

l3doc 文档类 — 实验性质*

The L^AT_EX3 Project[†] 2023 年 12 月 11 日 发布

张泓知 2023 年 12 月 24 日 【译】

目 录

1	介绍	2
2	其他包的特性	2
2.1	hypdoc 包	2
2.2	docmfp 包	2
2.3	xdoc2 包	3
2.4	gmdoc 包	3
3	问题与待办事项	3
4	文档	4
4.1	配置	4
4.2	类选项	4
4.3	文档和代码实现的分割	5
4.4	一般文本标记	5
4.5	在文档中描述函数	7
4.6	描述实现中的函数	8
4.7	保持一致性	9
4.8	文档化模板	10
	索引	10

*根据广泛需求，我们现在发布了这个实验性的类文档。但请注意，它绝不是最终版本，并且很有可能会经历修改，甚至是不兼容的修改！因此，如果类发生变化，可能需要进行更新才能继续使用。

[†]<https://www.latex-project.org/latex3/>

1 介绍

在 doc 从版本 2 变更为版本 3 之前编写了此类的代码和文档，这已经显示出这个类目前的落后程度。所以请认真对待以下警告：

**它的稳定性远不如主要的 expl3 包。
请自行承担风险！**

这是一个专门用于记录 expl3 捆绑包的类，它是组成 L^AT_EX3 编程环境的模块或包的集合。最终它将取代 ltxdoc 类作为 L^AT_EX3 的文档类，但在吸收 hypdoc、xdoc2、docmfp 和 gmdoc 中的优秀思想之前不会这样做。

它被编写为一个“自包含”的 docstrip 文件：执行 `latex l3doc.dtx` 将生成文件 `l3doc.cls` 并排版此文档；执行 `tex l3doc.dtx` 将只生成 `l3doc.cls`。

2 其他包的特性

这个类基于 ltxdoc 类和 doc 宏包，但在它们最初编写之后，一些改进和替代方案出现了，我们希望能够借鉴这些新特性。

这些包或类有 hypdoc、docmfp、gmdoc 和 xdoc。我在下面对它们进行了总结，以便确定我们至少应该为 l3doc 设定什么样的最低特性。

2.1 hypdoc 包

此包为 doc 包提供了超链接支持。我将它包含在此列表中是为了提醒我，文档和方法实现之间的交叉引用并不是很好。（例如，能够自动地从方法实现链接到其文档说明，反之亦然，将会很不错。）

2.2 docmfp 包

- 为 MetaFont 和 MetaPost 代码提供了 `\DescribeRoutine` 和 `routine` 环境（等等）。
- 为更通用的代码提供了 `\DescribeVariable` 和 `variable` 环境（等等）。
- 提供了 `\Describe` 和 `Code` 环境（等等）作为上述两个实例的一般化。
- 对 DocStrip 系统进行了小的调整，以帮助非 L^AT_EX 的使用。

2.3 xdoc2 包

- 双面打印支持。
- `\NewMacroEnvironment`、`\NewDescribeEnvironment`；与 `docmfp` 类似的概念但更全面。
- 大量小改进。

2.4 gmdoc 包

将 `doc` 作为包或类进行了根本性的重新实现。

- 不需要 `\begin{macrocode}` 块！
- 自动插入 `\begin{macro}` 块！
- 还有许多其他细微的改进。

3 问题与待办事项

目前存在的问题：(1) 对可以记录的内容类型不够灵活；(2) `\begin{function}` 环境用于记录内容，与在实现中类似地使用的 `\begin{macro}` 函数之间没有明显的联系。

在用于实现部分时，`macro` 可能应该改名为 `function`。但在这种改名发生之前，它们应该具有相同的语法！

此外，我们需要另一层文档命令来处理“用户宏”与“代码函数”；`expl3` 函数可能需要不同的文档方式（至少在索引方面），与 `ltxcmd` 用户宏不同。

以下是一些待完成事项的列表，没有特定顺序：

- 将 `function/macro` 环境重命名，以更好地描述其用途。
- 普遍化 `function/macro`，用于记录“其他内容”，如环境名称、包选项，甚至键值选项。
- 像 `\part` 一样新增一个用于文件的函数（删除笨拙的“File”作为 `\partname`）。
- 寻找更好的替代方案来取代 `\StopEventually`；我考虑使用两个环境 `documentation` 和 `implementation`，它们可以有条件地排版/忽略其内容。（这已经被实现，但需要进一步考虑。）
- 将宏的文档和实现进行超链接（参考 `svn-multi v2` 的 `DTX` 文件）。现在这部分已经部分完成，但需要改进。

4 文档

4.1 配置

在处理类选项之前，l3doc 如果存在配置文件 l3doc.cfg，将加载它，允许你在不必更改文档源文件的情况下定制类的行为。

例如，要在信纸大小的纸张上生成文档而不是默认的 A4 大小，创建 l3doc.cfg，并包含以下内容：

```
\PassOptionsToClass{letterpaper}{l3doc}
```

默认情况下，l3doc 选择 T1 字体编码并加载 Latin Modern 字体。要阻止这一行为，可以使用类选项 cm-default。

4.2 类选项

该类识别了许多选项，其中一些是通常有用的，另一些则专门针对内核团队使用。

- `full` 当设置 full 选项时（标准设置），源文件的文档和实现部分都会排版。另一方面，如果设置了 onlydoc 选项，则只会排版文档部分。
- `onlydoc`
- `lm-default` 选择标准字体设置是在 T1 编码下的 Latin Modern（标准设置），还是保持字体设置不变。
- `kernel` 确定 l3doc 是否将 `__kernel_` 命令和 `\(cgl)__kernel_` 变量视为代码中可接受的内容。一般来说，不允许来自当前模块外部的内部内容。然而，为了引导 expl3 内核，需要一些跨模块的功能。为了避免否则会出现的错误消息，可以使用类选项 kernel。
- `check` 给定 check 选项时，类将记录在 `<name>.cmds` 文件中定义和记录的所有命令。这将显示哪些命令既被记录又被定义，哪些仅被记录，以及哪些仅被定义。（这里，“定义”指的是在源文件的实现部分使用 macro 或 variable 环境列出的命令。）
- `checktest` 给定 checktest 选项时，类将检查源文件实现部分中的每个函数条目是否使用了 `\UnitTest` 进行了标记。
- `show-notes` 这些互补选项确定是否打印使用 `\NB` 和 `\NOTE` 命令提供的信息。
- `hide-notes` 命令 `\cmd` 和 `\cs` 允许在大多数下划线后进行连字符的处理。默认情况下，会使用连字符标记连字符位置，但可以使用 `cs-break-nohyphen` 类选项进行更改。若要完全禁用控制序列的连字符处理，使用 `cs-break = false`。
- `cs-break`
- `cs-break-nohyphen`

4.3 文档和代码实现的分割

doc 使用 `\OnlyDocumentation/\AlsoImplementation` 宏来指导 `\StopEventually{}` 的使用，该命令用于在单个 `.dtx` 文件中分隔文档和实现部分。

这并不十分灵活，因为它假定我们总是要打印文档部分。对于 `expl3` 源文件，我希望能够以两种模式输入 `.dtx` 文件：只显示文档部分和只显示实现部分。例如：

```
\DisableImplementation
\DocInput{l3basics,l3prg,...}
\EnableImplementation
\DisableDocumentation
\DocInputAgain
```

`expl3` 包的整个文档，包括实现部分在最后。这不是完美的，但是这是一个开始。

在文档部分使用 `\begin{documentation}...\end{documentation}`，在实现部分使用 `\begin{implementation}...\end{implementation}`。

`\EnableDocumentation/\EnableImplementation` 使其在 `.dtx` 文件 `\DocInput` 时能够排版；使用 `\DisableDocumentation/\DisableImplementation` 可以省略这些环境的内容。

注意，`\DocInput` 现在接受逗号分隔的参数，并且 `\DocInputAgain` 可以重新输入以这种方式先前输入的所有 `.dtx` 文件。

4.4 一般文本标记

本节中的许多命令来自于 `ltxdoc`，做了一些改进。

`\cmd` `\cmd` [*options*] *control sequence*

`\cs` `\cs` [*options*] {*csname*}

这些命令用于排版控制序列。`\cmd\foo` 生成 “\foo”，而 `\cs{foo}` 也生成相同的效果。通常情况下，`\cs` 更健壮，因为它不依赖于类别码是否“正确”，因此更推荐使用。

这些命令知道 `@@ l3docstrip` 语法，并正确替换文档中的这些实例。这仅在 `%<@@=<module>` 声明之后发生。

此外，命令可以用在 `\cs` 的参数中。例如，`\cs{\meta{name}:\meta{signature}}` 生成 `\<name>:\<signature>`。

`<选项>` 是一个键值列表，可以包含以下键：

- `index=<name>`：将 `<csname>` 索引，就好像写了 `\cs{<name>}` 一样。
- `no-index`：不索引 `<csname>`。
- `module=<module>`：在 `<module>` 的命令列表中索引 `<csname>`；特别的，`<module>` 可以是 `TeX`，表示 “`TeX` 和 `LaTeX 2ε`” 命令，或者为空，表示放在主索引中。默认情况下，`<module>` 从命令名称中自动推断。
- `replace` 是一个布尔键（默认为 `true`），表示是否像 `l3docstrip` 那样替换 `@@`。

这些命令允许在大多数下划线后进行连字符处理。默认情况下，会使用连字符标记连字符位置，但可以使用 `cs-break-nohyphen` 类选项进行更改。若要完全禁用控制序列的连字符处理，使用 `cs-break = false`。

`\tn` `\tn` [*options*] {*csname*}

与 `\cs` 类似，但用于“传统”`TeX` 或 `LaTeX 2ε` 命令；它们会相应地进行索引。实际上，这相当于 `\cs [module=TeX, replace=false, <options>] {<csname>}`。

`\meta` `\meta` {*name*}

`\meta` 以斜体在 *<angle brackets>* 中排版 *<name>*。在 `function` 等环境中，尖括号 `<...>` 被设置为 `\meta{...}` 的简写。

与其 `ltxdoc` 版本相比，此函数有额外功能；下划线可以用于标记数学模式中的下标。例如，`\meta{arg_{xy}}` 生成 “*<arg_{xy}>*”。

`\Arg` `\Arg` {*name*}

`\marg` 将 *<name>* 以 `\meta` 的方式排版，并用大括号包裹。

`\oarg` `\marg/\oarg/\parg` 版本从 `ltxdoc` 派生，分别用于 `LaTeX 2ε` 语法中的“必选”、“可选”或“图片”的括号形式。

`\file` `\pkg` `{\name}`

`\env` 这些命令都接受一个参数，用于表示文件、环境、包名和类名的语义命令。

`\pkg`

`\cls`

`\NB` `\NB` `{\tag}` `{\comments}`

`\NOTE` `\begin{NOTE}` `{\tag}`

`\end{NOTE}`

`\end{NOTE}`

在源文件中做注释，默认情况下不进行排版。当激活 `show-notes` 类选项时，注释以非标记和抄录的方式排版。

4.5 在文档中描述函数

`function` (*env.*) 有两个经常使用的环境来描述 `expl3` 的函数和变量。如果描述一个变量，使用后者的环境；它与 `function` 环境的行为完全相同。通常，上述两个环境会与 `syntax` `syntax` (*env.*) 环境结合使用，以描述它们的语法。

```
\begin{function}{\package_function_one:N, \package_function_two:n}
  \begin{syntax}
    \cs{package_function_one:N} \meta{cs}
    \cs{package_function_two:n} \marg{Argument}
  \end{syntax}
  这里是描述的文字 ...
\end{function}
```

`\package_function_one:N` `\package_function_one:N` `<cs>`

`\package_function_two:n` `\package_function_two:n` `{\Argument}`

这里是描述的文字...

函数环境可以带有可选参数，表示所描述的函数是可展开的（使用`EXP`）、受限可展开的（使用`rEXP`），或以条件形式定义（使用`TF`、`pTF`或`noTF`）。注意，`pTF` 意味着 `EXP`，因为谓词必须始终是可展开的，而`noTF`表示函数在没有`TF`的情况下应该另外进行文档化。对于条件形式`TF`和`pTF`，`function`环境的参数实际上并不是一个存在的命令：在下面的示例中，`\tl_if_empty:N`并不存在，但它的条件形式`\tl_if_empty:NT`、`\tl_if_empty:NF`、`\tl_if_empty:NTF`，以及谓词形式`\tl_if_empty_p:N`是存在的：

```

\begin{function}[pTF]{\tl_if_empty:N, \tl_if_empty:c}
  \begin{syntax}
    \cs{tl_if_empty_p:N} \meta{tl~var}
    \cs{tl_if_empty:NTF} \meta{tl~var} \Arg{true code} \Arg{false code}
  \end{syntax}
  检查\meta{token list variable} 是否完全为空（即不包含任何标记）。
\end{function}

```

```

\__tl_if_empty_p:N * \tl_if_empty_p:N <tl var>
\__tl_if_empty_p:c * \tl_if_empty:NTF <tl var> {\true code}
\__tl_if_empty:N\TF * {\false code}
\__tl_if_empty:c * 检查<token list variable>是否完全为空
                    (即不包含任何标记)。

```

`texnote (env.)` 这个环境用于突出显示仅对经验丰富的 T_EX 开发人员感兴趣的 `function` 和类似环境中的部分内容。

4.6 描述实现中的函数

`macro (env.)` 在 L^AT_EX 2_ε 中用于标记宏/函数实现的常用环境仍然是 `macro` 环境。在 l3doc 中有一些变化：现在它接受逗号分隔的函数列表，以避免大量连续的 `\end{macro}` 语句。空格和换行被忽略（选项 `[verb]` 可以防止这种情况）。

```

% \begin{macro}{\foo:N, \foo:c}
%   \begin{macrocode}
... code for \foo:N and \foo:c ...
%   \end{macrocode}
% \end{macro}

```

如果你正在文档化辅助宏，通常不需要如此突出它，也不需要检查它是否具有测试函数，是否在 `function` 环境中先有文档块。l3doc 将从名称中的 `__` 的存在或使用 `\begin{macro}[int]` 强制标记为内部来识别这些情况。对于这些情况，边距标注将以灰色打印出来。

对于文档化 expl3 类型的条件语句，你也可以在环境中传递 TF 选项（并从函数名称中省略它），表示该函数提供了 T、F 和 TF 后缀。类似的 pTF 选项会打印出 TF 和 _p 谓词形式。选项 noTF 会打印出 TF 形式和既没有 T 也没有 F 的形式，用于文档化诸如 `\prop_get:NN` 这样也有条件形式的函数（`\prop_get:NNTF`）。

在极少数情况下，一个“公共”函数没有用户文档。在这些罕见情况下，可以添加选项 `no-user-doc` 来抑制未定义引用。

`\TestFiles` `\TestFiles{<文件列表>}` 用于指示当前代码使用的测试文件；它们将在文档中打印出来。

`\UnitTested` 在 `macro` 环境中，标记命令是否已创建单元测试是个好主意。这可通过在 `\begin{macro} ... \end{macro}` 之间的任何位置写入 `\UnitTested` 来表示。

如果启用了类选项 `checktest`，那么在没有调用 `Testfiles` 的 `macro` 环境中会产生一个错误。这是为了像 `expl3` 这样的大型包设计的，这些包应该有完全详尽的测试套件，其作者在添加新代码时可能不总是如应该般及时添加新测试。

`\TestMissing` 如果一个函数缺少测试，可以通过写（需要多次）`\TestMissing {<explanation of test required>}` 来标记这些缺失的测试。这些缺失的测试将在编译运行结束时的列表中进行总结打印。

`variable (env.)` 在文档化变量定义时，使用 `variable` 环境代替。它的行为与 `macro` 环境完全相同，只是如果启用了类选项 `checktest`，则不需要为变量提供测试文件。

`arguments (env.)` 在 `macro` 环境中，你可以使用 `arguments` 环境描述函数的参数。它的行为类似于修改后的 `enumerate` 环境。

```
% \begin{macro}{\foo:nn, \foo:VV}
% \begin{arguments}
%   \item Name of froozle to be frazzled
%   \item Name of muble to be jubled
% \end{arguments}
%   \begin{macrocode}
... code for \foo:nn and \foo:VV ...
%   \end{macrocode}
% \end{macro}
```

4.7 保持一致性

每当使用 `function` 或 `macro` 文档化或定义一个函数时，其名称都会存储在一个序列中以供以后处理。

在文档末尾（即在处理完 `.dtx` 文件之后），会分析名称列表，检查是否所有已定义的函数都已经文档化，反之亦然。结果将打印在控制台输出中。

如果你需要对这些名称列表进行更严格的处理，可以查看数据结构和用于直接存储和访问它们的方法的实现。

4.8 文档化模板

提供以下宏用于文档化模板；可能最终会变成完全不同的内容，但谁知道呢。

```
\begin{TemplateInterfaceDescription} {\langle template type name \rangle}
  \TemplateArgument{none}{---}
```

或者一个或多个这些：

```
\TemplateArgument {\langle arg no \rangle} {\langle meaning \rangle}
```

和

```
\TemplateSemantics
  \langle text describing the template type semantics \rangle
```

```
\end{TemplateInterfaceDescription}
```

```
\begin{TemplateDescription} {\langle template type name \rangle} {\langle name \rangle}
```

一个或多个这些：

```
\TemplateKey {\langle key name \rangle} {\langle type of key \rangle}
  {\langle textual description of meaning \rangle}
  {\langle default value if any \rangle}
```

和

```
\TemplateSemantics
  \langle text describing special additional semantics of the template \rangle
```

```
\end{TemplateDescription}
```

```
\begin{InstanceDescription} [\langle text to specify key column width (optional) \rangle]
  {\langle template type name \rangle} {\langle instance name \rangle} {\langle template name \rangle}
```

一个或多个这些：

```
\InstanceKey {\langle key name \rangle} {\langle value \rangle}
```

和

```
\InstanceSemantics
  \langle text describing the result of this instance \rangle
```

```
\end{InstanceDescription}
```

索引

斜体数字指向相应条目描述的页面，下划线数字指向定义的代码行，其它的都指向使用条目的页面。

A	\Arg	6
\AlsoImplementation		5
		10

arguments (env.)	9		
		L	
		lm-default (option)	4
	C		
check (option)	4		M
checktest (option)	4	macro (env.)	8
\cls	7	\marg	6
\cmd	6	\meta	6
\cs	6		N
cs-break (option)	4	\NB	7
cs-break-nohyphen (option)	4	\NewDescribeEnvironment	3
		\NewMacroEnvironment	3
	D	\NOTE	4
\Describe	2		O
\DescribeRoutine	2	\oarg	6
\DescribeVariable	2	onlydoc (option)	4
\DisableDocumentation	5	\OnlyDocumentation	5
\DisableImplementation	5	options:	
\DocInput	5	check	4
\DocInputAgain	5	checktest	4
	E	cs-break	4
\EnableDocumentation	5	cs-break-nohyphen	4
\EnableImplementation	5	full	4
\env	7	hide-notes	4
environments:		kernel	4
arguments	9	lm-default	4
function	7	onlydoc	4
macro	8	show-notes	4
syntax	7		P
texnote	8	package commands:	
variable	9, 7	\package_function_one:N	7
		\package_function_two:n	7
	F	\parg	6
\file	7	\pkg	7
\foo	6		S
full (option)	4	show-notes (option)	4
function (env.)	7	\StopEventually	3
	H	syntax (env.)	7
hide-notes (option)	4		T
	K	\TestFiles	9
kernel (option)	4		

<code>\TestFiles</code>	9	<code>\tl_if_empty_p:N</code>	8
<code>\TestMissing</code>	9	<code>\tn</code>	6
<code>\TestMissing</code>	9		
U			
TeX and L ^A T _E X 2 _ε commands:			
<code>\part</code>	3	<code>\UnitTest</code>	4
<code>\partname</code>	3	<code>\UnitTested</code>	9
<code>texnote (env.)</code>	8	<code>\UnitTested</code>	9
tl commands:			
V			
<code>\tl_if_empty:NTF</code>	7	<code>variable (env.)</code>	9, 7