的内部命令。

Note that this situation is different from the s argument type: if the signature of an environment starts with s then the star is searched for after the argument of \begin. For instance, the following typesets star.
请注意，这种情况与 s 参数类型不同：如果一个环境的签名以 s 开头，则星号会在 \begin 的参数之后搜索。例如，以下代码将排版出 star。

```
\NewDocumentEnvironment { envstar } { s }
  {\IfBooleanTF {#1} {star} {no star}} {}
\begin{envstar}*
\end{envstar}
```

1.8 Backwards Compatibility
向后兼容性

One role of xparse is to describe existing LaTeX interfaces, including some that are rather unusual in LaTeX (as opposed to formats such as plain TeX) such as delimited arguments. As such, the package defines some argument specifiers that should largely be avoided nowadays as using them in packages leads to inconsistent user interfaces.

The simplest syntax is often best, with argument specifications such as `mmmm` or `ommmm`, namely an optional argument followed by some standard mandatory ones. The optional argument can be made to support key–value syntax using tools from l3keys.
xparse 的一个作用是描述现有的 LaTeX 接口，包括一些在 LaTeX 中非常不寻常的接口（与 plain TeX 等格式不同），例如定界参数。因此，该包定义了一些应该尽量避免在包中使用的参数说明符，因为使用它们会导致不一致的用户界面。最简单的语法通常是最好的，使用参数说明符例如 `mmmm` 或 `ommmm`，即一个可选参数后面跟一些标准的必选参数。可选参数可以使用 l3keys 中的工具支持键值语法。

The argument types that are not recommended any longer are:
不再推荐使用的参数类型包括：

`l` A mandatory argument which reads everything up to the first begin-group token: in standard LaTeX this is a left brace.
必选参数，读取直到遇到第一个组开始标记的所有内容：在标准 LaTeX 中，这是左大括号。

`u` Reads a mandatory argument "until" ⟨*tokens*⟩ are encountered, where the desired ⟨*tokens*⟩ are given as an argument to the specifier: `u{`⟨*tokens*⟩`}`.
读取必选参数，直到遇到 ⟨*tokens*⟩ 为止，所需的 ⟨*tokens*⟩ 作为参数传递给该说明符：`u{`⟨*tokens*⟩`}`。

`g` An optional argument given inside a pair of TeX group tokens (in standard LaTeX, `{ …}`), which returns `-NoValue-` if not present.
可选参数，给定在一对 TeX 组标记内（在标准 LaTeX 中为 `{ …}`），如果不存在则返回 `-NoValue-`。

`G` As for `g` but returns ⟨*default*⟩ if no value is given: `G{`⟨*default*⟩`}`.`G{`⟨*default*⟩`}`。
与 `g` 相同，但如果没有给定值，则返回 ⟨*default*⟩：

1.9 Details about argument delimiters
关于参数分界符的详细信息

In normal (non-expandable) commands, the delimited types look for the initial delimiter by peeking ahead (using expl3's `\peek_...` functions) looking for the delimiter token. The token has to have the same meaning and "shape" of the token defined as delimiter. There are three possible cases of delimiters: character tokens, control

sequence tokens, and active character tokens. For all practical purposes of this description, active character tokens will behave exactly as control sequence tokens.
在普通（不可展开的）命令中，分界符类型通过向前查看（使用 expl3 的\peek_...函数）寻找初始分界符，查找分界符令牌。该令牌必须具有与定义为分界符的令牌相同的含义和“形状”。分界符有三种可能的情况：字符令牌、控制序列令牌和活动字符令牌。在本描述的实际目的上，活动字符令牌将完全像控制序列令牌一样行事。

1.9.1 Character tokens
字符记号

A character token is characterised by its character code, and its meaning is the category code (\catcode). When a command is defined, the meaning of the character token is fixed into the definition of the command and cannot change. A command will correctly see an argument delimiter if the open delimiter has the same character and category codes as at the time of the definition. For example in:
字符记号由其字符代码和类别码 (\catcode) 来确定其含义。当定义一个命令时，字符记号的含义将被固定在命令的定义中，不能改变。如果开放式分隔符与定义时相同的字符和类别码匹配，那么命令将正确地看到一个参数分隔符。例如：

```
\NewDocumentCommand { \foobar } { D<>{default} } {(#1)}
\foobar <hello> \par
\char_set_catcode_letter:N <
\foobar <hello>
```

the output would be:

```
(hello)
(default)<hello>
```

as the open-delimter < changed in meaning between the two calls to \foobar, so the second one doesn't see the < as a valid delimiter. Commands assume that if a valid open-delimiter was found, a matching close-delimiter will also be there. If it is not (either by being omitted or by changing in meaning), a low-level TEX error is raised and the command call is aborted.
由于两个对 \foobar 的调用之间打开分隔符 < 的含义改变了，因此第二个调用不会将 < 视为有效的分隔符。命令假定如果找到了有效的开分隔符，相应的闭分隔符也将存在。如果没有（因为被省略或含义改变了），就会引发低级 TEX 错误并中止命令调用。

1.9.2 Control sequence tokens
控制序列标记

A control sequence (or control character) token is characterised by is its name, and its meaning is its definition. A token cannot have two different meanings at the same time. When a control sequence is defined as delimiter in a command, it will be detected as delimiter whenever the control sequence name is found in the document regardless of its current definition. For example in:
控制序列（或控制字符）标记以其名称为特征，其含义为其定义。一个标记不能同时具有两个不同的含义。当控制序列被定义为命令中的分隔符时，无论当前定义如何，只要在文档中找到了控制序列名称，它就会被检测为分隔符。例如，在：

```
\cs_set:Npn \x { abc }
\NewDocumentCommand { \foobar } { D\x\y{default} } {(#1)}
\foobar \x hello\y \par
\cs_set:Npn \x { def }
\foobar \x hello\y
```

the output would be:
中，输出将是：

```
(hello)
(hello)
```

with both calls to the command seeing the delimiter `\x`.
两次调用命令都看到了分隔符 `\x`。

2 Declaring commands and environments
声明命令和环境

With the concept of an argument specifier defined, it is now possible to describe the methods available for creating both functions and environments using xparse.
有了参数规范的概念，现在可以描述使用 xparse 创建函数和环境的方法。

The interface-building commands are the preferred method for creating document-level functions in LaTeX3. All of the functions generated in this way are naturally robust (using the ε-TeX `\protected` mechanism).

在 LaTeX3 中，界面构建命令是创建文档级函数的首选方法。所有以这种方式生成的函数都是天然的健壮（使用 ε-TeX 的 `\protected` 机制）。

`\NewDocumentCommand`
`\RenewDocumentCommand`
`\ProvideDocumentCommand`
`\DeclareDocumentCommand`

`\NewDocumentCommand` ⟨*function*⟩ {⟨*arg spec*⟩} {⟨*code*⟩}

This family of commands are used to create a document-level ⟨*function*⟩. The argument specification for the function is given by ⟨*arg spec*⟩, and the function expands to the ⟨*code*⟩ with `#1`, `#2`, etc. replaced by the arguments found by `xparse`.
这一系列命令用于创建文档级 ⟨*function*⟩。函数的参数规范由 ⟨*arg spec*⟩ 给出，函数展开为 `xparse` 找到的参数替换 `#1`、`#2` 等等的 ⟨*code*⟩。

As an example:
例如：

```
\NewDocumentCommand \chapter { s o m }
  {
    \IfBooleanTF {#1}
      { \typesetstarchapter {#3} }
      { \typesetnormalchapter {#2} {#3} }
  }
```

would be a way to define a `\chapter` command which would essentially behave like the current LaTeX 2_ε command (except that it would accept an optional argument even when a `*` was parsed). The `\typesetnormalchapter` could test its first argument for being `-NoValue-` to see if an optional argument was present.
这是一种定义 `\chapter` 命令的方式，它基本上表现得像当前的 LaTeX 2_ε 命令（除了即使解析了 `*`，它也会接受可选参数）。`\typesetnormalchapter` 可以测试其第一个参数是否为 `-NoValue-`，以查看可选参数是否存在。

The difference between the `\New…` `\Renew…`, `\Provide…` and `\Declare…` versions is the behaviour if ⟨*function*⟩ is already defined.
`\New…`、`\Renew…`、`\Provide…` 和 `\Declare…` 版本之间的区别是如果已经定义了 ⟨*function*⟩，它们的行为不同。

- `\NewDocumentCommand` will issue an error if ⟨*function*⟩ has already been defined.
  如果已经定义了 ⟨*function*⟩，`\NewDocumentCommand` 将发出错误。
- `\RenewDocumentCommand` will issue an error if ⟨*function*⟩ has not previously been defined.
  如果之前未定义 ⟨*function*⟩，`\RenewDocumentCommand` 将发出错误。

- `\ProvideDocumentCommand` creates a new definition for ⟨*function*⟩ only if one has not already been given.
  仅当没有给出定义时，`\ProvideDocumentCommand` 才创建一个新的 ⟨*function*⟩ 定义。

- `\DeclareDocumentCommand` will always create the new definition, irrespective of any existing ⟨*function*⟩ with the same name. This should be used sparingly.
  `\DeclareDocumentCommand` 总是创建新的定义，而不管同名的现有 ⟨*function*⟩。这应该谨慎使用。

TEXhackers note: Unlike LaTeX 2ε's `\newcommand` and relatives, the `\NewDocumentCommand` family of functions do not prevent creation of functions with names starting `\end…`.
与 LaTeX 2ε 的 `\newcommand` 等不同，`\NewDocumentCommand` 函数族不会阻止创建以 `\end…` 开头的函数。

`\NewDocumentEnvironment` `\NewDocumentEnvironment {⟨environment⟩} {⟨arg spec⟩}`
`\RenewDocumentEnvironment` `{⟨start code⟩} {⟨end code⟩}`
`\ProvideDocumentEnvironment`
`\DeclareDocumentEnvironment`

These commands work in the same way as `\NewDocumentCommand`, etc., but create environments (`\begin{⟨environment⟩} … \end{⟨environment⟩}`). Both the ⟨*start code*⟩ and ⟨*end code*⟩ may access the arguments as defined by ⟨*arg spec*⟩. The arguments will be given following `\begin{⟨environment⟩}`.
这些命令与 `\NewDocumentCommand` 等的工作方式相同，但创建环境（`\begin{⟨environment⟩} … \end{⟨environment⟩}`）。⟨*start code*⟩ 和 ⟨*end code*⟩ 都可以访问按 ⟨*arg spec*⟩ 定义的参数。这些参数将在 `\begin{⟨environment⟩}` 之后给出。

3 Other xparse commands
xparse 的其他命令

3.1 Testing special values
测试特殊值

Optional arguments created using xparse make use of dedicated variables to return information about the nature of the argument received.

使用 xparse 创建的可选参数使用专用变量返回有关接收到的参数的信息。

\IfNoValueT ⋆ \IfNoValueTF {⟨*argument*⟩} {⟨*true code*⟩} {⟨*false code*⟩}
\IfNoValueF ⋆ \IfNoValueT {⟨*argument*⟩} {⟨*true code*⟩}
\IfNoValueTF ⋆ \IfNoValueF {⟨*argument*⟩} {⟨*false code*⟩}

The \IfNoValue(TF) tests are used to check if ⟨*argument*⟩ (#1, #2, *etc.*) is the special -NoValue- marker For example
\IfNoValue(TF) 测试用于检查 ⟨*argument*⟩（#1、#2 等）是否为特殊的 -NoValue- 标记。例如：

```
\NewDocumentCommand \foo { o m }
  {
    \IfNoValueTF {#1}
      { \DoSomethingJustWithMandatoryArgument {#2} }
      {  \DoSomethingWithBothArguments {#1} {#2}   }
  }
```

will use a different internal function if the optional argument is given than if it is not present.
如果提供了可选参数，将使用不同的内部函数，否则将使用不同的内部函数。

Note that three tests are available, depending on which outcome branches are required: \IfNoValueTF, \IfNoValueT and \IfNoValueF.
请注意，根据所需的结果分支，有三种可用的测试：\IfNoValueTF、\IfNoValueT 和 \IfNoValueF。

As the \IfNoValue(TF) tests are expandable, it is possible to test these values later, for example at the point of typesetting or in an expansion context.
由于\IfNoValue(TF)测试是可展开的，因此可以在稍后进行这些值的测试，例如在排版或扩展上下文的点上。

It is important to note that -NoValue- is constructed such that it will *not* match the simple text input -NoValue-, *i.e.* that
需要注意的是，-NoValue-被构建成不会匹配简单文本输入 -NoValue-，也就是说，它不会被认为是相同的。

```
\IfNoValueTF{-NoValue-}
```

will be logically false.
将逻辑上为 false。

When two optional arguments follow each other (a syntax we typically discourage), it can make sense to allow users of the command to specify only the second argument by providing an empty first argument. Rather than testing separately for emptyness and for -NoValue- it is then best to use the argument type O with an empty default value, and simply test for emptyness using the expl3 conditional \tl_if_blank:nTF or its etoolbox analogue \ifblank.
当两个可选参数相互跟随（一种我们通常不鼓励的语法）时，允许命令的用户仅通过提供空的第一个参数来指定第二个参数是有意义的。与其分别测试空值和-NoValue-，最好使用带有空默认值的参数类型 O，并使用 expl3 条件\tl_if_-

\IfValueT ⋆
\IfValueF ⋆
\IfValueTF ⋆

\IfValueTF {⟨*argument*⟩} {⟨*true code*⟩} {⟨*false code*⟩}

The reverse form of the \IfNoValue(TF) tests are also available as \IfValue(TF). The context will determine which logical form makes the most sense for a given code scenario.

\IfNoValue(TF)测试的反向形式也可以使用\IfValue(TF)。上下文将决定哪种逻辑形式在给定的代码场景中最合理。

\BooleanFalse
\BooleanTrue

The true and false flags set when searching for an optional character (using s or t⟨*char*⟩) have names which are accessible outside of code blocks.

在搜索可选字符（使用 s 或 t⟨*char*⟩）时设置的 true 和 false 标志具有可以在代码块之外访问的名称。

\IfBooleanT ⋆
\IfBooleanF ⋆
\IfBooleanTF ⋆

\IfBooleanTF {⟨*argument*⟩} {⟨*true code*⟩} {⟨*false code*⟩}

Used to test if ⟨*argument*⟩ (#1, #2, *etc.*) is \BooleanTrue or \BooleanFalse. For example

用于测试 ⟨*argument*⟩（#1，#2，等等）是否为\BooleanTrue或\BooleanFalse。例如：

```
\NewDocumentCommand \foo { s m }
  {
    \IfBooleanTF {#1}
      { \DoSomethingWithStar {#2} }
      { \DoSomethingWithoutStar {#2} }
  }
```

checks for a star as the first argument, then chooses the action to take based on this information.

检查第一个参数是否为星号，然后根据这些信息选择要采取的动作。

3.2 Argument processors
参数处理器

xparse introduces the idea of an argument processor, which is applied to an argument *after* it has been grabbed by the underlying system but before it is passed to ⟨*code*⟩. An argument processor can therefore be used to regularise input at an early

stage, allowing the internal functions to be completely independent of input form. Processors are applied to user input and to default values for optional arguments, but *not* to the special `-NoValue-` marker.
xparse 引入了参数处理器的概念，它在底层系统获取参数后，但在传递给 ⟨*code*⟩ 之前被应用于参数。因此，参数处理器可以在早期阶段规范化输入，使得内部函数完全独立于输入形式。处理器应用于用户输入和可选参数的默认值，但不应用于特殊的 `-NoValue-` 标记。

Each argument processor is specified by the syntax `>{`⟨*processor*⟩`}` in the argument specification. Processors are applied from right to left, so that
每个参数处理器由语法 `>{`⟨处理器⟩`}` 在参数规范中指定。处理器从右向左应用，这样就可以...

```
>{\ProcessorB} >{\ProcessorA} m
```

would apply `\ProcessorA` followed by `\ProcessorB` to the tokens grabbed by the `m` argument.
将 `\ProcessorA` 应用于由 `m` 参数抓取的标记，接着应用 `\ProcessorB`。

It might sometimes be useful to use the value of another argument as one of the arguments of a processor. For example, using the `\SplitList` processor defined below,
有时候将另一个参数的值用作处理器的参数可能会很有用。例如，可以使用下面定义的`\SplitList`处理器，

```
\NewDocumentCommand \foo { O{,} >{\SplitList{#1}} m } { \foobar{#2} }
\foo{a,b;c,d}
```

results in `\foobar` receiving the argument `{a}{b;c}{d}` because `\SplitList` receives as its two arguments the optional one (whose value here is the default, a comma) and the mandatory one. To summarize, first the arguments are searched for in the input, then any default argument is determined as explained in Section 1.4, then these default arguments are passed to any processor. When referring to arguments (through `#1`, `#2` and so on) in a processor, the arguments used are always those before applying any processor.
因为 `\SplitList` 接收两个参数，第一个是可选参数（这里的默认值是逗号），第二个是必选参数，所以 `\foobar` 接收到了参数 `{a}{b;c}{d}`。总之，首先在输入中搜索参数，然后根据第1.4节中的说明确定任何默认参数，然后将这些默认参数传递给任何处理器。在处理器中引用参数（通过 `#1`、`#2` 等），使用的参数总是在应用任何

处理器之前的参数。

\ProcessedArgument

xparse defines a very small set of processor functions. In the main, it is anticipated that code writers will want to create their own processors. These need to accept one argument, which is the tokens as grabbed (or as returned by a previous processor function). Processor functions should return the processed argument as the variable \ProcessedArgument.

xparse 定义了一组非常小的处理器函数。主要预期代码编写者将想要创建自己的处理器。这些处理器需要接受一个参数，即被抓取的标记（或由先前的处理器函数返回的标记）。处理器函数应将处理后的参数作为变量 \ProcessedArgument 返回。

\ReverseBoolean

\ReverseBoolean

This processor reverses the logic of \BooleanTrue and \BooleanFalse, so that the example from earlier would become

这个处理器反转了 \BooleanTrue 和 \BooleanFalse 的逻辑，因此之前的例子将变成

```
\NewDocumentCommand \foo { > { \ReverseBoolean } s m }
  {
    \IfBooleanTF #1
      { \DoSomethingWithoutStar {#2} }
      { \DoSomethingWithStar {#2} }
  }
```

\SplitArgument

Updated: 2012-02-12

\SplitArgument {⟨*number*⟩} {⟨*token(s)*⟩}

This processor splits the argument given at each occurrence of the ⟨*tokens*⟩ up to a maximum of ⟨*number*⟩ tokens (thus dividing the input into ⟨*number*⟩ + 1 parts). An error is given if too many ⟨*tokens*⟩ are present in the input. The processed input is placed inside ⟨*number*⟩ + 1 sets of braces for further use. If there are fewer than {⟨*number*⟩} of {⟨*tokens*⟩} in the argument then -NoValue- markers are added at the end of the processed argument.

这个处理器将给定的参数在每次出现⟨*tokens*⟩处分割，最多分割⟨*number*⟩个 tokens（因此将输入分成 ⟨*number*⟩ + 1 部分）。如果输入中存在太多的⟨*tokens*⟩，则会给出错误。处理后的输入被放置在 ⟨*number*⟩ + 1 组大括号中以供进一步使用。如果参数中的 {⟨*tokens*⟩} 少于 {⟨*number*⟩} 个，则在处理后的参数末尾添加-NoValue-标记。

```
\NewDocumentCommand \foo
  { > { \SplitArgument { 2 } { ; } } m }
  { \InternalFunctionOfThreeArguments #1 }
```

If only a single character ⟨*token*⟩ is used for the split, any category code 13 (active) character matching the ⟨*token*⟩ will be replaced before the split takes place. Spaces are trimmed at each end of each item parsed.

如果只使用单个字符⟨*token*⟩进行拆分，则在进行拆分之前，任何与⟨*token*⟩匹配的类别码为 13（活动）的字符都将被替换。每个解析的项的两端都会修剪空格。

\SplitList `\SplitList {⟨token(s)⟩}`

This processor splits the argument given at each occurrence of the ⟨*token(s)*⟩ where the number of items is not fixed. Each item is then wrapped in braces within `#1`. The result is that the processed argument can be further processed using a mapping function.
该处理器在每个出现⟨*token(s)*⟩的地方将给定的参数分割，其中项数不固定。然后，在`#1` 中将每个项包裹在大括号中。结果是，可以使用映射函数进一步处理处理后的参数。

```
\NewDocumentCommand \foo
  { > { \SplitList { ; } } m }
  { \MappingFunction #1 }
```

If only a single character ⟨*token*⟩ is used for the split, any category code 13 (active) character matching the ⟨*token*⟩ will be replaced before the split takes place. Spaces are trimmed at each end of each item parsed.
如果只使用单个字符 ⟨*token*⟩ 进行拆分，则在拆分之前，与 ⟨*token*⟩ 匹配的任何类别码 13（活动）字符都将被替换。在解析每个项的每端修剪空格。

\ProcessList ⋆ `\ProcessList {⟨list⟩} {⟨function⟩}`

To support `\SplitList`, the function `\ProcessList` is available to apply a ⟨*function*⟩ to every entry in a ⟨*list*⟩. The ⟨*function*⟩ should absorb one argument: the list entry. For example
为了支持`\SplitList`,函数`\ProcessList`可用于对⟨*list*⟩中的每个条目应用一个⟨*function*⟩。该⟨*function*⟩应该吸收一个参数：列表条目。例如：

```
\NewDocumentCommand \foo
  { > { \SplitList { ; } } m }
  { \ProcessList {#1} { \SomeDocumentFunction } }
```

This function is experimental.

\TrimSpaces \TrimSpaces

Removes any leading and trailing spaces (tokens with character code 32 and category code 10) for the ends of the argument. Thus for example declaring a function
去除参数两端的任何前导和尾随空格（字符码为 32 且类别码为 10 的标记）。例如，声明一个函数：

```
\NewDocumentCommand \foo
  { > { \TrimSpaces } m }
  { \showtokens {#1} }
```

and using it in a document as
并将其用于文档中作为

```
\foo{ hello world }
```

will show `hello world` at the terminal, with the space at each end removed. `\TrimSpaces` will remove multiple spaces from the ends of the input in cases where these have been included such that the standard TEX conversion of multiple spaces to a single space does not apply.
将在终端上显示 `hello world`，并删除每个末尾的空格。`\TrimSpaces`将从输入的末尾删除多个空格，在这种情况下，如果标准的 TEX 将多个空格转换为单个空格，则不适用。

This function is experimental.
这个函数是试验性的。

3.3 Fully-expandable document commands
完全可展开的文档命令

There are *very rare* occasion when it may be useful to create functions using a fully-expandable argument grabber. To support this, xparse can create expandable functions as well as the usual robust ones. This imposes a number of restrictions on the nature of the arguments accepted by a function, and the code it implements. This facility should only be used when *absolutely necessary*; if you do not understand when this might be, *do not use these functions*!
在极为罕见的情况下，创建使用完全可展开的参数获取器的函数可能会很有用。为了支持这一点，xparse 可以创建可展开函数以及通常的强健函数。这对函数所接受的参数的性质和它实现的代码施加了一些限制。只有在绝对必要的情况下才应该使用

这个功能；如果你不知道什么时候可能需要，不要使用这些函数！

\NewExpandableDocumentCommand
\RenewExpandableDocumentCommand
\ProvideExpandableDocumentCommand
\DeclareExpandableDocumentCommand

\NewExpandableDocumentCommand
⟨*function*⟩ {⟨*arg spec*⟩} {⟨*code*⟩}

This family of commands is used to create a document-level ⟨*function*⟩, which will grab its arguments in a fully-expandable manner. The argument specification for the function is given by ⟨*arg spec*⟩, and the function will execute ⟨*code*⟩. In general, ⟨*code*⟩ will also be fully expandable, although it is possible that this will not be the case (for example, a function for use in a table might expand so that `\omit` is the first non-expandable non-space token).
这组命令用于创建一个文档级别的 ⟨*function*⟩，它将以完全可展开的方式获取其参数。函数的参数规格由 ⟨*arg spec*⟩ 给出，函数将执行 ⟨*code*⟩。通常情况下，⟨*code*⟩ 也将是完全可展开的，尽管有可能不是这种情况（例如，用于表格的函数可能会展开，以便 `\omit` 是第一个不可展开的非空格记号）。

Parsing arguments expandably imposes a number of restrictions on both the type of arguments that can be read and the error checking available:
可扩展解析参数会对可以读取的参数类型和可用的错误检查施加一些限制。

- The last argument (if any are present) must be one of the mandatory types `m`, `r`, `R`, `l` or `u`.
  最后一个参数（如果有的话）必须是以下必选类型之一：`m`、`r`、`R`、`l` 或 `u`。
- All short arguments appear before long arguments.
  所有短参数必须出现在长参数之前。
- The mandatory argument types `l` and `u` may not be used after optional arguments.
  必选参数类型 `l` 和 `u` 不能在可选参数之后使用。
- The optional argument types `g` and `G` are not available.
  可选参数类型 `g` 和 `G` 不可用。
- The "verbatim" argument type `v` is not available.
  "抄录" 参数类型 `v` 不可用。
- Argument processors (using `>`) are not available.
  参数处理器（使用 `>`）不可用。
- It is not possible to differentiate between, for example `\foo[` and `\foo{[}`: in both cases the `[` will be interpreted as the start of an optional argument. As a result, checking for optional arguments is less robust than in the standard version.
  不可能区分类似于 `\foo[` 和 `\foo{[}` 的情况：在两种情况下，`[` 都将被解释为可选参数的开始。因此，检查可选参数不如标准版本健壮。

xparse will issue an error if an argument specifier is given which does not conform to the first six requirements. The last item is an issue when the function is used, and

3.4 Access to the argument specification
获取参数规范

The argument specifications for document commands and environments are available for examination and use.
文档命令和环境的参数规范可供检查和使用。

`\GetDocumentCommandArgSpec` `\GetDocumentCommandArgSpec` ⟨*function*⟩
`\GetDocumentEnvironmentArgSpec` `\GetDocumentEnvironmentArgSpec {`⟨*environment*⟩`}`

These functions transfer the current argument specification for the requested ⟨*function*⟩ or ⟨*environment*⟩ into the token list variable `\ArgumentSpecification`. If the ⟨*function*⟩ or ⟨*environment*⟩ has no known argument specification then an error is issued. The assignment to `\ArgumentSpecification` is local to the current TEX group.
这些函数将请求的⟨*function*⟩或⟨*environment*⟩的当前参数规范转换为记号列表变量`\ArgumentSpecification`。如果⟨*function*⟩或⟨*environment*⟩没有已知的参数规范，则会发出错误。对`\ArgumentSpecification`的赋值是局部的，仅限于当前的 TEX 组。

`\ShowDocumentCommandArgSpec` `\ShowDocumentCommandArgSpec` ⟨*function*⟩
`\ShowDocumentEnvironmentArgSpec` `\ShowDocumentEnvironmentArgSpec {`⟨*environment*⟩`}`

These functions show the current argument specification for the requested ⟨*function*⟩ or ⟨*environment*⟩ at the terminal. If the ⟨*function*⟩ or ⟨*environment*⟩ has no known argument specification then an error is issued.
这些函数在终端显示所请求的⟨*function*⟩或⟨*environment*⟩的当前参数规范。如果⟨*function*⟩或⟨*enviror*
有已知的参数规范，则会发出错误。

4 Load-time options
加载时选项

`log-declarations` The package recognises the load-time option `log-declarations`, which is a key–value option taking the value `true` and `false`. By default, the option is set to `false`, meaning that no command or environment declared is logged. By loading xparse using
该包识别加载时选项 `log-declarations`，这是一个键-值选项，取值为 `true` 和

`false`。默认情况下，该选项设置为 `false`，这意味着不会记录任何声明的命令或环境。通过使用 xparse 加载，可以将该选项设置为 `true`，以记录所有声明的命令和环境。

```
\usepackage[log-declarations=true]{xparse}
```

each new, declared or renewed command or environment is logged.
每个新声明或重新声明的命令或环境都会被记录。