l3doc 文档类 ——实验性质*

The LaTeX3 Project† 2023 年 12 月 11 日 发布

张泓知 2023 年 12 月 24 日 【译】

目 录

*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 LaTeX3 编程环境的模块或包的集合。最终它将取代 ltxdoc 类作为 LaTeX3 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 LaTeX 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 ltcmd 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
onlydoc
lm-default
kernel
check
checktest
show-notes
hide-notes
cs-break
cs-break-nohyphen

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。

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。

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。

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。）

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。

这些互补选项确定是否打印使用 \NB 和 \NOTE 命令提供的信息。

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。

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 "$\langle arg_{xy} \rangle$"。

\Arg `\Arg {⟨name⟩}`

\marg

\oarg

\parg

将 ⟨*name*⟩ 以 `\meta` 的方式排版，并用大括号包裹。

`\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" 方括号。

\file \pkg {⟨*name*⟩}
\env
\pkg 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。
\cls

\NB \NB {⟨*tag*⟩} {⟨*comments*⟩}
\NOTE \begin{NOTE} {⟨*tag*⟩}
⟨*comments*⟩
\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.*)
variable (*env.*)
syntax (*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 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 TEX 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 LATEX 2ε 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} {⟨template type name⟩}
  \TemplateArgument{none}{---}
或者一个或多个这些:
  \TemplateArgument {⟨arg no⟩} {⟨meaning⟩}
和
\TemplateSemantics
   ⟨text describing the template type semantics⟩
\end{TemplateInterfaceDescription}

\begin{TemplateDescription} {⟨template type name⟩} {⟨name⟩}
一个或多个这些:
  \TemplateKey {⟨key name⟩} {⟨type of key⟩}
     {⟨textual description of meaning⟩}
     {⟨default value if any⟩}
和
\TemplateSemantics
   ⟨text describing special additional semantics of the template⟩
\end{TemplateDescription}

\begin{InstanceDescription} [⟨text to specify key column width (optional)⟩]
          {⟨template type name⟩}{⟨instance name⟩}{⟨template name⟩}
一个或多个这些:
  \InstanceKey {⟨key name⟩} {⟨value⟩}
和
\InstanceSemantics
   ⟨text describing the result of this instance⟩
\end{InstanceDescription}
```

5 l3doc 代码实现

```
1 ⟨*class⟩
2 ⟨@@=codedoc⟩
```

5.1 变量

\g_docinput_clist 通过 \DocInput 输入的文件列表。

```
\clist_new:N \g_docinput_clist
```

(*\g_docinput_clist* 定义结束。这个变量被记录在第??页。)

\g_doc_functions_seq 通过 function 文档化的所有函数和通过 macro 引入的所有宏。可以进行比较，查
\g_doc_macros_seq 看文档或代码缺失的部分。

```
\seq_new:N \g_doc_functions_seq
\seq_new:N \g_doc_macros_seq
```

(*\g_doc_functions_seq* 和 *\g_doc_macros_seq* 定义结束。这些变量被记录在第??页。)

\l__codedoc_detect_internals_bool 如果为 true，l3doc 将检查在 macro 环境的参数中，以及在 macrocode 环境中排版
\l__codedoc_detect_internals_tl 的代码中使用其他包中的内部命令 __⟨*pkg*⟩_…，但不包括在 \cs 中。还有一个记载临时数据的记号列表用于此目的。

```
\bool_new:N \l__codedoc_detect_internals_bool
\bool_set_true:N \l__codedoc_detect_internals_bool
\tl_new:N \l__codedoc_detect_internals_tl
\tl_new:N \l__codedoc_detect_internals_cs_tl
```

(*\l__codedoc_detect_internals_bool* 和 *\l__codedoc_detect_internals_tl* 定义结束。)

\l__codedoc_output_coffin function 环境通过将包含各种部分（函数名称、描述、等等）的线槽组合成一个线槽来排版。

```
\coffin_new:N \l__codedoc_output_coffin
```

(*\l__codedoc_output_coffin* 定义结束。)

\l__codedoc_functions_coffin 这些线槽分别包含函数名称列表（function 环境的参数）、\begin{function} 和
\l__codedoc_descr_coffin \end{function} 之间的文本，以及在 syntax 环境中给出的语法。
\l__codedoc_syntax_coffin

```
\coffin_new:N \l__codedoc_functions_coffin
\coffin_new:N \l__codedoc_descr_coffin
\coffin_new:N \l__codedoc_syntax_coffin
```

(*\l__codedoc_functions_coffin*, *\l__codedoc_descr_coffin*, 和 *\l__codedoc_syntax_coffin* 定义结束。)

\g__codedoc_syntax_box 在转移到 \l__codedoc_syntax_coffin 之前，syntax 环境的内容被排版在这个盒子中。

```
\box_new:N \g__codedoc_syntax_box
```

(*\g__codedoc_syntax_box* 定义结束。)

\l__codedoc_in_function_bool 当在 function 或 variable 环境中时为真。被 syntax 环境用于确定其行为。

```
\bool_new:N \l__codedoc_in_function_bool
```

(*\l__codedoc_in_function_bool 定义结束。*)

\l__codedoc_long_name_bool
\l__codedoc_trial_width_dim
如果线棟 \l__codedoc_functions_coffin（包含当前函数名称）的宽度 \l__codedoc_trial_width_dim 大于边距中可用的空间，则布尔值 \l__codedoc_long_name_bool 为 true。

```
\bool_new:N \l__codedoc_long_name_bool
\dim_new:N \l__codedoc_trial_width_dim
```

(*\l__codedoc_long_name_bool 和 \l__codedoc_trial_width_dim 定义结束。*)

\l__codedoc_nested_macro_int macro 环境的嵌套层级（在 macro 环境外现在为 0）。

```
\int_new:N \l__codedoc_nested_macro_int
```

(*\l__codedoc_nested_macro_int 定义结束。*)

\l__codedoc_macro_tested_bool
\g__codedoc_missing_tests_prop
\g__codedoc_not_tested_seq
\g__codedoc_testfiles_seq
一个布尔值，描述当前宏是否有测试，并且一些全局结构包含了关于测试文件以及哪些测试缺失的信息。

```
\bool_new:N \l__codedoc_macro_tested_bool
\prop_new:N \g__codedoc_missing_tests_prop
\seq_new:N \g__codedoc_not_tested_seq
\seq_new:N \g__codedoc_testfiles_seq
```

(*\l__codedoc_macro_tested_bool 以及其它的定义结束。*)

\l__codedoc_macro_deprecated_bool
\l__codedoc_macro_internal_bool
\l__codedoc_macro_nodoc_bool
\l__codedoc_macro_TF_bool
\l__codedoc_macro_pTF_bool
\l__codedoc_macro_noTF_bool
\l__codedoc_macro_EXP_bool
\l__codedoc_macro_rEXP_bool
\l__codedoc_macro_var_bool
\l__codedoc_override_module_tl
\l__codedoc_macro_documented_tl
包含有关函数/宏环境某些选项的信息。我们初始化 \l__codedoc_override_module_tl 以避免通过空名称（表示无模块）覆盖模块名称。

```
\bool_new:N \l__codedoc_macro_deprecated_bool
\bool_new:N \l__codedoc_macro_internal_bool
\bool_new:N \l__codedoc_macro_nodoc_bool
\bool_new:N \l__codedoc_macro_TF_bool
\bool_new:N \l__codedoc_macro_pTF_bool
\bool_new:N \l__codedoc_macro_noTF_bool
\bool_new:N \l__codedoc_macro_EXP_bool
\bool_new:N \l__codedoc_macro_rEXP_bool
\bool_new:N \l__codedoc_macro_var_bool
\tl_new:N \l__codedoc_override_module_tl
\tl_set:Nn \l__codedoc_override_module_tl { \q_no_value }
\tl_new:N \l__codedoc_macro_documented_tl
```

(*\l__codedoc_macro_deprecated_bool 以及其它的定义结束。*)

\g__codedoc_lmodern_bool
\g__codedoc_checkfunc_bool
\g__codedoc_checktest_bool
\g__codedoc_cs_break_bool
\g__codedoc_show_notes_bool
\g__codedoc_kernel_bool

关于包选项的信息。

```
\bool_new:N \g__codedoc_lmodern_bool
\bool_new:N \g__codedoc_checkfunc_bool
\bool_new:N \g__codedoc_checktest_bool
\bool_new:N \g__codedoc_kernel_bool
\bool_new:N \g__codedoc_cs_break_bool
\bool_new:N \g__codedoc_show_notes_bool
\bool_gset_true:N \g__codedoc_cs_break_bool
```

(*\g__codedoc_lmodern_bool 以及其它的定义结束。*)

\l__codedoc_tmpa_tl
\l__codedoc_tmpb_tl
\l__codedoc_tmpa_int
\l__codedoc_tmpa_seq

一些临时变量。

```
\tl_new:N \l__codedoc_tmpa_tl
\tl_new:N \l__codedoc_tmpb_tl
\int_new:N \l__codedoc_tmpa_int
\int_new:N \l__codedoc_tmpa_seq
```

(*\l__codedoc_tmpa_tl 以及其它的定义结束。*)

\l__codedoc_names_block_tl

本地序列变量列表（通过 `\__codedoc_lseq_name:n` 生成），每个都对应于 `function` 或 `macro` 环境中一组变体。更准确地说，这些序列以基本形式命名，如 `\clist_count:n` 或 `\clist_count:N`（它们不是变体）。每个序列都有基本名称（没有任何签名）作为其第一项，后跟变体签名的列表，或者使用 `\scan_stop:` 表示无签名（无冒号）。

```
\tl_new:N \l__codedoc_names_block_tl
```

(*\l__codedoc_names_block_tl 定义结束。*)

\g__codedoc_variants_seq

暂时存储正在文档化的函数/宏的变体列表（仅签名）。它是全局的，因为我们需要在对齐的单元格中保持其值。

```
\seq_new:N \g__codedoc_variants_seq
```

(*\g__codedoc_variants_seq 定义结束。*)

\l__codedoc_names_verb_bool

如果宏/函数环境的主参数应该保留原样，不移除任何逗号或空格，则设置为 `true`。

```
\bool_new:N \l__codedoc_names_verb_bool
```

(*\l__codedoc_names_verb_bool 定义结束。*)

\l__codedoc_names_seq

函数/环境等作为给定 `function` 或 `macro` 环境参数出现的名称列表。这些名称经过 `_@@` 和 `@@` 转换为 `__`⟨*module name*⟩ 和其他清理。

```
\seq_new:N \l__codedoc_names_seq
```

(*\l__codedoc_names_seq* 定义结束。)

\g__codedoc_nested_names_seq

收集所有嵌套 macro 环境中的宏，以在“End definition”文本中使用。

```
\seq_new:N \g__codedoc_nested_names_seq
```

(*\g__codedoc_nested_names_seq* 定义结束。)

\l__codedoc_index_macro_tl
\l__codedoc_index_key_tl
\l__codedoc_index_module_tl
\l__codedoc_index_internal_bool
\l__codedoc_macro_do_not_index_tl

当分析 macrocode 环境中发现的控制序列时，\l__codedoc_index_macro_tl 保存控制序列（部分为字符串），\l__codedoc_index_key_tl 保存索引中的未来排序键，\l__codedoc_index_module_tl 是控制序列应该列入的子索引。\l__codedoc_index_internal_bool 表示控制序列是内部的，应在略微不同的子索引中索引。最后，\l__codedoc_macro_do_not_index_tl 表示不应在特定 macro 环境中索引的控制序列。

```
\tl_new:N \l__codedoc_index_macro_tl
\tl_new:N \l__codedoc_index_key_tl
\tl_new:N \l__codedoc_index_module_tl
\tl_new:N \l__codedoc_macro_do_not_index_tl
\bool_new:N \l__codedoc_index_internal_bool
```

(*\l__codedoc_index_macro_tl* 以及其它的定义结束。)

\g__codedoc_module_name_tl

模块名称，在读取行 <@@=⟨*module*⟩> 时设置。

```
\tl_new:N \g__codedoc_module_name_tl
```

(*\g__codedoc_module_name_tl* 定义结束。)

\c__codedoc_iow_rule_tl
\c__codedoc_iow_midrule_tl

40 个等号。

```
\tl_const:Nn \c__codedoc_iow_rule_tl
  { ======================================== }
\tl_const:Nn \c__codedoc_iow_mid_rule_tl
  { ---------------------------------------- }
```

(*\c__codedoc_iow_rule_tl* 和 *\c__codedoc_iow_midrule_tl* 定义结束。)

\l__codedoc_macro_box
\l__codedoc_macro_index_box
\l__codedoc_macro_int

用于储存宏环境中给定的名称的垂直盒子，储存索引命令创建的目标的水平盒子，以及到目前为止的宏数量（包括周围 macro 环境中的宏）。

```
\box_new:N \l__codedoc_macro_box
\box_new:N \l__codedoc_macro_index_box
\int_new:N \l__codedoc_macro_int
```

(*\l__codedoc_macro_box*, *\l__codedoc_macro_index_box*, 和 *\l__codedoc_macro_int* 定义结束。)

\l__codedoc_cmd_tl
\l__codedoc_cmd_index_tl
\l__codedoc_cmd_module_tl
\l__codedoc_cmd_noindex_bool
\l__codedoc_cmd_replace_bool

用于控制 \cmd、\cs 和 \tn 行为的变量。

```
\tl_new:N \l__codedoc_cmd_tl
\tl_new:N \l__codedoc_cmd_index_tl
\tl_new:N \l__codedoc_cmd_module_tl
\bool_new:N \l__codedoc_cmd_noindex_bool
\bool_new:N \l__codedoc_cmd_replace_bool
```

(*\l__codedoc_cmd_tl 以及其它的定义结束。*)

\l__codedoc_in_implementation_bool

在 implementation 环境内为 true，其他地方为 false。

```
\bool_new:N \l__codedoc_in_implementation_bool
```

(*\l__codedoc_in_implementation_bool 定义结束。*)

\g__codedoc_typeset_documentation_bool
\g__codedoc_typeset_implementation_bool

控制是否排版文档/实现的布尔值。默认情况下，两者都应为真。

```
\bool_new:N \g__codedoc_typeset_documentation_bool
\bool_new:N \g__codedoc_typeset_implementation_bool
\bool_set_true:N \g__codedoc_typeset_documentation_bool
\bool_set_true:N \g__codedoc_typeset_implementation_bool
```

(*\g__codedoc_typeset_documentation_bool 和 \g__codedoc_typeset_implementation_bool 定义结束。*)

\g__codedoc_base_name_tl
\l__codedoc_variants_prop

正在文档化的宏名称（不带签名），以及将变体的基本形式映射到具有相同基本形式的所有变体的属性列表。

```
\tl_new:N \g__codedoc_base_name_tl
\prop_new:N \l__codedoc_variants_prop
```

(*\g__codedoc_base_name_tl 和 \l__codedoc_variants_prop 定义结束。*)

\l__codedoc_function_label_clist
\l__codedoc_no_label_bool

function 环境的选项，用于替换给定控制序列列表的标签，通常情况下标签会被多次插入函数文档，这样可以避免重复标签。

```
\clist_new:N \l__codedoc_function_label_clist
\bool_new:N \l__codedoc_no_label_bool
```

(*\l__codedoc_function_label_clist 和 \l__codedoc_no_label_bool 定义结束。*)

\l__codedoc_date_added_tl
\l__codedoc_date_updated_tl

function 环境选项的值。

```
\tl_new:N \l__codedoc_date_added_tl
\tl_new:N \l__codedoc_date_updated_tl
```

(*\l__codedoc_date_added_tl 和 \l__codedoc_date_updated_tl 定义结束。*)

\l__codedoc_macro_argument_tl

保存 macro 或 function 环境的参数，以便在错误消息中使用。

```
\tl_new:N \l__codedoc_macro_argument_tl
```

(*\l__codedoc_macro_argument_tl 定义结束。*)

```
% \int_new:N \c@CodelineNo
```

5.2 变量和辅助函数

__codedoc_tmpa:w
__codedoc_tmpb:w

用于临时使用的辅助宏。

```
\cs_new_eq:NN \__codedoc_tmpa:w ?
\cs_new_eq:NN \__codedoc_tmpb:w ?
```

(*__codedoc_tmpa:w* 和 *__codedoc_tmpb:w* 定义结束。)

\seq_set_split:NoV
\tl_to_str:f

一些缺失的变体。

```
\cs_generate_variant:Nn \seq_set_split:Nnn { NoV }
\cs_generate_variant:Nn \tl_to_str:n { f }
```

(*\seq_set_split:NoV* 和 *\tl_to_str:f* 定义结束。这些函数被记录在第??页。)

__codedoc_if_almost_str:n*TF*

用于测试 \cmd 或其他要索引的宏的参数是否几乎是字符串：例如，如果 #1 包含 \meta{...}，则为 false。意外的 f 展开是为了处理以 \c_backslash_str 开头的 #1，应该被展开并视为“正常”。

```
\prg_new_protected_conditional:Npnn \__codedoc_if_almost_str:n #1 { TF , T , F }
  {
    \int_compare:nNnTF
      { \tl_count:n {#1} }
      < { \tl_count:e { \tl_to_str:f {#1} } }
      { \prg_return_false: }
      { \prg_return_true: }
  }
\prg_generate_conditional_variant:Nnn \__codedoc_if_almost_str:n { V } { T }
```

(*__codedoc_if_almost_str:nTF* 定义结束。)

__codedoc_trim_right:Nn
__codedoc_trim_right:No

在 token 列表变量 #1 中移除 #2 后的所有内容。可能与 __codedoc_key_trim_module:n 结合使用?

```
\cs_new_protected:Npn \__codedoc_trim_right:Nn #1#2
  {
    \cs_set:Npn \__codedoc_tmp:w ##1 #2 ##2 \q_stop { \exp_not:n {##1} }
    \__kernel_tl_set:Ne #1 { \exp_after:wN \__codedoc_tmp:w #1 #2 \q_stop }
  }
\cs_generate_variant:Nn \__codedoc_trim_right:Nn { No }
```

(*__codedoc_trim_right:Nn* 定义结束。)

__codedoc_str_if_begin:nn*TF*
__codedoc_str_if_begin:oo*TF*

如果第一个字符串以第二个字符串开头，则为真。

```
\prg_new_protected_conditional:Npnn \__codedoc_str_if_begin:nn #1#2 { TF , T , F }
  {
```

```
    \tl_if_in:ooTF
      { \exp_after:wN \scan_stop: \tl_to_str:n {#1} }
      { \exp_after:wN \scan_stop: \tl_to_str:n {#2} }
      { \prg_return_true: }
      { \prg_return_false: }
  }
\prg_generate_conditional_variant:Nnn \__codedoc_str_if_begin:nn
  { oo } { TF , T , F }
```

(*__codedoc_str_if_begin:nnTF 定义结束。*)

__codedoc_replace_at_at:N
__codedoc_replace_at_at_aux:Nn

目标是将 `@@` 替换为当前模块名称。我们利用此函数还可检测内部宏。如果没有 ⟨*module name*⟩，则不执行任何操作。否则，规范化 `@` 和 `_` 的分类码，临时将 `@@@@` 更改为具有不同分类码的 `aa`，稍后变为 `@@`，并将 `__@@`、`_@@` 和 `@@` 替换为 `__`⟨*module name*⟩。结果中包含带有字母分类码的 `_`，因为这是 `macrocode` 环境的预期。其他用例可通过 `\tl_to_str:n` 应用。注意，下面的代码中包括 `@` 之间的空格，因为它也经过相同的替换规则处理。

```
\cs_new_protected:Npn \__codedoc_replace_at_at:N #1
  {
    \tl_if_empty:NF \g__codedoc_module_name_tl
      {
        \exp_args:NNo \__codedoc_replace_at_at_aux:Nn
          #1 \g__codedoc_module_name_tl
      }
  }
\cs_new_protected:Npe \__codedoc_replace_at_at_aux:Nn #1#2
  {
    \tl_replace_all:Nnn #1 { \token_to_str:N @ } { @ }
    \tl_replace_all:Nnn #1 { \token_to_str:N _ } { _ }
    \tl_replace_all:Nnn #1 { @ @ @ @ } { \token_to_str:N a a }
    \tl_replace_all:Nnn #1 { _ _ @ @ } { _ _ #2 }
    \tl_replace_all:Nnn #1 {   _ @ @ } { _ _ #2 }
    \tl_replace_all:Nnn #1 {     @ @ } { _ _ #2 }
    \tl_replace_all:Nnn #1 { \token_to_str:N a a } { @ @ }
  }
```

(*__codedoc_replace_at_at:N 和 __codedoc_replace_at_at_aux:Nn 定义结束。*)

__codedoc_detect_internals:N
__codedoc_detect_internals_aux:N
__codedoc_if_detect_internals_ok:NF

在每个 `__` 处分割后，移除序列的开头项目（因为它不是跟在 `__` 后面），移除任何空格或行尾后的所有内容，以得到控制序列的良好近似（用于警告消息）。然后检查该序列是否以允许的内容开头：`@@` 模块名称和 `:` 或 `_`，或者相关的布尔值设置为

kernel_（似乎可以安全地假设我们不会定义 __kernel:... 命令）。对于消息本身，移除任何 _ 或 : 后的所有内容（具有任一分类码），以获得模块名称的猜测。

```
\cs_new_protected:Npn \__codedoc_detect_internals:N #1
  {
    \bool_if:NT \l__codedoc_detect_internals_bool
      { \__codedoc_detect_internals_aux:N #1 }
  }
\group_begin:
  \char_set_catcode_active:N \^^M
  \cs_new_protected:Npn \__codedoc_detect_internals_aux:N #1
    {
      \tl_set_eq:NN \l__codedoc_detect_internals_tl #1
      \tl_replace_all:NVn \l__codedoc_detect_internals_tl \c_underscore_str { _ }
      \seq_set_split:NnV \l__codedoc_tmpa_seq { _ _ } \l__codedoc_detect_internals_tl
      \seq_pop_left:NN \l__codedoc_tmpa_seq \l__codedoc_detect_internals_tl
      \seq_map_variable:NNn \l__codedoc_tmpa_seq \l__codedoc_detect_internals_tl
        {
          \__codedoc_trim_right:No \l__codedoc_detect_internals_tl
            \c_catcode_active_space_tl
          \__codedoc_trim_right:Nn \l__codedoc_detect_internals_tl ^^M
          \__codedoc_if_detect_internals_ok:NF \l__codedoc_detect_internals_tl
            {
              \tl_set_eq:NN \l__codedoc_detect_internals_cs_tl \l__codedoc_detect_internals_
              \__codedoc_trim_right:Nn \l__codedoc_detect_internals_tl _
              \__codedoc_trim_right:Nn \l__codedoc_detect_internals_tl :
              \__codedoc_trim_right:No \l__codedoc_detect_internals_tl { \token_to_str:N : }
              \msg_warning:nneee { l3doc } { foreign-internal }
                { \tl_to_str:N \l__codedoc_detect_internals_cs_tl }
                { \tl_to_str:N \l__codedoc_detect_internals_tl }
                { \tl_to_str:N \g__codedoc_module_name_tl }
            }
        }
    }
\group_end:
\prg_new_protected_conditional:Npnn \__codedoc_if_detect_internals_ok:N #1 { F }
  {
    \__codedoc_str_if_begin:ooTF {#1} { \g__codedoc_module_name_tl _ }
      { \prg_return_true: }
      {
        \__codedoc_str_if_begin:ooTF {#1} { \g__codedoc_module_name_tl : }
          { \prg_return_true: }
          {
```

```
            \bool_if:NTF \g__codedoc_kernel_bool
              {
                \__codedoc_str_if_begin:ooTF {#1} { kernel _ }
                  { \prg_return_true: }
                  { \prg_return_false: }
              }
              { \prg_return_false: }
          }
      }
  }
```

(*__codedoc_detect_internals:N*, *__codedoc_detect_internals_aux:N*, 和 *__codedoc_if_detect_internals_ok:NF* 定义结束。)

__codedoc_signature_base_form:n
__codedoc_signature_base_form_aux:n
__codedoc_signature_base_form_aux:w

扩展为签名的“基本形式”。例如，对于 noxcfvV，将得到 nnnNnnn，或对于 ow，将得到 nw。循环停在第一个无法识别的令牌处；其余部分被封装在 \exp_not:n 中。

```
\cs_new:Npn \__codedoc_signature_base_form:n #1
  { \__codedoc_signature_base_form_aux:n #1 \q_stop }
\cs_new:Npn \__codedoc_signature_base_form_aux:n #1
  {
    \str_case:nnTF {#1}
      {
        { N } { N }
        { c } { N }
        { n } { n }
        { o } { n }
        { f } { n }
        { e } { n }
        { x } { n }
        { V } { n }
        { v } { n }
      }
      { \__codedoc_signature_base_form_aux:n }
      { \__codedoc_signature_base_form_aux:w #1 }
  }
\cs_new:Npn \__codedoc_signature_base_form_aux:w #1 \q_stop
  { \exp_not:n {#1} }
```

(*__codedoc_signature_base_form:n*, *__codedoc_signature_base_form_aux:n*, 和 *__codedoc_signature_base_form_aux:w* 定义结束。)

__codedoc_predicate_from_base:n

从函数的基本名称获取谓词。代码不受没有签名的函数影响。n 类型的版本可用于键和其他非控制序列。在 e-展开后的输出是一个字符串。

```
\cs_new:Npn \__codedoc_predicate_from_base:n #1
  {
    \__codedoc_get_function_name:n {#1}
    \tl_to_str:n { _p: }
    \__codedoc_get_function_signature:n {#1}
  }
```

(__codedoc_predicate_from_base:n 定义结束。)

__codedoc_split_function_do:nn
__codedoc_split_function_do:on
__codedoc_get_function_name:n
__codedoc_get_function_signature:n
__codedoc_split_function_auxi:w
__codedoc_split_function_auxii:w

类似于 l3basics 中定义的内部函数，但在这里我们直接在字符串上操作，而不是控制序列。

```
\cs_new:Npn \__codedoc_get_function_name:n #1
  { \__codedoc_split_function_do:nn {#1} { \use_i:nnn } }
\cs_new:Npn \__codedoc_get_function_signature:n #1
  { \__codedoc_split_function_do:nn {#1} { \use_ii:nnn } }
\cs_set_protected:Npn \__codedoc_tmpa:w #1
  {
    \cs_new:Npn \__codedoc_split_function_do:nn ##1
      {
        \exp_after:wN \__codedoc_split_function_auxi:w
        \tl_to_str:n {##1} \q_mark \c_true_bool
        #1 \q_mark \c_false_bool
        \q_stop
      }
    \cs_new:Npn \__codedoc_split_function_auxi:w
      ##1 #1 ##2 \q_mark ##3##4 \q_stop ##5
      { \__codedoc_split_function_auxii:w {##5} ##1 \q_mark \q_stop {##2} ##3 }
    \cs_new:Npn \__codedoc_split_function_auxii:w
      ##1##2 \q_mark ##3 \q_stop
      { ##1 {##2} }
  }
\exp_args:No \__codedoc_tmpa:w { \token_to_str:N : }
\cs_generate_variant:Nn \__codedoc_split_function_do:nn { o }
```

(__codedoc_split_function_do:nn 以及其它的定义结束。)

__codedoc_key_get_base:nN

获取函数的基本形式并存储。作为获取基本形式的一部分，将尾部的 T 或 F 更改为 TF，如果函数不包含冒号，则跳过此更改，以避免更改一些以 PDF 结尾或类似结尾的名称。各种字母 z 作为不同于 \tl_to_str:n 任何结果的终止符。

```
\cs_new_protected:Npn \__codedoc_key_get_base:nN #1#2
  {
    \__codedoc_if_almost_str:nTF {#1}
      {
```

```
        \__codedoc_key_get_base_TF:nN {#1} \l__codedoc_tmpa_tl
        \__kernel_tl_set:Ne #2
          { \__codedoc_split_function_do:on \l__codedoc_tmpa_tl { \__codedoc_base_form_aux:n
      }
      { \tl_set:Nn #2 {#1} }
  }
\cs_new:Npe \__codedoc_key_get_base_TF:nN #1#2
  {
    \__kernel_tl_set:Ne #2 { \exp_not:N \tl_to_str:n {#1} }
    \tl_if_in:NoF #2 { \tl_to_str:n {:} }
      { \exp_not:N \prg_break: }
    \tl_if_in:onT { #2 z } { \tl_to_str:n {TF} z }
      { \exp_not:N \prg_break: }
    \tl_if_in:onT { #2 z } { \tl_to_str:n {T} z }
      {
        \tl_put_right:Nn #2 { \tl_to_str:n {F} }
        \exp_not:N \prg_break:
      }
    \tl_if_in:onT { #2 z } { \tl_to_str:n {F} z }
      {
        \tl_put_right:Nn #2 { z }
        \tl_replace_once:Nnn #2 { \tl_to_str:n {F} z } { \tl_to_str:n {TF} }
        \exp_not:N \prg_break:
      }
    \exp_not:N \prg_break_point:
  }
\cs_new:Npn \__codedoc_base_form_aux:nnN #1#2#3
  {
    \exp_not:n {#1}
    \bool_if:NT #3
      {
        \token_to_str:N :
        \bool_lazy_or:nnTF
            { \str_if_eq_p:nn { #1 ~ } { \exp_args } }
            { \str_if_eq_p:nn { #1 ~ } { \exp_last_unbraced } }
          { \exp_not:n {#2} }
          { \__codedoc_signature_base_form:n {#2} }
      }
  }
```

（*__codedoc_key_get_base:nN* 定义结束。）

__codedoc_base_form_signature_do:nnn

如果没有签名，或者 #1 中包含连续两个冒号（这包括奇怪的函数 \::N 等），则执行

#2{#1}。否则，应用 #3，并使用以下两个参数：#1 的基本形式，以及原始签名和额外的一对大括号。

```
\cs_new_protected:Npn \__codedoc_base_form_signature_do:nnn #1#2#3
  {
    \__codedoc_split_function_do:nn {#1}
      { \__codedoc_base_form_aux:nnnnnN {#1} {#2} {#3} }
  }
\cs_new_protected:Npn \__codedoc_base_form_aux:nnnnnN #1#2#3#4#5#6
  {
    \bool_if:NTF #6
      {
        \tl_if_head_eq_charcode:nNTF {#4} :
          { #2 {#1} }
          {
            \use:e
              {
                \exp_not:n {#3}
                { \__codedoc_base_form_aux:nnN {#4} {#5} #6 }
              }
                {#4} {#5}
          }
      }
      { #2 {#1} }
  }
```

（*__codedoc_base_form_signature_do:nnn* 定义结束。）

__codedoc_date_compare_p:nNn
__codedoc_date_compare:nNn*TF*
__codedoc_date_compare_aux:nnnNnnn
__codedoc_date_compare_aux:w

假设 #1 和 #3 是格式为 YYYY-MM-DD 的日期（也接受 YYYY 或 YYYY-MM）。使用 #2（其中之一为 <、= 或 >）进行比较，对缺失数据填充零。

```
\prg_new_conditional:Npnn \__codedoc_date_compare:nNn #1#2#3 { TF , T , F , p }
  { \__codedoc_date_compare_aux:w #1--- \q_mark #2 #3--- \q_stop }
\cs_new:Npn \__codedoc_date_compare_aux:w
    #1 - #2 - #3 - #4 \q_mark #5 #6 - #7 - #8 - #9 \q_stop
  {
    \__codedoc_date_compare_aux:nnnNnnn
      { \tl_if_empty:nTF {#1} { 0 } {#1} }
      { \tl_if_empty:nTF {#2} { 0 } {#2} }
      { \tl_if_empty:nTF {#3} { 0 } {#3} }
      #5
      { \tl_if_empty:nTF {#6} { 0 } {#6} }
      { \tl_if_empty:nTF {#7} { 0 } {#7} }
      { \tl_if_empty:nTF {#8} { 0 } {#8} }
```

```
  }
\cs_new:Npn \__codedoc_date_compare_aux:nnnNnnn #1#2#3#4#5#6#7
  {
    \int_compare:nNnTF {#1} = {#5}
      {
        \int_compare:nNnTF {#2} = {#6}
          {
            \int_compare:nNnTF {#3} #4 {#7}
              { \prg_return_true: } { \prg_return_false: }
          }
          {
            \int_compare:nNnTF {#2} #4 {#6}
              { \prg_return_true: } { \prg_return_false: }
          }
      }
      {
        \int_compare:nNnTF {#1} #4 {#5}
          { \prg_return_true: } { \prg_return_false: }
      }
    \use_none:n
    \q_stop
  }
```

(*__codedoc_date_compare:nNnTF*, *__codedoc_date_compare_aux:nnnNnnn*, 和 *__codedoc_date_compare_aux:w* 定义结束。)

__codedoc_gprop_name:n
__codedoc_lseq_name:n

我们需要跟踪一些有关正在（或已经）文档化的控制序列（和其他字符串）的信息。一些信息存储在全局属性中，一些存储在本地序列中，其名称不遵循惯例：它是 \g__codedoc 或 \l__codedoc，后面跟着一个空格和字符串，这个字符串可以是任意的。由于速度原因，我们不能合理地使用一个大的 prop。

```
\cs_new:Npn \__codedoc_gprop_name:n #1 { g__codedoc ~ \tl_to_str:n {#1} }
\cs_new:Npn \__codedoc_lseq_name:n #1 { l__codedoc ~ \tl_to_str:n {#1} }
```

(*__codedoc_gprop_name:n* 和 *__codedoc_lseq_name:n* 定义结束。)

5.3 信息

```
\msg_new:nnnn { l3doc } { no-signature-TF }
  { Function/macro~'#1'~cannot~be~turned~into~a~conditional. }
  {
    A~function~or~macro~environment~with~option~pTF,~TF~or~noTF~
    received~the~argument~'#1'.~This~function's~name~has~no~
    ':'~hence~it~is~not~clear~where~to~add~'_p'~or~'TF'.~
```

```
    Please~follow~expl3~naming~conventions.
  }
\msg_new:nnn { l3doc } { date-format }
  { The~date~'#1'~should~be~given~in~YYYY-MM-DD~format. }
\msg_new:nnn { l3doc } { future-date }
  { The~added/updated~date~'#2'~of~'#1'~is~in~the~future. }
\msg_new:nnn { l3doc } { syntax-nested-function }
  {
    The~'syntax'~environment~should~be~used~in~the~
    innermost~'function'~environment.
  }
\msg_new:nnn { l3doc } { multiple-syntax }
  {
    The~'syntax'~environment~should~only~be~used~once~in~
    a~'function'~environment.
  }
\msg_new:nnn { l3doc } { deprecated-option }
  { The~option~'#1'~has~been~deprecated~for~'#2'. }
\msg_new:nnn { l3doc } { foreign-internal }
  {
    A~control~sequence~of~the~form~'..._#1'~was~used.~
    It~should~only~be~used~in~the~module~'#2'
    \tl_if_empty:nF {#3} { ,~not~in~'#3' } .
  }
```

5.4 选项和配置

```
\DeclareKeys [ l3doc / options ]
  {
    a5paper .code:n = \@latexerr { Option~not~supported } { } ,
    full .code:n =
      {
        \bool_gset_true:N \g__codedoc_typeset_documentation_bool
        \bool_gset_true:N \g__codedoc_typeset_implementation_bool
      } ,
   onlydoc .code:n =
    {
      \bool_gset_true:N \g__codedoc_typeset_documentation_bool
      \bool_gset_false:N \g__codedoc_typeset_implementation_bool
    } ,
    check .bool_gset:N = \g__codedoc_checkfunc_bool ,
    checktest .bool_gset:N = \g__codedoc_checktest_bool ,
    kernel .bool_gset:N = \g__codedoc_kernel_bool ,
```

```
    stdmodule .bool_gset_inverse:N = \g__codedoc_kernel_bool ,
    lm-default .bool_gset:N = \g__codedoc_lmodern_bool ,
    cs-break .bool_gset_inverse:N = \g__codedoc_cs_break_bool ,
    cs-break-nohyphen .code:n = \PassOptionsToPackage{nohyphen}{underscore} ,
    show-notes .bool_gset:N = \g__codedoc_show_notes_bool,
    hide-notes .bool_gset_inverse:N = \g__codedoc_show_notes_bool
  }
\DeclareUnknownKeyHandler [ l3doc / options ]
  { \PassOptionsToClass { \CurrentOption } { article } }
\SetKeys [ l3doc / options ]
  { full , kernel , check = false , checktest = false , lm-default }
\PassOptionsToClass { a4paper } { article }
```

如果存在本地配置文件，则输入该文件，并向控制台输出已发生此操作的消息。因为我们与类一起分发了一个 `.cfg` 文件，所以通常情况下这应该是真的。因此，检查 `\ExplMakeTitle`（在我们的 `.cfg` 文件中定义），只有当找不到它时才输出信息性消息。

```
\msg_new:nnn { l3doc } { input-cfg }
  { Local~config~file~l3doc.cfg~loaded. }
\file_if_exist:nT { l3doc.cfg }
  {
    \file_input:n { l3doc.cfg }
    \cs_if_exist:cF { ExplMakeTitle }
      { \msg_info:nn { l3doc } { input-cfg } }
  }
\ProcessKeyOptions [ l3doc / options ]
```

5.5 类文件与宏包加载

```
\LoadClass{article}
\RequirePackage{doc}
\RequirePackage
  {
    array,
    alphalph,
    amsmath,
    amssymb,
    booktabs,
    color,
    colortbl,
    hologo,
    enumitem,
    pifont,
```

```
    textcomp,
    trace,
    csquotes,
    fancyvrb,
    underscore,
    verbatim
  }
\raggedbottom
```

根据选项加载 lmodern 包设置字体。然后将斜的打字机字体替换为斜的字形；前者让我感到不适。（Will, Aug 2011）

```
\bool_if:NT \g__codedoc_lmodern_bool
  {
    \RequirePackage[T1]{fontenc}
    \RequirePackage{lmodern}
    \group_begin:
      \ttfamily
      \DeclareFontShape{T1}{lmtt}{m}{it}{<->ec-lmtto10}{}
    \group_end:
  }
```

必须放在最后，像往常一样。

```
\RequirePackage{hypdoc}
```

5.6 配置和调整

\MakePrivateLetters

在 LaTeX3 编程环境中，还有一些字母是“私有的”。

```
\cs_gset:Npn \MakePrivateLetters
  {
    \char_set_catcode_letter:N \@
    \char_set_catcode_letter:N \_
    \char_set_catcode_letter:N \:
  }
```

(*\MakePrivateLetters* 定义结束。这个函数被记录在第??页。)

CodelineNo

与行编号有关的一些配置。

```
\setcounter{StandardModuleDepth}{1}
\@addtoreset{CodelineNo}{part}
\tl_replace_once:Nnn \theCodelineNo
  { \HDorg@theCodelineNo }
  { \textcolor[gray]{0.5} { \sffamily\tiny\arabic{CodelineNo} } }
```

(*CodelineNo* 定义结束。这个函数被记录在第??页。)

\verbatim
\endverbatim

在 .dtx 文档中，verbatim 环境会因为只删除第一个“%”符号，而不是缩进（通常是一个空格），导致额外的空间。用 fancyvrb 修复它：

```
\fvset{gobble=2}
\cs_gset_eq:NN \verbatim \Verbatim
\cs_gset_eq:NN \endverbatim \endVerbatim
```

(*\verbatim 和 \endverbatim 定义结束。这些函数被记录在第??页。*)

\ifnot@excluded

此函数测试存储在 \macro@namepart 中的宏名称是否被 \DoNotIndex 排除了索引。与其尝试修复传入的类别码，不如将所有内容转换为字符串类别码。这略微低效，因为我们本可以确保 \index@excludelist 一开始就是字符串类别码。

```
\cs_set_protected:Npn \ifnot@excluded
  {
    \exp_args:Nee \expanded@notin
      { \c_backslash_str \tl_to_str:N \macro@namepart , }
      { \exp_args:NV \tl_to_str:n \index@excludelist }
  }
```

(*\ifnot@excluded 定义结束。这个函数被记录在第??页。*)

\pdfstringnewline
__codedoc_pdfstring_newline:w

通过使书签中的 \\（几乎）变得微不足道，我们避免了一些 hyperref 警告。更确切地说，它可能与星号和可选参数一起使用，因此我们使用 ltcmd 可扩展命令将其删除。由于不能有尾随的可选参数，所以捡起额外的必选参数并将其放回。

```
\cs_new:Npn \pdfstringnewline { : ~ }
\DeclareExpandableDocumentCommand
  { \__codedoc_pdfstring_newline:w } { s o m } { \pdfstringnewline #3 }
\pdfstringdefDisableCommands
  { \cs_set_eq:NN \\ \__codedoc_pdfstring_newline:w }
```

(*\pdfstringnewline 和 __codedoc_pdfstring_newline:w 定义结束。这个函数被记录在第??页。*)

5.7 设计

稍微增加文本宽度，以便在 macrocode 环境中显示 72 列标准字体的代码。稍微增加边注宽度，以适应较长的命令名称。同时，将左边距增加相同的量。

```
\setlength    \textwidth       { 385pt }
\addtolength \marginparwidth {  30pt }
\addtolength \oddsidemargin  {  20pt }
\addtolength \evensidemargin {  20pt }
```

(这些设置是在使用 article 文档类时引入的，但我暂时保留它们，以便提醒自己稍后处理。)

\list 自定义列表。

__codedoc_oldlist:nn

```
\cs_new_eq:NN \__codedoc_oldlist:nn \list
\cs_gset:Npn \list #1 #2
  { \__codedoc_oldlist:nn {#1} { #2 \dim_zero:N \listparindent } }
\setlength \parindent  { 2em }
\setlength \itemindent { 0pt }
\setlength \parskip    { 0pt plus 3pt minus 0pt }
```

(*\list 和 __codedoc_oldlist:nn 定义结束。这个函数被记录在第??页。*)

\partname 在部 (Part) 标题中使用“File”作为名称。

```
\tl_gset:Nn \partname {File}
```

(*\partname 定义结束。这个函数被记录在第??页。*)

\l@section 自定义目录（因为有许多章节，需要不同的设计和/或结构)。

\l@subsection

```
\@addtoreset{section}{part}
\cs_gset:Npn \l@section #1#2
  {
    \ifnum \c@tocdepth >\z@
      \addpenalty\@secpenalty
      \addvspace{1.0em \@plus\p@}
      \setlength\@tempdima{2.5em}  % was 1.5em
      \begingroup
        \parindent \z@ \rightskip \@pnumwidth
        \parfillskip -\@pnumwidth
        \leavevmode \bfseries
        \advance\leftskip\@tempdima
        \hskip -\leftskip
        #1\nobreak\hfil \nobreak\hb@xt@\@pnumwidth{\hss #2}\par
      \endgroup
    \fi
  }
\cs_gset:Npn \l@subsection
  { \@dottedtocline{2}{2.5em}{2.3em} }  % #2 = 1.5em
```

(*\l@section 和 \l@subsection 定义结束。这些函数被记录在第??页。*)

5.8 文本标记

使 | 和 " 成为“短引用”字符，但不在文档导言部分使用，因为活动字符可能会干扰加载的包。在读取 .aux 文件之前，在文档末尾删除这些短引用，因为它们可能出现在标签中（例如，l3fp 中有一个“”的操作)。

```
\AtBeginDocument
  {
    \MakeShortVerb \"
    \MakeShortVerb \|
  }
\AtEndDocument
  {
    \DeleteShortVerb \"
    \DeleteShortVerb \|
  }
```

\eTeX 一些 logo 的命令。

\IniTeX
\Lua
\LuaTeX
\pdfTeX
\XeTeX
\pTeX
\upTeX
\epTeX
\eupTeX

```
\providecommand*\eTeX{\hologo{eTeX}}
\providecommand*\IniTeX{\hologo{iniTeX}}
\providecommand*\Lua{Lua}
\providecommand*\LuaTeX{\hologo{LuaTeX}}
\providecommand*\pdfTeX{\hologo{pdfTeX}}
\providecommand*\XeTeX{\hologo{XeTeX}}
\providecommand*\pTeX{p\kern-.2em\hologo{TeX}}
\providecommand*\upTeX{up\kern-.2em\hologo{TeX}}
\providecommand*\epTeX{$\varepsilon$-\pTeX}
\providecommand*\eupTeX{$\varepsilon$-\upTeX}
\providecommand*\ConTeXt{\hologo{ConTeXt}}
```

(*\eTeX* 以及其它的定义结束。这些函数被记录在第??页。)

\cmd
\cs
\tn

它们依赖于一个共同的辅助命令 `\__codedoc_cmd:nn`，接收选项和一些以反斜杠开头的标记的字符串表示（以支持诸如 `\cs{pkg_\ldots}` 这样的情况，我们不将整个参数转换为字符串）。

```
\DeclareDocumentCommand \cmd { O{} m }
  { \__codedoc_cmd:no {#1} { \token_to_str:N #2 } }
\DeclareDocumentCommand \cs  { O{} m }
  { \__codedoc_cmd:no {#1} { \c_backslash_str #2 } }
\DeclareDocumentCommand \tn  { O{} m }
  {
    \__codedoc_cmd:no
      { module = TeX , replace = false , #1 }
      { \c_backslash_str #2 }
  }
```

(*\cmd*, *\cs*, 和 *\tn* 定义结束。这些函数被记录在第7页。)

\meta 一个文档级别的命令。

```
\DeclareDocumentCommand \meta { m }
  { \__codedoc_meta:n {#1} }
```

(*\meta* 定义结束。这个函数被记录在第7页。)

__codedoc_pdfstring_cmd:w
__codedoc_pdfstring_cs:w
__codedoc_pdfstring_meta:w

为了在书签中工作，这些命令必须是可展开的。

```
\DeclareExpandableDocumentCommand
  { \__codedoc_pdfstring_cmd:w } { o m } { \token_to_str:N #2 }
\DeclareExpandableDocumentCommand
  { \__codedoc_pdfstring_cs:w }  { o m } { \textbackslash \tl_to_str:n {#2} }
\cs_new:Npn \__codedoc_pdfstring_meta:w #1
  { < \tl_to_str:n {#1} > }
\pdfstringdefDisableCommands
  {
    \cs_set_eq:NN \cmd  \__codedoc_pdfstring_cmd:w
    \cs_set_eq:NN \cs   \__codedoc_pdfstring_cs:w
    \cs_set_eq:NN \tn   \__codedoc_pdfstring_cs:w
    \cs_set_eq:NN \meta \__codedoc_pdfstring_meta:w
  }
```

(*__codedoc_pdfstring_cmd:w*, *__codedoc_pdfstring_cs:w*, 和 *__codedoc_pdfstring_meta:w* 定义结束。)

\Arg
\marg
\oarg
\parg

`\marg{text}` 输出 {⟨*text*⟩}, “必选参数”.
`\oarg{text}` 输出 [⟨*text*⟩], “可选参数”.
`\parg{te,xt}` 输出 (⟨*te,xt*⟩), “图形模式参数”. 最后，`\Arg` 与 `\marg` 相同。

```
\newcommand\Arg[1]
  { \texttt{\char`\{} \meta{#1} \texttt{\char`\}} }
\providecommand\marg[1]{ \Arg{#1} }
\providecommand\oarg[1]{ \texttt[ \meta{#1} \texttt] }
\providecommand\parg[1]{ \texttt( \meta{#1} \texttt) }
```

(*\Arg* 以及其它的定义结束。这些函数被记录在第7页。)

\file
\env
\pkg
\cls

这个列表可能会改变……这只是我对标记的偏好。

```
\DeclareRobustCommand \file {\nolinkurl}
\DeclareRobustCommand \env {\texttt}
\DeclareRobustCommand \pkg {\textsf}
\DeclareRobustCommand \cls {\textsf}
```

(*\file* 以及其它的定义结束。这些函数被记录在第8页。)

\EnableDocumentation
\EnableImplementation
\DisableDocumentation
\DisableImplementation

控制是否排版文档/实现。这些只是设定了两个开关。

```
\NewDocumentCommand \EnableDocumentation { }
  { \bool_gset_true:N \g__codedoc_typeset_documentation_bool }
\NewDocumentCommand \EnableImplementation { }
  { \bool_gset_true:N \g__codedoc_typeset_implementation_bool }
\NewDocumentCommand \DisableDocumentation { }
  { \bool_gset_false:N \g__codedoc_typeset_documentation_bool }
\NewDocumentCommand \DisableImplementation { }
  { \bool_gset_false:N \g__codedoc_typeset_implementation_bool }
```

(*\EnableDocumentation* 以及其它的定义结束。这些函数被记录在第??页。)

documentation (*env.*)
implementation (*env.*)

如果需要排版文档/实现, 只需设置布尔值 \l__codedoc_in_implementation_bool, 指示是否在实现部分。否则使用 \comment（和配对的 \endcomment）。

```
\NewDocumentEnvironment { documentation } { }
  {
    \bool_if:NTF \g__codedoc_typeset_documentation_bool
      { \bool_set_false:N \l__codedoc_in_implementation_bool }
      { \comment }
  }
  { \bool_if:NF \g__codedoc_typeset_documentation_bool { \endcomment } }
\NewDocumentEnvironment { implementation } { }
  {
    \bool_if:NTF \g__codedoc_typeset_implementation_bool
      { \bool_set_true:N \l__codedoc_in_implementation_bool }
      { \comment }
  }
  { \bool_if:NF \g__codedoc_typeset_implementation_bool { \endcomment } }
```

variable (*env.*)

variable 环境根据文档部分的不同行为类似于 function 或 macro 环境。

```
\DeclareDocumentEnvironment { variable } { O{} +v }
  {
    \bool_if:NTF \l__codedoc_in_implementation_bool
      { \__codedoc_macro:nnw { var , #1 } {#2} }
      { \__codedoc_function:nnw {#1} {#2} }
  }
  {
    \bool_if:NTF \l__codedoc_in_implementation_bool
      { \__codedoc_macro_end: }
      { \__codedoc_function_end: }
  }
```

function (*env.*) 用于记录函数的环境，以及用于记录宏实现的环境。

macro (*env.*)

```
\DeclareDocumentEnvironment { function } { O{} +v }
  { \__codedoc_function:nnw {#1} {#2} }
  { \__codedoc_function_end: }
\DeclareDocumentEnvironment { macro } { O{} +v }
  { \__codedoc_macro:nnw {#1} {#2} }
  { \__codedoc_macro_end: }
```

syntax (*env.*) 语法块放置在函数列表旁边，用于说明它们的用法。TODO：测试 syntax 环境是否仅在 function 环境内使用，且仅出现一次。

```
\NewDocumentEnvironment { syntax } { }
  { \__codedoc_syntax:w }
  {
    \__codedoc_syntax_end:
    \ignorespacesafterend
  }
```

texnote (*env.*) 用于描述仅供 TEX 专家阅读的信息。

```
\NewDocumentEnvironment { texnote } { }
  {
    \endgraf
    \vspace{3mm}
    \small\textbf{\TeX~hackers~note:}
  }
  {
    \vspace{3mm}
  }
```

arguments (*env.*) 此环境设计用于在 macro 环境中描述宏/函数的参数。

```
\NewDocumentEnvironment { arguments } { }
  {
    \enumerate [
      nolistsep ,
      label = \texttt{\#\arabic*} ~ : ,
      labelsep = * ,
    ]
  }
  {
    \endenumerate
  }
```

\CodedocExplain
\CodedocExplainEXP
\CodedocExplainREXP
\CodedocExplainTF

解释星号和 TF 符号的用法，供第三方包使用。

```
\NewDocumentCommand { \CodedocExplain } { }
  { \CodedocExplainEXP \ \CodedocExplainREXP \ \CodedocExplainTF }
\NewDocumentCommand { \CodedocExplainEXP } { }
  {
    \raisebox{\baselineskip}[0pt][0pt]{\hypertarget{expstar}{}}%
    \write \@auxout { \def \string \Codedoc@expstar { } }
    \__codedoc_typeset_exp:\ indicates~fully~expandable~functions,~which~
    can~be~used~within~an~\texttt{e}-type~argument~(inside~an~\tn{expanded}),~
    \texttt{x}-type~argument~(in~plain~\TeX{}~terms,~inside~an~\tn{edef}),~
    as~well~as~within~an~\texttt{f}-type~argument.
  }
\NewDocumentCommand { \CodedocExplainREXP } { }
  {
    \raisebox{\baselineskip}[0pt][0pt]{\hypertarget{rexpstar}{}}%
    \write \@auxout { \def \string \Codedoc@rexpstar { } }
    \__codedoc_typeset_rexp:\ indicates~
    restricted~expandable~functions,~which~can~be~used~within~an~
    \texttt{x}-type~argument~or~an~\texttt{e}-type~argument,~
    but~cannot~be~fully~expanded~within~an~\texttt{f}-type~argument.
  }
\NewDocumentCommand { \CodedocExplainTF } { }
  {
    \raisebox{\baselineskip}[0pt][0pt]{\hypertarget{explTF}{}}%
    \write \@auxout { \def \string \Codedoc@explTF { } }
    \__codedoc_typeset_TF:\ indicates~conditional~(\texttt{if})~functions~
    whose~variants~with~\texttt{T},~\texttt{F}~and~\texttt{TF}~
    argument~specifiers~expect~different~
    \enquote{true}/\enquote{false}~branches.
  }
```

(*\CodedocExplain 以及其它的定义结束。这些函数被记录在第??页。*)

5.9 实现文本标记

\cmd、\cs和\tn的键。

```
\keys_define:nn { l3doc/cmd }
  {
    index     .tl_set:N     = \l__codedoc_cmd_index_tl        ,
    module    .tl_set:N     = \l__codedoc_cmd_module_tl       ,
    no-index  .bool_set:N   = \l__codedoc_cmd_noindex_bool    ,
    replace   .bool_set:N   = \l__codedoc_cmd_replace_bool    ,
```

```
}
```

__codedoc_cmd:nn
__codedoc_cmd:no

在设置一些默认值后应用键-值 ⟨*options*⟩ #1。然后（除非 replace=false）替换 @@ 中的 #2，这有点棘手：_ 必须给出 __codedoc_replace_at_at:N 期望的类别码，但应该在不重新扫描整个参数的情况下恢复到其原始的类别码（通常是活动的，需要进行换行）。然后，在将其转换为无害字符后（保持下划线可断行），将命令设置为 \verbatim@font 中进行排版；在任何情况下，空格必须转换为 \@xobeysp，我们必须使用 \@ 来避免在控制序列之后产生更长的空格，例如以冒号结尾的控制序列（空签名）。最后，生成索引条目。当 \l__codedoc_cmd_noindex_bool 为 true 时，索引被抑制。

```
\cs_new_protected:Npn \__codedoc_cmd:nn #1#2
  {
    \bool_set_false:N \l__codedoc_cmd_noindex_bool
    \bool_set_true:N \l__codedoc_cmd_replace_bool
    \tl_set:Nn \l__codedoc_cmd_index_tl { \q_no_value }
    \tl_set:Nn \l__codedoc_cmd_module_tl { \q_no_value }
    \keys_set:nn { l3doc/cmd } {#1}
    \tl_set:Nn \l__codedoc_cmd_tl {#2}
    \bool_if:NT \l__codedoc_cmd_replace_bool
      {
        \tl_set_rescan:Nnn \l__codedoc_tmpb_tl { } { _ }
        \tl_replace_all:NVn \l__codedoc_cmd_tl \l__codedoc_tmpb_tl { _ }
        \__codedoc_replace_at_at:N \l__codedoc_cmd_tl
        \tl_replace_all:NnV \l__codedoc_cmd_tl { _ } \l__codedoc_tmpb_tl
      }
```

排版。注意下划线的替换是为了允许换行。underscore 宏包添加了换行，正则表达式导致仅将可断行的下划线应用于下划线的最后一个，而不是下划线跟在反斜杠后面的情况。

```
    \mode_if_math:T { \mbox }
      {
        \bool_if:NT \l__codedoc_allow_indexing_bool { \__codedoc_target: }
        \verbatim@font
        \__codedoc_if_almost_str:VT \l__codedoc_cmd_tl
          {
            \__kernel_tl_set:Ne \l__codedoc_cmd_tl { \tl_to_str:N \l__codedoc_cmd_tl }
            \bool_if:NT \g__codedoc_cs_break_bool
              {
                \regex_replace_all:nnN
                  { ([^\\\_]\_*) \_ ([^\_]) }
                  { \1 \c{BreakableUnderscore} \2 }
```

```
                \l__codedoc_cmd_tl
              }
          }
        \tl_replace_all:Nnn \l__codedoc_cmd_tl { ~ } { \@xobeysp }
        \l__codedoc_cmd_tl
        \@
      }
```

索引。

```
    \bool_if:NT \l__codedoc_allow_indexing_bool
     {
      \bool_if:NF \l__codedoc_cmd_noindex_bool
       {
        \quark_if_no_value:NF \l__codedoc_cmd_index_tl
          {
            \__kernel_tl_set:Ne \l__codedoc_cmd_tl
              { \c_backslash_str \exp_not:o { \l__codedoc_cmd_index_tl } }
          }
        \exp_args:No \__codedoc_key_get:n { \l__codedoc_cmd_tl }
        \quark_if_no_value:NF \l__codedoc_cmd_module_tl
          {
            \__kernel_tl_set:Ne \l__codedoc_index_module_tl
              { \tl_to_str:N \l__codedoc_cmd_module_tl }
          }
        \__codedoc_special_index_module:ooonN
          { \l__codedoc_index_key_tl }
          { \l__codedoc_index_macro_tl }
          { \l__codedoc_index_module_tl }
          { usage }
          \l__codedoc_index_internal_bool
       }
     }
  }
\cs_generate_variant:Nn \__codedoc_cmd:nn { no }
```

(*__codedoc_cmd:nn* 定义结束。)

__codedoc_meta:n
__codedoc_ensuremath_sb:n
__codedoc_meta_original:n

将 #1 存储在 \l__codedoc_tmpa_tl 中，并替换每个下划线，无论其类别是什么（“math toggle”、“alignment”、“superscript”、“subscript”、“letter”、“other” 或 “active”），都用 __codedoc_ensuremath_sb:n 替换（这会创建数学下标），然后运行 doc.sty 中用于 \meta 的代码。

```
\cs_new_protected:Npn \__codedoc_meta:n #1
  {
```

```
    \tl_set:Nn \l__codedoc_tmpa_tl {#1}
    \tl_map_inline:nn
      { { 3 } { 4 } { 7 } { 8 } { 11 } { 12 } { 13 } }
      {
        \tl_set_rescan:Nnn \l__codedoc_tmpb_tl
          { \char_set_catcode:nn { `_ } {##1} } { _ }
        \tl_replace_all:NVn \l__codedoc_tmpa_tl \l__codedoc_tmpb_tl
          { \__codedoc_ensuremath_sb:n }
      }
    \exp_args:NV \__codedoc_meta_original:n \l__codedoc_tmpa_tl
  }
\cs_new_protected:Npn \__codedoc_ensuremath_sb:n #1
  { \ensuremath { \sb {#1} } }
\cs_new_protected:Npn \__codedoc_meta_original:n #1
  {
    \ensuremath \langle
    \mode_if_math:T { \nfss@text }
    {
      \meta@font@select
      \edef \meta@hyphen@restore
        { \hyphenchar \the \font \the \hyphenchar \font }
      \hyphenchar \font \m@ne
      \language \l@nohyphenation
      #1 \/
      \meta@hyphen@restore
    }
    \ensuremath \rangle
  }
```

(__codedoc_meta:n, __codedoc_ensuremath_sb:n, 和 __codedoc_meta_original:n 定义结束。)

5.9.1 macro 和 function 之间共通的部分

__codedoc_typeset_exp:
__codedoc_typeset_rexp:
__codedoc_typeset_TF:
__codedoc_typeset_aux:n

被 __codedoc_macro_single:nNN 使用，也在 function 环境中用于排版条件语句和辅助函数。

```
\cs_new_protected:Npn \__codedoc_typeset_exp:
  {
    \cs_if_exist:NTF \Codedoc@expstar
      { \hyperlink { expstar } }
      { \mbox }
    {$\star$}
  }
\cs_new_protected:Npn \__codedoc_typeset_rexp:
```

```
  {
    \cs_if_exist:NTF \Codedoc@rexpstar
      { \hyperlink { rexpstar } }
      { \mbox }
    { \ding { 73 } } % hollow star
  }
\cs_new_protected:Npn \__codedoc_typeset_TF:
  {
    \cs_if_exist:NTF \Codedoc@explTF
      { \hyperlink { explTF } }
      { \mbox }
      {
        \color{black}
        \itshape TF
        \makebox[0pt][r]
          {
            \cs_if_exist:NT \Codedoc@explTF { \color{red} }
            \underline { \phantom{\itshape TF} \kern-0.1em }
          }
      }
  }
\cs_new_protected:Npn \__codedoc_typeset_aux:n #1
  {
    { \color[gray]{0.5} #1 }
  }
```

（*__codedoc_typeset_exp:* 以及其它的定义结束。）

__codedoc_get_hyper_target:nN
__codedoc_get_hyper_target:oN
__codedoc_get_hyper_target:eN

从宏名称 #1 创建一个 hyperref 锚点，并将其存储在记号列表变量 #2 中。例如，\prg_replicate:nn 变成 doc/function//prg/replicate:nn。

```
\cs_new_protected:Npn \__codedoc_get_hyper_target:nN #1#2
  {
    \__kernel_tl_set:Ne #2 { \tl_to_str:n {#1} }
    \tl_replace_all:NVn #2 \c_underscore_str { / }
    \tl_remove_all:NV   #2 \c_backslash_str
    \tl_put_left:Nn #2 { doc/function// }
  }
\cs_generate_variant:Nn \__codedoc_get_hyper_target:nN { o , e }
```

（*__codedoc_get_hyper_target:nN* 定义结束。）

__codedoc_names_get_seq:nN

参数 #1（在 function 或 macro 环境的参数）的类别码为 10（空格），12（其他）和 13（活动）。对类别码进行清理。如果使用了 verb 选项，则输出一个单项序列。否

则，移除位于行首的任何 “%” 字符。移除制表符和换行符。最后，将 _@@ 和 @@ 转换为 __⟨*module name*⟩（如果非空）。此时，\l__codedoc_tmpa_tl 包含一个逗号分隔的名称列表，其中 @ 和 _ 的类别码为字母。将其转换为字符串，解析为逗号分隔的列表（特别是移除了空格），并输出一系列函数/宏名称。

```
\cs_new_protected:Npn \__codedoc_names_get_seq:nN #1#2
  {
    \__kernel_tl_set:Ne \l__codedoc_tmpa_tl { \tl_to_str:n {#1} }
    \bool_if:NTF \l__codedoc_names_verb_bool
      {
        \seq_clear:N #2
        \seq_put_right:NV #2 \l__codedoc_tmpa_tl
      }
      {
        \tl_remove_all:Ne \l__codedoc_tmpa_tl
          { \iow_char:N \^^M \c_percent_str }
        \tl_remove_all:Ne \l__codedoc_tmpa_tl { \tl_to_str:n { ^ ^ A } }
        \tl_remove_all:Ne \l__codedoc_tmpa_tl { \iow_char:N \^^I }
        \tl_remove_all:Ne \l__codedoc_tmpa_tl { \iow_char:N \^^M }
        \__codedoc_detect_internals:N \l__codedoc_tmpa_tl
        \__codedoc_replace_at_at:N \l__codedoc_tmpa_tl
        \exp_args:NNe \seq_set_from_clist:Nn #2
          { \tl_to_str:N \l__codedoc_tmpa_tl }
      }
  }
```

（*__codedoc_names_get_seq:nN* 定义结束。）

__codedoc_names_parse:
__codedoc_names_parse_one:n

目标是将变体组合在一起。我们使用 __codedoc_lseq_name:n 在基本形式后创建局部序列变量，并将其填充到 \l__codedoc_names_block_tl 中。当遇到新的基本形式时，设置相应的局部序列来保存 ⟨*base name*⟩（去除签名），并将局部序列添加到列表 \l__codedoc_names_block_tl 中。在所有情况下，将签名附加到局部序列中，因此它采用了这样的形式：⟨*base name*⟩、⟨$signature_1$⟩、⟨$signature_2$⟩ 等等。如果原始函数没有签名（没有冒号），则使用 \scan_stop: 作为签名（没有变体）。我们特别处理以 \:: 开头的命令 #1，即命名为 \::N 等的奇怪函数。

```
\cs_new_protected:Npn \__codedoc_names_parse:
  {
    \tl_clear:N \l__codedoc_names_block_tl
    \seq_map_function:NN
      \l__codedoc_names_seq
      \__codedoc_names_parse_one:n
  }
```

```
\cs_new_protected:Npn \__codedoc_names_parse_one:n #1
  {
    \__codedoc_split_function_do:nn {#1}
      { \__codedoc_names_parse_one_aux:nnNn }
    {#1}
  }
\cs_new_protected:Npn \__codedoc_names_parse_one_aux:nnNn #1#2#3#4
  {
    \bool_if:NTF #3
      {
        \tl_if_head_eq_charcode:nNTF {#2} :
          { \__codedoc_names_parse_aux:nnn {#4} {#4} { \scan_stop: } }
          {
            \exp_args:Ne \__codedoc_names_parse_aux:nnn
              { \__codedoc_base_form_aux:nnN {#1} {#2} #3 }
              {#1} {#2}
          }
      }
      {
        \bool_if:NT \l__codedoc_macro_TF_bool
          { \msg_error:nne { l3doc } { no-signature-TF } {#4} }
        \__codedoc_names_parse_aux:nnn {#4} {#4} { \scan_stop: }
      }
  }
\cs_new_protected:Npn \__codedoc_names_parse_aux:nnn #1
  { \exp_args:Nc \__codedoc_names_parse_aux:Nnn { \__codedoc_lseq_name:n {#1} } }
\cs_new_protected:Npn \__codedoc_names_parse_aux:Nnn #1#2#3
  {
    \tl_if_in:NnF \l__codedoc_names_block_tl {#1}
      {
        \tl_put_right:Nn \l__codedoc_names_block_tl {#1}
        \seq_clear_new:N #1
        \seq_put_right:Nn #1 {#2}
      }
    \seq_put_right:Nn #1 {#3}
  }
```

（*__codedoc_names_parse:* 和 *__codedoc_names_parse_one:n* 定义结束。）

__codedoc_names_typeset:
__codedoc_names_typeset_auxi:n

此代码特别用于在 function 环境中排版函数名称时。对于 \l__codedoc_names_block_tl 的映射不能使用 \tl_map_inline:Nn，因为跟在 \\ 之后的代码将不可展开，因此会破坏 \bottomrule。

对每个局部序列（其中包含一组变体）调用 `\__codedoc_names_typeset_auxi:n`。第一步是弹出基本形式，并将空格更改为其他类别码，以便最终显示出来。然后将变体存储在 `\g__codedoc_variants_seq` 中，移除第一个（它将更显眼地显示），并重建实际名称，将其传递给 `\__codedoc_names_typeset_auxii:n`。

```
\cs_new_protected:Npn \__codedoc_names_typeset:
  {
    \tl_map_function:NN \l__codedoc_names_block_tl
      \__codedoc_names_typeset_auxi:n
  }
\cs_new_protected:Npn \__codedoc_names_typeset_auxi:n #1
  {
    \seq_pop:NN #1 \l__codedoc_tmpa_tl
    \tl_gset_eq:NN \g__codedoc_base_name_tl \l__codedoc_tmpa_tl
    \tl_greplace_all:NnV \g__codedoc_base_name_tl
      { ~ } \c_catcode_other_space_tl
    \seq_get:NN #1 \l__codedoc_tmpa_tl
    \str_if_eq:VnTF \l__codedoc_tmpa_tl { \scan_stop: }
      {
        \seq_gclear:N \g__codedoc_variants_seq
        \__codedoc_names_typeset_auxii:e { \g__codedoc_base_name_tl }
      }
      {
        \seq_gset_eq:NN \g__codedoc_variants_seq #1
        \seq_gpop:NN \g__codedoc_variants_seq \l__codedoc_tmpb_tl
        \__codedoc_names_typeset_auxii:e
          { \g__codedoc_base_name_tl : \l__codedoc_tmpb_tl }
      }
  }
```

（*__codedoc_names_typeset: 和 __codedoc_names_typeset_auxi:n 定义结束。*）

`\__codedoc_names_typeset_auxii:n`
`\__codedoc_names_typeset_auxii:e`

如果给定选项 `pTF`，在 `TF` 函数之前排版谓词。如果给定选项 `noTF`，也排版非 `TF` 函数。在这两种情况下都传递相关的布尔值，控制是否追加 `TF`。

```
\cs_new_protected:Npn \__codedoc_names_typeset_auxii:n #1
  {
    \bool_if:NT \l__codedoc_macro_pTF_bool
      {
        \__codedoc_names_typeset_block:eN
          { \__codedoc_predicate_from_base:n {#1} }
          \c_false_bool
      }
    \bool_if:NT \l__codedoc_macro_noTF_bool
```

```
      { \__codedoc_names_typeset_block:nN {#1} \c_false_bool }
    \__codedoc_names_typeset_block:nN {#1} \l__codedoc_macro_TF_bool
  }
\cs_generate_variant:Nn \__codedoc_names_typeset_auxii:n { e }
```

(*__codedoc_names_typeset_auxii:n* 定义结束。)

__codedoc_names_typeset_block:nN
__codedoc_names_typeset_block:eN

function 和 macro 环境中的名称排版方式不同。要区分这两者，请注意当处于 macro 环境时 \l__codedoc_nested_macro_int 至少为 1 （我们假设 function 不会嵌套在其中）。一个块是包含其所有变体的函数。

```
\cs_new_protected:Npn \__codedoc_names_typeset_block:nN
  {
    \int_compare:nNnTF \l__codedoc_nested_macro_int = 0
      { \__codedoc_typeset_function_block:nN }
      { \__codedoc_macro_typeset_block:nN }
  }
\cs_generate_variant:Nn \__codedoc_names_typeset_block:nN { e }
```

(*__codedoc_names_typeset_block:nN* 定义结束。)

__codedoc_if_macro_internal_p:n
__codedoc_if_macro_internal:nTF
__codedoc_if_macro_internal_aux:w

确定给定的宏应视为内部还是公共。如果给出诸如 int 的选项，则答案是 \l__codedoc_macro_internal_bool，否则检查宏名称是否包含 __。

```
\prg_new_conditional:Npnn \__codedoc_if_macro_internal:n #1 { p , T , F , TF }
  {
    \bool_if:NTF \l__codedoc_macro_internal_bool
      { \prg_return_true: }
      {
        \tl_if_empty:eTF
          {
            \exp_after:wN \__codedoc_if_macro_internal_aux:w
            \tl_to_str:n { #1 ~ __ }
          }
          { \prg_return_false: } { \prg_return_true: }
      }
  }
\exp_last_unbraced:NNNNo
  \cs_new:Npn \__codedoc_if_macro_internal_aux:w #1 { \tl_to_str:n { __ } } { }
```

(*__codedoc_if_macro_internal:nTF* 和 *__codedoc_if_macro_internal_aux:w* 定义结束。)

__codedoc_names_block_base_map:N

\l__codedoc_names_block_tl 包含与不同基本函数及其变体对应的序列变量。对于每个这样的序列，将第一个和第二个项目放入 \l__codedoc_tmpa_tl 和 \l__codedoc_tmpb_tl 中，并构建基本函数的名称。

```
\cs_new_protected:Npn \__codedoc_names_block_base_map:N #1
  {
    \tl_map_inline:Nn \l__codedoc_names_block_tl
      {
        \group_begin:
          \seq_set_eq:NN \l__codedoc_tmpa_seq ##1
          \seq_pop:NN \l__codedoc_tmpa_seq \l__codedoc_tmpa_tl
          \seq_get:NN \l__codedoc_tmpa_seq \l__codedoc_tmpb_tl
          \exp_args:NNe
        \group_end:
        #1
          {
            \l__codedoc_tmpa_tl
            \str_if_eq:VnF \l__codedoc_tmpb_tl { \scan_stop: }
              { : \l__codedoc_tmpb_tl }
            \bool_if:NT \l__codedoc_macro_TF_bool { TF }
          }
      }
  }
```

(*__codedoc_names_block_base_map:N 定义结束。*)

5.9.2 function 环境

```
\keys_define:nn { l3doc/function }
  {
    TF .value_forbidden:n = true ,
    TF .code:n =
      {
        \bool_set_true:N \l__codedoc_macro_TF_bool
      } ,
    EXP .value_forbidden:n = true ,
    EXP .code:n =
      {
        \bool_set_true:N \l__codedoc_macro_EXP_bool
        \bool_set_false:N \l__codedoc_macro_rEXP_bool
      } ,
    rEXP .value_forbidden:n = true ,
    rEXP .code:n =
      {
        \bool_set_false:N \l__codedoc_macro_EXP_bool
        \bool_set_true:N \l__codedoc_macro_rEXP_bool
      } ,
```

```
    pTF .value_forbidden:n = true ,
    pTF .code:n =
      {
        \bool_set_true:N \l__codedoc_macro_pTF_bool
        \bool_set_true:N \l__codedoc_macro_TF_bool
        \bool_set_true:N \l__codedoc_macro_EXP_bool
        \bool_set_false:N \l__codedoc_macro_rEXP_bool
      } ,
    noTF .value_forbidden:n = true ,
    noTF .code:n =
      {
        \bool_set_true:N \l__codedoc_macro_noTF_bool
        \bool_set_true:N \l__codedoc_macro_TF_bool
      } ,
    added .code:n = { \__codedoc_date_set_past:Nn \l__codedoc_date_added_tl {#1} },
    updated .code:n = { \__codedoc_date_set_past:Nn \l__codedoc_date_updated_tl {#1} } ,
    deprecated .bool_set:N = \l__codedoc_macro_deprecated_bool ,
    no-user-doc .bool_set:N = \l__codedoc_macro_nodoc_bool ,
    tested .code:n = { } ,
    label .code:n =
      {
        \clist_set:Nn \l__codedoc_function_label_clist {#1}
        \bool_set_true:N \l__codedoc_no_label_bool
      } ,
    verb .value_forbidden:n = true ,
    verb .bool_set:N = \l__codedoc_names_verb_bool ,
    module .tl_set:N = \l__codedoc_override_module_tl ,
  }
```

__codedoc_date_set:Nn
__codedoc_date_set_past:Nn

将日期标准化为格式 YYYY-MM-DD；更精确地说，允许月份和日期为单个数字。函数 __codedoc_date_set_past:Nn 仅允许过去（或同一天）的日期。

```
\cs_new_protected:Npn \__codedoc_date_set:Nn #1#2
  {
    \tl_set:Nn #1 {#2}
    \regex_replace_once:nnNF
      { \A(\d\d\d\d)[-/](\d\d?)[-/](\d\d?)\Z } { \1-\2-\3 } #1
      {
        \msg_error:nnn { l3doc } { date-format } {#2}
        \tl_set:Nn #1 { 1970-01-01 }
      }
  }
\cs_new_protected:Npn \__codedoc_date_set_past:Nn #1#2
  {
```

```
    \__codedoc_date_set:Nn #1 {#2}
    \exp_args:No \__codedoc_date_compare:nNnT
      {#1} > { \c_sys_year_int - \c_sys_month_int - \c_sys_day_int }
      {
        \msg_error:nnee { l3doc } { future-date }
          { \tl_to_str:N \l__codedoc_macro_argument_tl }
          {#1}
      }
  }
```

(*__codedoc_date_set:Nn* 和 *__codedoc_date_set_past:Nn* 定义结束。)

__codedoc_function:nnw

#1： 键-值列表。

#2： 逗号分隔的函数列表；输入在读取参数之前已经通过改变类别码进行了清理。

__codedoc_function_end:

确保任何段落都已结束，并确保环境开始时类似的安全实践。初始化一些变量。解析键-值列表。清理函数列表，然后遍历其中提取一些数据。在此之后，在存放函数名的装订盒 `\l__codedoc_functions_coffin` 中排版函数名称，并测量其大小以确定是否适合边距。最后，开始一个垂直装订盒用于环境的主要部分。当环境结束时，此装订盒停止，然后所有部分被组装成一个单一的装订盒，并进行排版。

```
\cs_new_protected:Npn \__codedoc_function:nnw #1#2
  {
    \__codedoc_function_typeset_start:
    \__codedoc_function_init:
    \tl_set:Nn \l__codedoc_macro_argument_tl {#2}
    \keys_set:nn { l3doc/function } {#1}
    \__codedoc_names_get_seq:nN {#2} \l__codedoc_names_seq
    \__codedoc_names_parse:
    \__codedoc_function_typeset:
    \__codedoc_function_reset:
    \__codedoc_function_descr_start:w
  }
\cs_new_protected:Npn \__codedoc_function_end:
  {
    \__codedoc_function_descr_stop:
    \__codedoc_function_assemble:
    \__codedoc_function_typeset_stop:
  }
```

(*__codedoc_function:nnw* 和 *__codedoc_function_end:* 定义结束。)

__codedoc_function_typeset_start:
__codedoc_function_typeset_stop:

在 `function` 环境开始之前，在执行任何赋值之前，关闭上一个段落，并设置排版场景。进一步的代码排版一个装订盒，因此我们结束段落并允许换页。

```
\cs_new_protected:Npn \__codedoc_function_typeset_start:
  {
    \par \bigskip \noindent
  }
\cs_new_protected:Npn \__codedoc_function_typeset_stop:
  {
    \par
    \dim_set:Nn \prevdepth { \box_dp:N \l__codedoc_descr_coffin }
    \allowbreak
  }
```

(__codedoc_function_typeset_start: 和 __codedoc_function_typeset_stop: 定义结束。)

__codedoc_function_init: 如果嵌套了 function 环境，则发出警告。清除各种变量。

```
\cs_new_protected:Npn \__codedoc_function_init:
  {
    \box_if_empty:NF \g__codedoc_syntax_box
      { \msg_error:nn { l3doc } { syntax-nested-function } }
    \coffin_clear:N \l__codedoc_descr_coffin
    \box_gclear:N \g__codedoc_syntax_box
    \coffin_clear:N \l__codedoc_syntax_coffin
    \coffin_clear:N \l__codedoc_functions_coffin
    \bool_set_false:N \l__codedoc_macro_TF_bool
    \bool_set_false:N \l__codedoc_macro_pTF_bool
    \bool_set_false:N \l__codedoc_macro_noTF_bool
    \bool_set_false:N \l__codedoc_macro_EXP_bool
    \bool_set_false:N \l__codedoc_macro_rEXP_bool
    \bool_set_false:N \l__codedoc_no_label_bool
    \bool_set_false:N \l__codedoc_names_verb_bool
    \bool_set_true:N \l__codedoc_in_function_bool
    \clist_clear:N \l__codedoc_function_label_clist
    \tl_set:Nn \l__codedoc_override_module_tl { \q_no_value }
    \char_set_active_eq:NN \< \__codedoc_shorthand_meta:
    \char_set_catcode_active:N \<
  }
```

(__codedoc_function_init: 定义结束。)

__codedoc_shorthand_meta: 允许使用 <...> 作为 \meta{...} 的标记。

__codedoc_shorthand_meta:w

```
\cs_new_protected:Npn \__codedoc_shorthand_meta:
  { \mode_if_math:TF { < } { \__codedoc_shorthand_meta:w } }
\cs_new_protected_nopar:Npn \__codedoc_shorthand_meta:w #1 > { \meta {#1} }
```

(__codedoc_shorthand_meta: 和 __codedoc_shorthand_meta:w 定义结束。)

__codedoc_function_reset: 清除一些变量。

```
\cs_new_protected:Npn \__codedoc_function_reset:
  {
    \tl_set:Nn \l__codedoc_override_module_tl { \q_no_value }
  }
```

(*__codedoc_function_reset: 定义结束。*)

__codedoc_function_typeset: 在装订盒 \l__codedoc_functions_coffin 中排版 \l__codedoc_names_block_tl 中列出的函数以及相关日期，然后如果此装订盒大于边距中可用的宽度，则将 \l__codedoc_long_name_bool 设置为 true。函数 __codedoc_typeset_functions: 相当复杂，因此稍后给出。

```
\cs_new_protected:Npn \__codedoc_function_typeset:
  {
    \dim_zero:N \l__codedoc_trial_width_dim
    \hcoffin_set:Nn \l__codedoc_functions_coffin { \__codedoc_typeset_functions: }
    \dim_set:Nn \l__codedoc_trial_width_dim
      { \box_wd:N \l__codedoc_functions_coffin }
    \bool_set:Nn \l__codedoc_long_name_bool
      { \dim_compare_p:nNn \l__codedoc_trial_width_dim > \marginparwidth }
  }
```

(*__codedoc_function_typeset: 定义结束。*)

__codedoc_function_descr_start:w
__codedoc_function_descr_stop: __codedoc_function:nnw 的最后一步（function 环境的开始）是打开一个装订盒，用于捕获函数的描述，即 function 环境的正文。这由 __codedoc_function_end: （function 环境的结束）关闭。

```
\cs_new_protected:Npn \__codedoc_function_descr_start:w
  {
    \vcoffin_set:Nnw \l__codedoc_descr_coffin { \textwidth }
      \noindent \ignorespaces
  }
\cs_new_protected:Npn \__codedoc_function_descr_stop:
  { \vcoffin_set_end: }
```

(*__codedoc_function_descr_start:w 和 __codedoc_function_descr_stop: 定义结束。*)

__codedoc_function_assemble: \g__codedoc_syntax_box 包含语法环境的内容（如果使用了）。现在我们有了所有的部分，将语法装订盒、名称装订盒和描述装订盒组合在一起。相对位置取决于名称装订盒是否适合边距。然后排版组合内容。

```
\cs_new_protected:Npn \__codedoc_function_assemble:
  {
```

```
    \hcoffin_set:Nn  \l__codedoc_syntax_coffin
      { \box_use_drop:N \g__codedoc_syntax_box }
    \bool_if:NTF \l__codedoc_long_name_bool
      {
        \coffin_join:NnnNnnnn
          \l__codedoc_output_coffin {hc} {vc}
          \l__codedoc_syntax_coffin {l} {T}
          {0pt} {0pt}
        \coffin_join:NnnNnnnn
          \l__codedoc_output_coffin {l} {t}
          \l__codedoc_functions_coffin  {r} {t}
          {-\marginparsep} {0pt}
        \coffin_join:NnnNnnnn
          \l__codedoc_output_coffin {l} {b}
          \l__codedoc_descr_coffin  {l} {t}
          {0.75\marginparwidth + \marginparsep} {-\medskipamount}
        \coffin_typeset:Nnnnn \l__codedoc_output_coffin
          {\l__codedoc_descr_coffin-l} {\l__codedoc_descr_coffin-t}
          {0pt} {0pt}
      }
      {
        \coffin_join:NnnNnnnn
          \l__codedoc_output_coffin {hc} {vc}
          \l__codedoc_syntax_coffin {l} {t}
          {0pt} {0pt}
        \coffin_join:NnnNnnnn
          \l__codedoc_output_coffin {l} {b}
          \l__codedoc_descr_coffin  {l} {t}
          {0pt} {-\medskipamount}
        \coffin_join:NnnNnnnn
          \l__codedoc_output_coffin {l} {t}
          \l__codedoc_functions_coffin  {r} {t}
          {-\marginparsep} {0pt}
        \coffin_typeset:Nnnnn \l__codedoc_output_coffin
          {\l__codedoc_syntax_coffin-l} {\l__codedoc_syntax_coffin-T}
          {0pt} {0pt}
      }
  }
```

(*__codedoc_function_assemble:* 定义结束。)

__codedoc_typeset_functions:

此函数通过在 tabular 环境中排版函数名称（带有变体）和相关日期来构建 \l__codedoc_functions_coffin。使用 \toprule、\midrule 和 \bottomrule 的规则要

求最后一个 \\ 和规则之间的任何内容都是可展开的，这使得我们的工作变得有些复杂。

```
\cs_new_protected:Npn \__codedoc_typeset_functions:
  {
    \small\ttfamily
    \__codedoc_target:
    \Hy@MakeCurrentHref { HD. \int_use:N \c@HD@hypercount }
    \begin{tabular} [t] { @{} l @{} >{\hspace{\tabcolsep}} r @{} }
      \toprule
      \__codedoc_function_extra_labels:
      \__codedoc_names_typeset:
      \__codedoc_typeset_dates:
      \bottomrule
    \end{tabular}
    \normalfont\normalsize
  }
```

(*__codedoc_typeset_functions:* 定义结束。)

__codedoc_typeset_function_block:nN
__codedoc_typeset_function_block:eN
__codedoc_function_index:n
__codedoc_function_index:e

#1 是一个控制序列名称，#2 是一个布尔值，指示是否添加 TF。

```
\cs_new_protected:Npn \__codedoc_typeset_function_block:nN #1#2
  {
    \__codedoc_function_index:e
      { #1 \bool_if:NT #2 { \tl_to_str:n {TF} } }
    \__codedoc_function_label:eN {#1} #2
    #1
    \bool_if:NT #2 { \__codedoc_typeset_TF: }
    \__codedoc_typeset_expandability:
    \seq_if_empty:NF \g__codedoc_variants_seq
      { \__codedoc_typeset_variant_list:nN {#1} #2 }
    \\
  }
\cs_generate_variant:Nn \__codedoc_typeset_function_block:nN { e }
\cs_new_protected:Npn \__codedoc_function_index:n #1
  {
    \seq_gput_right:Nn \g_doc_functions_seq {#1}
    \__codedoc_special_index:nn {#1} { usage }
  }
\cs_generate_variant:Nn \__codedoc_function_index:n { e }

\cs_new_protected:Npn \__codedoc_typeset_expandability:
  {
    &
```

```
    \bool_if:NT \l__codedoc_macro_EXP_bool  { \__codedoc_typeset_exp: }
    \bool_if:NT \l__codedoc_macro_rEXP_bool { \__codedoc_typeset_rexp: }
  }
```

#1 是函数，#2 是否添加 TF。

```
\cs_new_protected:Npn \__codedoc_typeset_variant_list:nN #1#2
  {
    \\
    \__codedoc_typeset_aux:n { \__codedoc_get_function_name:n {#1} }
    :
    \int_compare:nTF { \seq_count:N \g__codedoc_variants_seq == 1 }
      { \seq_use:Nn \g__codedoc_variants_seq { } }
      {
        \hbox_set:Nn \l_tmpa_box
          { \seq_use:Nn \g__codedoc_variants_seq { \textrm| \nolinebreak[2] } }
        \textrm(
```

设置长变体列表在一个段落盒子中，短列表设置自然长度。

```
        \dim_compare:nNnTF { \box_wd:N \l_tmpa_box } > { .4\columnwidth }
          {
            \parbox[t]{.4\columnwidth}
              {
                \raggedright
                \hbox_unpack_drop:N \l_tmpa_box
                \textrm)
                \bool_if:NT #2 { \__codedoc_typeset_TF: }
              }
          }
          {
            \hbox_unpack_drop:N \l_tmpa_box
            \textrm)
            \bool_if:NT #2 { \__codedoc_typeset_TF: }
          }
      }
    \__codedoc_typeset_expandability:
  }
```

#1 是函数名，#2 是否添加 TF。

```
\cs_new_protected:Npn \__codedoc_function_extra_labels:
  {
    \bool_if:NT \l__codedoc_no_label_bool
      {
        \clist_map_inline:Nn \l__codedoc_function_label_clist
          {
```

```
            \__codedoc_get_hyper_target:oN { \token_to_str:N ##1 }
              \l__codedoc_tmpa_tl
            \exp_args:No \label { \l__codedoc_tmpa_tl }
          }
      }
  }
\cs_new_protected:Npn \__codedoc_function_label:nN #1#2
  {
    \bool_if:NF \l__codedoc_no_label_bool
      {
        \__codedoc_get_hyper_target:eN
          {
            \exp_not:n {#1}
            \bool_if:NT #2 { \tl_to_str:n {TF} }
          }
          \l__codedoc_tmpa_tl
        \exp_args:No \label { \l__codedoc_tmpa_tl }
      }
  }
\cs_generate_variant:Nn \__codedoc_function_label:nN { e }
```

(__codedoc_typeset_function_block:nN 和 __codedoc_function_index:n 定义结束。)

__codedoc_typeset_dates: 用于显示函数添加/修改时的元数据。此函数必须是可展开的，因为它生成用于对齐的规则。

```
\cs_new:Npn \__codedoc_typeset_dates:
  {
    \bool_lazy_and:nnF
      { \tl_if_empty_p:N \l__codedoc_date_added_tl }
      { \tl_if_empty_p:N \l__codedoc_date_updated_tl }
      { \midrule }
    \tl_if_empty:NF \l__codedoc_date_added_tl
      {
        \multicolumn { 2 } { @{} r @{} }
          { \scriptsize New: \, \l__codedoc_date_added_tl } \\
      }

    \tl_if_empty:NF \l__codedoc_date_updated_tl
      {
        \multicolumn { 2 } { @{} r @{} }
          { \scriptsize Updated: \, \l__codedoc_date_updated_tl } \\
      }
  }
```

(__codedoc_typeset_dates: 定义结束。)

__codedoc_syntax:w
__codedoc_syntax_end:

实现 syntax 环境。

```
\dim_new:N \l__codedoc_syntax_dim
\cs_new_protected:Npn \__codedoc_syntax:w
  {
    \box_if_empty:NF \g__codedoc_syntax_box
      { \msg_error:nn { l3doc } { multiple-syntax } }
    \dim_set:Nn \l__codedoc_syntax_dim
      {
        \textwidth
        \bool_if:NT \l__codedoc_long_name_bool
          { + 0.75 \marginparwidth - \l__codedoc_trial_width_dim }
      }
    \hbox_gset:Nw \g__codedoc_syntax_box
      \small \ttfamily
      \arrayrulecolor{white}
      \begin{tabular} { @{} l @{} }
        \toprule
        \begin{minipage}[t]{\l__codedoc_syntax_dim}
          \raggedright
          \obeyspaces
          \obeylines
  }
\cs_new_protected:Npn \__codedoc_syntax_end:
  {
        \end{minipage}
      \end{tabular}
      \arrayrulecolor{black}
    \hbox_gset_end:
    \bool_if:NF \l__codedoc_in_function_bool
      {
        \begin{quote}
          \mode_leave_vertical:
          \box_use_drop:N \g__codedoc_syntax_box
        \end{quote}
      }
  }
```

(__codedoc_syntax:w 和 __codedoc_syntax_end: 定义结束。)

5.9.3 macro 环境

macro 环境的键值。TODO: 提供文档命令以记录键。

```
\keys_define:nn { l3doc/macro }
  {
    aux .value_forbidden:n = true ,
    aux .code:n =
      {
        \msg_warning:nnnn { l3doc } { deprecated-option }
          { aux } { function/macro }
      } ,
    deprecated .bool_set:N = \l__codedoc_macro_deprecated_bool ,
    internal .value_forbidden:n = true ,
    internal .code:n =
      { \bool_set_true:N \l__codedoc_macro_internal_bool } ,
    int .value_forbidden:n = true ,
    int .code:n =
      { \bool_set_true:N \l__codedoc_macro_internal_bool } ,
    no-user-doc .bool_set:N = \l__codedoc_macro_nodoc_bool ,
    var .value_forbidden:n = true ,
    var .code:n =
      { \bool_set_true:N \l__codedoc_macro_var_bool } ,
    TF .value_forbidden:n = true ,
    TF .code:n =
      { \bool_set_true:N \l__codedoc_macro_TF_bool } ,
    pTF .value_forbidden:n = true ,
    pTF .code:n =
      {
        \bool_set_true:N \l__codedoc_macro_TF_bool
        \bool_set_true:N \l__codedoc_macro_pTF_bool
        \bool_set_true:N \l__codedoc_macro_EXP_bool
        \bool_set_false:N \l__codedoc_macro_rEXP_bool
      } ,
    noTF .value_forbidden:n = true ,
    noTF .code:n =
      {
        \bool_set_true:N \l__codedoc_macro_TF_bool
        \bool_set_true:N \l__codedoc_macro_noTF_bool
      } ,
    EXP .value_forbidden:n = true ,
    EXP .code:n =
      {
```

```
      \bool_set_true:N \l__codedoc_macro_EXP_bool
      \bool_set_false:N \l__codedoc_macro_rEXP_bool
    } ,
  rEXP .value_forbidden:n = true ,
  rEXP .code:n =
    {
      \bool_set_false:N \l__codedoc_macro_EXP_bool
      \bool_set_true:N \l__codedoc_macro_rEXP_bool
    } ,
  tested .code:n =
    {
      \bool_set_true:N \l__codedoc_macro_tested_bool
    } ,
  added .code:n = {} , % TODO
  updated .code:n = {} , % TODO
  verb .bool_set:N = \l__codedoc_names_verb_bool ,
  module .tl_set:N = \l__codedoc_override_module_tl ,
  documented-as .tl_set:N = \l__codedoc_macro_documented_tl ,
  do-not-index .value_required:n = true ,
  do-not-index .tl_set:N = \l__codedoc_macro_do_not_index_tl ,
  % do-not-index .default:n = \q_no_value ,
}
```

__codedoc_macro:nnw

参数是一个 ⟨选项⟩ 的键-值列表和一个逗号分隔的 ⟨名称⟩ 列表，由 ltcmd 以逐字方式读取。在应用 ⟨选项⟩ 之前，先初始化一些变量，然后解析 ⟨名称⟩ 以获取宏名称序列，然后对每个应用 __codedoc_macro_single:nNN（这一步比较微妙，因为涉及到 TF/pTF/noTF）。最后在边缘上排版宏名称。

```
\cs_new_protected:Npn \__codedoc_macro:nnw #1#2
  {
    \__codedoc_macro_init:
    \tl_set:Nn \l__codedoc_macro_argument_tl {#2}
    \keys_set:nn { l3doc/macro } {#1}
    \__codedoc_names_get_seq:nN {#2} \l__codedoc_names_seq
    \__codedoc_names_parse:
    \__codedoc_macro_exclude_index:
    \__codedoc_macro_save_names:
    \__codedoc_names_typeset:
    \__codedoc_macro_dump:
    \__codedoc_macro_reset:
  }
```

(*__codedoc_macro:nnw 定义结束。*)

__codedoc_macro_init: 布尔值保存各种键-值选项，\l__codedoc_nested_macro_int 计算当前点周围的 macro 环境数量（在外部为 0）。

```
\cs_new_protected:Npn \__codedoc_macro_init:
  {
    \int_incr:N \l__codedoc_nested_macro_int
    \bool_set_false:N \l__codedoc_macro_deprecated_bool
    \bool_set_false:N \l__codedoc_macro_internal_bool
    \bool_set_false:N \l__codedoc_macro_TF_bool
    \bool_set_false:N \l__codedoc_macro_pTF_bool
    \bool_set_false:N \l__codedoc_macro_noTF_bool
    \bool_set_false:N \l__codedoc_macro_EXP_bool
    \bool_set_false:N \l__codedoc_macro_rEXP_bool
    \bool_set_false:N \l__codedoc_macro_var_bool
    \bool_set_false:N \l__codedoc_macro_tested_bool
    \bool_set_false:N \l__codedoc_names_verb_bool
    \tl_set:Nn \l__codedoc_override_module_tl { \q_no_value }
    \tl_clear:N \l__codedoc_macro_documented_tl
    \cs_set_eq:NN \testfile \__codedoc_print_testfile:n
    \box_clear:N \l__codedoc_macro_index_box
    \vbox_set:Nn \l__codedoc_macro_box
      {
        \hbox:n
          {
            \strut
            \int_compare:nNnT \l__codedoc_macro_int = 0 { \__codedoc_target: }
          }
        \vskip \int_eval:n { \l__codedoc_macro_int - 1 } \baselineskip
      }
  }
```

(*__codedoc_macro_init: 定义结束。*)

__codedoc_macro_reset: 确保被强制模块的宏内的 \cs 命令不受影响。

```
\cs_new_protected:Npn \__codedoc_macro_reset:
  {
    \tl_set:Nn \l__codedoc_override_module_tl { \q_no_value }
  }
```

(*__codedoc_macro_reset: 定义结束。*)

__codedoc_macro_save_names: 在一组 macro 环境中定义的名称列表最终用于显示其文档的页面。如果给出了 documented-as 键，则使用该键，否则在 \l__codedoc_names_block_tl 中查找名称。

```
\cs_new_protected:Npn \__codedoc_macro_save_names:
  {
    \tl_if_empty:NTF \l__codedoc_macro_documented_tl
      { \__codedoc_names_block_base_map:N \__codedoc_macro_save_names_aux:n }
      {
        \seq_gput_right:Ne \g__codedoc_nested_names_seq
          { \tl_to_str:N \l__codedoc_macro_documented_tl }
      }
  }
\cs_new_protected:Npn \__codedoc_macro_save_names_aux:n #1
  { \seq_gput_right:Nn \g__codedoc_nested_names_seq {#1} }
```

(*__codedoc_macro_save_names:* 定义结束。)

__codedoc_macro_exclude_index:

在 macrocode 环境中的某些控制序列不应该被索引，原因各不相同。此宏解析 do-not-index 选项的参数，并从索引中局部移除给定的宏。

macro 的可选参数不使用逐字的类别码扫描,所以我们使用 \tl_set_rescan:NnV 以与 \DoNotIndex 相同的类别码重新扫描命令。扫描的标记列表包含控制序列后面的空格，在使用 \DoNotIndex 时不存在。由于 \seq_set_from_clist:Nn 移除了项目周围的空格，我们可以滥用这一点和 \seq_use:Nn 来规范化每个项目。然后 \DoNotIndex 就可以起作用了。

```
\cs_new_protected:Npn \__codedoc_macro_exclude_index:
  {
    \tl_if_empty:NF \l__codedoc_macro_do_not_index_tl
      {
        \tl_set_rescan:NnV \l__codedoc_macro_do_not_index_tl
          { \MakePrivateLetters \catcode`\\12 }
          \l__codedoc_macro_do_not_index_tl
        \exp_args:NNV \seq_set_from_clist:Nn
          \l__codedoc_tmpa_seq \l__codedoc_macro_do_not_index_tl
        \__kernel_tl_set:Ne \l__codedoc_macro_do_not_index_tl
          { \seq_use:Nn \l__codedoc_tmpa_seq { , } }
        \exp_args:NV \DoNotIndex \l__codedoc_macro_do_not_index_tl
      }
  }
```

(*__codedoc_macro_exclude_index:* 定义结束。)

__codedoc_macro_dump:

它调用了 \makelabel{}

```
\cs_new_protected:Npn \__codedoc_macro_dump:
  {
    \topsep\MacroTopsep
```

```
    \trivlist
    \cs_set:Npn \makelabel ##1
      {
        \llap
          {
            \hbox_unpack_drop:N \l__codedoc_macro_index_box
            \vtop to \baselineskip
              {
                \vbox_unpack_drop:N \l__codedoc_macro_box
                \vss
              }
          }
      }
    \item [ ]
  }
```

(__codedoc_macro_dump: 定义结束。)

__codedoc_macro_typeset_block:nN 用于排版宏及其变体。#1 是宏名称，#2 是控制是否添加TF的布尔值。

```
\cs_new_protected:Npn \__codedoc_macro_typeset_block:nN #1#2
  {
    \__codedoc_macro_single:nNN {#1} \c_true_bool #2
    \seq_if_empty:NF \g__codedoc_variants_seq
      {
        \__codedoc_macro_typeset_variant_list:eN
          { \__codedoc_get_function_name:n {#1} } #2
      }
  }
\cs_new_protected:Npn \__codedoc_macro_typeset_variant_list:nN #1#2
  {
    \seq_map_inline:Nn \g__codedoc_variants_seq
      { \__codedoc_macro_single:nNN { #1 : ##1 } \c_false_bool #2 }
  }
\cs_generate_variant:Nn \__codedoc_macro_typeset_variant_list:nN { e }
```

(__codedoc_macro_typeset_block:nN 定义结束。)

__codedoc_macro_single:nNN 参数是 #1（不含TF的宏名称），#2 是确定是否索引的布尔值，#3 是是否添加TF的布尔值。让我们开始操作 doc 的 macro 环境。参见 doc.dtx 以了解原始环境的详细解释。它被相当“热情地”注释了。

#1： 宏/函数/等名称；输入已经经过了净化。

对 \saved@macroname 和 \saved@indexname 的赋值是为了 doc 的 \changes 机制。

```
\cs_new_protected:Npn \__codedoc_macro_single:nNN #1#2#3
  {
    \tl_set:Nn \saved@macroname {#1}
    \__codedoc_macro_typeset_one:nN {#1} #3
    \bool_if:NT #3 { \DoNotIndex {#1} }
    \exp_args:Ne \__codedoc_macro_index:nN
      { #1 \bool_if:NT #3 { \tl_to_str:n { TF } } }
      #2
  }
\cs_new_protected:Npn \__codedoc_macro_index:nN #1#2
  {
    \DoNotIndex {#1}
    \bool_if:NT #2
      {
        \bool_lazy_any:nF
          {
            { \__codedoc_if_macro_internal_p:n {#1} }
            { \l__codedoc_macro_deprecated_bool }
            { \l__codedoc_macro_nodoc_bool }
          }
          { \seq_gput_right:Nn \g_doc_macros_seq {#1} }
        \hbox_set:Nw \l__codedoc_macro_index_box
          \hbox_unpack_drop:N \l__codedoc_macro_index_box
          \int_gincr:N \c@CodelineNo
          \__codedoc_special_index:nn {#1} { main }
          \int_gdecr:N \c@CodelineNo
        \exp_args:NNNo \hbox_set_end:
          \tl_set:Nn \saved@indexname { \l__codedoc_index_key_tl }
      }
  }
```

(__codedoc_macro_single:nNN 定义结束。)

__codedoc_macro_typeset_one:nN

长时间以来，l3doc 将宏名称作为嵌套的 \trivlist 中的标签收集起来，但是这些并没有用 \endtrivlist 正确关闭。而且，它与 hyperref 的目标产生了令人惊讶的交互方式。现在，我们手动在盒子 \l__codedoc_macro_box 中收集排版的宏名称。固定大小的空格 \MacroFont\ 可以被可定制的水平空格所取代；对所有宏而言，重要的是它们都是相同的。#1 是宏名称，#2 是否添加 TF。

```
\cs_new_protected:Npn \__codedoc_macro_typeset_one:nN #1#2
  {
    \vbox_set:Nn \l__codedoc_macro_box
```

```
      {
        \vbox_unpack_drop:N \l__codedoc_macro_box
        \hbox { \llap { \__codedoc_print_macroname:nN {#1} #2
            \MacroFont \
        } }
      }
    \int_incr:N \l__codedoc_macro_int
  }
```

(*__codedoc_macro_typeset_one:nN* 定义结束。)

__codedoc_print_macroname:nN 在名称中，空格被替换为其他空格，以确保它们能够显示（如果有的话）。

```
\cs_new_protected:Npn \__codedoc_print_macroname:nN #1#2
  {
    \strut
    \__codedoc_get_hyper_target:eN
      {
        \exp_not:n {#1}
        \bool_if:NT #2 { \tl_to_str:n {TF} }
      }
      \l__codedoc_tmpa_tl
    \cs_if_exist:cTF { r@ \l__codedoc_tmpa_tl }
      { \exp_last_unbraced:NNo \hyperref [ \l__codedoc_tmpa_tl ] }
      { \use:n }
      {
        \int_compare:nTF { \str_count:n {#1} <= 28 }
          { \MacroFont } { \MacroLongFont }
        \tl_set:Nn \l__codedoc_tmpa_tl {#1}
        \tl_replace_all:NnV \l__codedoc_tmpa_tl
          { ~ } \c_catcode_other_space_tl
        \__codedoc_macroname_prefix:o \l__codedoc_tmpa_tl
        \__codedoc_macroname_suffix:N #2
      }
  }
\cs_new_protected:Npn \__codedoc_macroname_prefix:n #1
  {
    \__codedoc_if_macro_internal:nTF {#1}
      { \__codedoc_typeset_aux:n {#1} } {#1}
  }
\cs_generate_variant:Nn \__codedoc_macroname_prefix:n { o }
\cs_new_protected:Npn \__codedoc_macroname_suffix:N #1
  { \bool_if:NTF #1 { \__codedoc_typeset_TF: } { } }
```

(*__codedoc_print_macroname:nN* 定义结束。)

\MacroLongFont

```
\providecommand \MacroLongFont
  {
    \fontfamily{lmtt}\fontseries{lc}\small
  }
```

(*\MacroLongFont* 定义结束。这个函数被记录在第??页。)

__codedoc_print_testfile:n 用于显示某个宏是否有测试。

__codedoc_print_testfile_aux:n

```
\cs_new_protected:Npn \__codedoc_print_testfile:n #1
  {
    \bool_set_true:N \l__codedoc_macro_tested_bool
    \tl_if_eq:nnF {#1} {*}
      {
        \seq_if_in:NnF \g__codedoc_testfiles_seq {#1}
          {
            \seq_gput_right:Nn \g__codedoc_testfiles_seq {#1}
            \par
            \__codedoc_print_testfile_aux:n {#1}
          }
      }
  }
\cs_new_protected:Npn \__codedoc_print_testfile_aux:n #1
  {
    \footnotesize
    (
    \textit
      {
        The~ test~ suite~ for~ this~ command,~
        and~ others~ in~ this~ file,~ is~ \textsf{#1}
      }.
    )\par
  }
```

(*__codedoc_print_testfile:n* 和 *__codedoc_print_testfile_aux:n* 定义结束。)

\TestFiles

```
\DeclareDocumentCommand \TestFiles {m}
  {
    \par
    \textit
```

```
      {
        The~ following~ test~ files~ are~
        used~ for~ this~ code:~ \textsf{#1}.
      }
    \par \ignorespaces
  }
```

(*\TestFiles* 定义结束。这个函数被记录在第??页。)

\UnitTested

```
\DeclareDocumentCommand \UnitTested { } { \testfile* }
```

(*\UnitTested* 定义结束。这个函数被记录在第??页。)

\TestMissing

```
\DeclareDocumentCommand \TestMissing { m }
  { \__codedoc_test_missing:n {#1} }
```

(*\TestMissing* 定义结束。这个函数被记录在第??页。)

__codedoc_test_missing:n

\g__codedoc_missing_tests_prop 中的键是以一个 macro 环境的参数给出的一组宏的列表。值是文件名和关于缺少测试的注释的对。

```
\cs_new_protected:Npn \__codedoc_test_missing:n #1
  {
    \__codedoc_test_missing_aux:Nen
      \g__codedoc_missing_tests_prop
      { \seq_use:Nn \l__codedoc_names_seq { , } }
      { { \g_file_curr_name_str \c_space_tl (#1) } }
  }
\cs_new_protected:Npn \__codedoc_test_missing_aux:Nnn #1#2#3
  {
    \prop_get:NnNTF #1 {#2} \l__codedoc_tmpa_tl
      { \tl_put_right:Nn \l__codedoc_tmpa_tl { , #3 } }
      { \tl_set:Nn \l__codedoc_tmpa_tl {#3} }
    \prop_put:Nno #1 {#2} \l__codedoc_tmpa_tl
  }
\cs_generate_variant:Nn \__codedoc_test_missing_aux:Nnn { Ne }
```

(*__codedoc_test_missing:n* 定义结束。)

__codedoc_macro_end:

现在对于任何人来说都太晚了，来为这个宏声明一个测试文件，所以我们现在可以检查这个宏是否经过了测试。如果正在结束的 macro 环境是最外层的环境，则用

\texttt（如果有的话，附加 TF）包裹每个宏，并显示两个信息：这结束了一些宏的定义，并且它们被记录在某一页上。

```
\cs_new_protected:Npn \__codedoc_macro_end:
  {
    \endtrivlist
    \__codedoc_macro_end_check_tested:
    \int_compare:nNnT \l__codedoc_nested_macro_int = 1
      { \__codedoc_macro_end_style:n { \__codedoc_print_end_definition: } }
  }
```

（*__codedoc_macro_end:* 定义结束。）

__codedoc_macro_end_check_tested:

如果发出了 checktest 选项，并且该宏不是辅助的也不是变量（并且它没有测试），则将其添加到未经测试的宏序列中。

```
\cs_new_protected:Npn \__codedoc_macro_end_check_tested:
  {
    \bool_lazy_all:nT
     {
       { \g__codedoc_checktest_bool }
       { ! \l__codedoc_macro_var_bool }
       { ! \l__codedoc_macro_tested_bool }
     }
     {
       \seq_set_filter:NNn \l__codedoc_tmpa_seq \l__codedoc_names_seq
         { ! \__codedoc_if_macro_internal_p:n {##1} }
       \seq_gput_right:Ne \g__codedoc_not_tested_seq
         {
           \seq_use:Nn \l__codedoc_tmpa_seq { , }
           \bool_if:NTF \l__codedoc_macro_pTF_bool {~(pTF)}
             { \bool_if:NT \l__codedoc_macro_TF_bool {~(TF)} }
         }
     }
  }
```

（*__codedoc_macro_end_check_tested:* 定义结束。）

__codedoc_macro_end_style:n

顶层 macro 环境结束时额外信息的样式。

```
\cs_new_protected:Npn \__codedoc_macro_end_style:n #1
  {
    \nobreak \noindent
    { \footnotesize ( \emph{#1} ) \par }
  }
```

(__codedoc_macro_end_style:n 定义结束。)

__codedoc_print_end_definition:
__codedoc_macro_end_wrap_item:n
__codedoc_print_documented:

将每个项目用 \texttt 包围起来，同时将 _ 替换为 _。然后通过 \seq_use:Nnnn 列出宏名称，除非太多了。最后，如果宏既不是辅助的也不是内部的，则添加指向其文档位置的链接。

```
\cs_new_protected:Npn \__codedoc_macro_end_wrap_item:n #1
  {
    \tl_set:Nn \l__codedoc_tmpa_tl {#1}
    \tl_replace_all:NVn \l__codedoc_tmpa_tl
      \c_underscore_str { \_ }
    \texttt { \l__codedoc_tmpa_tl }
  }
\cs_new_protected:Npn \__codedoc_print_end_definition:
  {
    \seq_set_map:NNn \l__codedoc_tmpa_seq
      \g__codedoc_nested_names_seq
      { \__codedoc_macro_end_wrap_item:n {##1} }
    End~ of~ definition~ for~
    \int_compare:nTF { \seq_count:N \l__codedoc_tmpa_seq <= 3 }
      {
        \seq_use:Nnnn \l__codedoc_tmpa_seq
          { \,~and~ } { \,,~ } { \,,~and~ }
      }
      { \seq_item:Nn \l__codedoc_tmpa_seq {1}\,~and~others }
    \@.
    \__codedoc_print_documented:
  }
\cs_new_protected:Npn \__codedoc_print_documented:
  {
    \seq_gset_filter:NNn \g__codedoc_nested_names_seq
      \g__codedoc_nested_names_seq
      {
        ! \bool_lazy_any_p:n
          {
            { \__codedoc_if_macro_internal_p:n {##1} }
            { \l__codedoc_macro_deprecated_bool }
            { \l__codedoc_macro_nodoc_bool }
          }
      }
    \seq_if_empty:NF \g__codedoc_nested_names_seq
      {
        \int_set:Nn \l__codedoc_tmpa_int
```

```
          { \seq_count:N \g__codedoc_nested_names_seq }
        \int_compare:nNnTF \l__codedoc_tmpa_int = 1 {~This~} {~These~}
        \bool_if:NTF \l__codedoc_macro_var_bool {variable} {function}
        \int_compare:nNnTF \l__codedoc_tmpa_int = 1 {~is~} {s~are~}
        documented~on~page~
        \__codedoc_get_hyper_target:eN
          { \seq_item:Nn \g__codedoc_nested_names_seq { 1 } }
          \l__codedoc_tmpa_tl
        \exp_args:Ne \pageref { \l__codedoc_tmpa_tl } .
      }
    \seq_gclear:N \g__codedoc_nested_names_seq
  }
```

(*__codedoc_print_end_definition:*, *__codedoc_macro_end_wrap_item:n*, 和 *__codedoc_print_documented:* 定义结束。)

5.9.4 杂项

\DescribeOption 用于描述包选项的功能：为保持一致性而保留，但针对 doc v3 进行了更新。

```
\NewDocElement[idxtype = option, idxgroup = options]{Option}{optionenv}
```

(*\DescribeOption* 定义结束。这个函数被记录在第??页。)

这里有一些额外标记的定义，有助于构建您的文档结构。

danger (*env.*)
ddanger (*env.*)

```
\begin{[d]danger}
dangerous code
\end{[d]danger}
```

提供了来自 TEXbook 中所知的危险符号。

来自 manfnt 字体的实际字符：

```
\font \manual = manfnt \scan_stop:
\cs_gset:Npn \dbend { {\manual\char127} }
```

定义单个危险符号。每当包中有可能使用的特性比较棘手时使用它。FIXME：在与宏定义组合使用时必须修复。

```
\newenvironment {danger}
  {
    \begin{trivlist}\item[]\noindent
    \begingroup\hangindent=2pc\hangafter=-2
    \cs_set:Npn \par{\endgraf\endgroup}
    \hbox to0pt{\hskip-\hangindent\dbend\hfill}\ignorespaces
  }
  {
```

```
    \par\end{trivlist}
  }
```

如果有一些东西在错误使用时可能会导致严重问题，请使用双重危险符号。最好普通用户不要了解此类情况。

```
\newenvironment {ddanger}
  {
    \begin{trivlist}\item[]\noindent
    \begingroup\hangindent=3.5pc\hangafter=-2
    \cs_set:Npn \par{\endgraf\endgroup}
    \hbox to0pt{\hskip-\hangindent\dbend\kern2pt\dbend\hfill}\ignorespaces
  }{
      \par\end{trivlist}
  }
```

5.9.5 NB 和 NOTE

这些宏旨在添加到源文件中的额外注释，不进行排版。

\NB `\NB{wspr}{this is what I think about this!}`

```
\bool_if:NTF \g__codedoc_show_notes_bool
  {
    \NewDocumentCommand\NB{mm}
      {
        (\emph{Note}\footnote{\ttfamily [#1]:~\detokenize{#2}})
      }
  }
  {
    \NewDocumentCommand\NB{mm}{}
  }
```

(*\NB* 定义结束。这个函数被记录在第8页。)

NOTE (*env.*)

```
\begin{NOTE}{wspr}
  this is what I #$%& think about this!
\end{NOTE}
```

```
\bool_if:NTF \g__codedoc_show_notes_bool
  {
    \NewDocumentEnvironment{NOTE}{m}
      {
```

```
        \par\noindent (\emph{Note}~[\texttt{#1}]:\par
        \verbatim
      }
      {
        \endverbatim
        \par\noindent \emph{Note~end})\par
      }
  }
  {
    \NewDocumentEnvironment{NOTE}{m}{\comment}{\endcomment}
  }
```

5.10 脚注支持

function 和 variable 环境是盒子，因此会失去脚注。以下实现了对其的支持。目前依赖于 hyperref 内部机制来获取正确的目标。

```
\providecommand\Hy@footnote@currentHref{}
\prop_new:N\g__codedoc_fnmark_prop
\cs_new_protected:Npn \__codedoc_fn_store:
 {
   \prop_gput:Nee\g__codedoc_fnmark_prop
     {fn\int_use:N\c@footnote}{{\Hy@footnote@currentHref}{\int_use:N\c@footnote}}
 }
\cs_new_protected:Npn \__codedoc_fn_restore:n  #1
 {
   \prop_get:NnN \g__codedoc_fnmark_prop {fn#1}\l__codedoc_tmpa_tl
   \tl_gset:Ne\Hy@footnote@currentHref
     {\exp_last_unbraced:NV\use_i:nn \l__codedoc_tmpa_tl }
   \setcounter{footnote}{\exp_last_unbraced:NV\use_ii:nn \l__codedoc_tmpa_tl}
 }

\cs_generate_variant:Nn \hook_gput_next_code:nn {ne}
\cs_new_protected:Npn \__codedoc_fn_footnote:nn #1 #2
  {
    \footnotemark
    \__codedoc_fn_store:
    \hook_gput_next_code:ne {env/#1/after}
      {\exp_not:N\__codedoc_fn_restore:n{\int_use:N\c@footnote}{\exp_not:n{\footnotetext{#2}

\AddToHook{env/function/begin}{\def\footnote{\__codedoc_fn_footnote:nn{function}}}
\AddToHook{env/variable/begin}{\def\footnote{\__codedoc_fn_footnote:nn{variable}}}
```

5.11 建立模板的文档

```
\newenvironment{TemplateInterfaceDescription}[1]
  {
    \subsection{The~object~type~`#1'}
    \begingroup
    \@beginparpenalty\@M
    \description
    \def\TemplateArgument##1##2{\item[Arg:~##1]##2\par}
    \def\TemplateSemantics
      {
        \enddescription\endgroup
        \subsubsection*{Semantics:}
      }
  }
  {
    \par\bigskip
  }
\newenvironment{TemplateDescription}[2]
  {
    \subsection{The~template~`#2'~(object~type~#1)}
    \subsubsection*{Attributes:}
    \begingroup
    \@beginparpenalty\@M
    \description
    \def\TemplateKey##1##2##3##4
      {
        \item[##1~(##2)]##3%
        \ifx\TemplateKey##4\TemplateKey\else
%         \hskip0pt plus3em\penalty-500\hskip 0pt plus 1filll Default:~##4%
          \hfill\penalty500\hbox{}\hfill Default:~##4%
          \nobreak\hskip-\parfillskip\hskip0pt\relax
        \fi
        \par
      }
    \def\TemplateSemantics
      {
        \enddescription\endgroup
        \subsubsection*{Semantics~\&~Comments:}
      }
  }
  { \par \bigskip }
```

```
\newenvironment{InstanceDescription}[4][xxxxxxxxxxxxxxx]
  {
    \subsubsection{The~instance~`#3'~(template~#2/#4)}
    \subsubsection*{Attribute~values:}
    \begingroup
    \@beginparpenalty\@M
    \def\InstanceKey##1##2{\>\textbf{##1}\>##2\\}
    \def\InstanceSemantics{\endtabbing\endgroup
      \vskip-30pt\vskip0pt
      \subsubsection*{Layout~description~\&~Comments:}}
    \tabbing
    xxxx\=#1\=\kill
  }
  { \par \bigskip }
```

5.12 继承文档

这里的代码来自于 doc，去除了注释并转换成了 expl3 语法。在各个地方增加了新功能。

\StopEventually
\MaybeStop
\Finale
\AlsoImplementation
\OnlyDescription
\g__codedoc_finale_tl

TODO: 彻底移除这四个命令，并记录最好使用 documentation 和 implementation 环境。

```
\DeclareDocumentCommand \OnlyDescription { }
  { \bool_gset_false:N \g__codedoc_typeset_implementation_bool }
\DeclareDocumentCommand \AlsoImplementation { }
  { \bool_gset_true:N \g__codedoc_typeset_implementation_bool }
\DeclareDocumentCommand \StopEventually { m }
  {
    \bool_if:NTF \g__codedoc_typeset_implementation_bool
      {
        \@bsphack
        \tl_gset:Nn \g__codedoc_finale_tl { #1 \check@checksum }
        \init@checksum
        \@esphack
      }
      { #1 \endinput }
  }
```

我们还需要支持 doc V3 的 \MaybeStop（可能不存在）。

```
\cs_if_exist:NT \MaybeStop
   { \RenewCommandCopy \MaybeStop \StopEventually }

\DeclareDocumentCommand \Finale { }
  { \tl_use:N \g__codedoc_finale_tl }
```

```
\tl_new:N \g__codedoc_finale_tl
```

(*\StopEventually* 以及其它的定义结束。这些函数被记录在第??页。)

__codedoc_input:n 对文件进行输入，并进行一些设置：在文件中的第一个 <@@=⟨*module*⟩> 行之前，模块名应为空。

```
\cs_new_protected:Npn \__codedoc_input:n #1
  {
    \tl_gclear:N \g__codedoc_module_name_tl
    \MakePercentIgnore
    \input{#1}
    \MakePercentComment
  }
```

(*__codedoc_input:n* 定义结束。)

\DocInput 从 doc 修改而来，可以接受逗号分隔的输入（文件名里有逗号吗?)。

```
\DeclareDocumentCommand \DocInput { m }
  {
    \clist_map_inline:nn {#1}
      {
        \clist_put_right:Nn \g_docinput_clist {##1}
        \__codedoc_input:n {##1}
      }
  }
```

(*\DocInput* 定义结束。这个函数被记录在第??页。)

\DocInputAgain 使用 \g_docinput_clist 重新输入到目前为止已经 \DocInput 进来的内容。可以多次使用。

```
\DeclareDocumentCommand \DocInputAgain { }
  { \clist_map_function:NN \g_docinput_clist \__codedoc_input:n }
```

(*\DocInputAgain* 定义结束。这个函数被记录在第??页。)

\DocInclude 几乎与 \include 完全相同，但是使用 \DocInput 处理 .dtx 文件，而不是处理 .tex 文件。

```
\NewDocumentCommand \DocInclude { m }
  {
    \relax\clearpage
    \docincludeaux
    \IfFileExists{#1.fdd}
      { \cs_set:Npn \currentfile{#1.fdd} }
```

```
      { \cs_set:Npn \currentfile{#1.dtx} }
    \int_compare:nNnTF \@auxout = \@partaux
      { \@latexerr{\string\include\space cannot~be~nested}\@eha }
      { \@docinclude {#1} }
  }

\cs_gset:Npn \@docinclude #1
  {
    \clearpage
    \immediate\write\@mainaux{\string\@input{#1.aux}}
    \@tempswatrue
    \if@partsw
      \@tempswafalse
      \cs_set:Npe \@tempb {#1}
      \clist_map_inline:Nn \@partlist
        {
          \if_meaning:w \@tempa \@tempb
            \@tempswatrue
          \fi:
        }
    \fi
    \if@tempswa
      \cs_set_eq:NN \@auxout                 \@partaux
      \immediate\openout\@partaux #1.aux
      \immediate\write\@partaux{\relax}
      \cs_set_eq:NN \@ltxdoc@PrintIndex      \PrintIndex
      \cs_set_eq:NN \PrintIndex              \relax
      \cs_set_eq:NN \@ltxdoc@PrintChanges    \PrintChanges
      \cs_set_eq:NN \PrintChanges            \relax
      \cs_set_eq:NN \@ltxdoc@theglossary     \theglossary
      \cs_set_eq:NN \@ltxdoc@endtheglossary  \endtheglossary
      \part{\currentfile}
      {
        \cs_set_eq:NN \ttfamily\relax
        \cs_gset:Npe \filekey
          { \filekey,~ \thepart = { \ttfamily \currentfile } }
      }
      \DocInput{\currentfile}
      \cs_set_eq:NN \PrintIndex              \@ltxdoc@PrintIndex
      \cs_set_eq:NN \PrintChanges            \@ltxdoc@PrintChanges
      \cs_set_eq:NN \theglossary             \@ltxdoc@theglossary
      \cs_set_eq:NN \endtheglossary          \@ltxdoc@endtheglossary
      \clearpage
```

```
      \@writeckpt{#1}
      \immediate \closeout \@partaux
    \else
      \@nameuse{cp@#1}
    \fi
    \cs_set_eq:NN \@auxout \@mainaux
  }
```

在这里，MMMMI（用于页面引用）和 MMMMV（用于代码行引用）被 makeindex 解释为大写罗马数字页码，应该足够大，以避免与大写罗马数字页码的其他用途发生冲突。\@wrindex 和 \codeline@wrindex 之间有两个细微差别，第一个必须延迟写入，因为页面号尚不知道，第二个必须关闭一组并完成一些空间调整。

我们还提供了适用于我们用途的版本，引用的方式是

```
\cs_gset_protected:Npn \@wrindex #1
  {
    \protected@write \@indexfile {}
      { \string \indexentry {#1} { MMMMI - \thepage } }
    \endgroup \@esphack
  }
\cs_gset_protected:Npn \codeline@wrindex #1
  {
    \immediate\write\@indexfile
      {
        \string\indexentry{#1}
          { MMMMV - \filesep \int_use:N \c@CodelineNo }
      }
  }
\tl_gclear:N \filesep
\cs_new_protected:Npn \__codedoc_index_page_hc:nn #1#2
  {
    \protected@write \@indexfile {}
      {
        \string \indexentry { #1 \encapchar hdpindex{#2} }
          { MMMMI - \thepage }
      }
  }
\cs_new_protected:Npn \__codedoc_index_codeline_hc:nn #1#2
  {
    \immediate\write\@indexfile
      {
        \string \indexentry { #1 \encapchar hdclindex{\the\c@HD@hypercount}{#2} }
          { MMMMV - \filesep \int_use:N \c@CodelineNo - MMMD - \the\c@HD@hypercount - M }
```

```
      }
  }
```

每行代码都已经有一个单独的 HD.xx 目标了。最好是每行代码都有一个 CL.\the\c@CodelineNo 目标，并且将 hdclindex{\the\c@HD@hypercount} 更改为更接近 hdpindex 的机制，但我们需要更好地理解不同类型的索引，并且对于索引 \{ 和 \} 有一些微妙之处。

(*\DocInclude* 定义结束。这个函数被记录在第??页。)

\docincludeaux

```
\cs_gset:Npn \docincludeaux
  {
    \tl_set:Nn \thepart { \alphalph { part } }
    \tl_set:Nn \filesep { \thepart - }
    \cs_set_eq:NN \filekey \use_none:n
    \tl_gput_right:Nn \index@prologue
      {
        \cs_gset:Npn \@oddfoot
          {
            \parbox { \textwidth }
              {
                \strut \footnotesize
                \raggedright { \bfseries File~Key: } ~ \filekey
              }
          }
        \cs_set_eq:NN \@evenfoot \@oddfoot
      }
    \cs_gset_eq:NN \docincludeaux \relax
    \cs_gset:Npn \@oddfoot
      {
        \cs_if_exist:cTF { ver @ \currentfile }
          { File~\thepart :~{\ttfamily\currentfile}~ }
          {
            \GetFileInfo{\currentfile}
            File~\thepart :~{\ttfamily\filename}~
            Date:~\ExplFileDate\ % space
            Version~\ExplFileVersion
          }
        \hfill \thepage
      }
    \cs_set_eq:NN \@evenfoot \@oddfoot
  }
```

(*\docincludeaux* 定义结束。这个函数被记录在第??页。)

5.12.1 macrocode 环境

\xmacro@code
__codedoc_xmacro_code:n
__codedoc_xmacro_code:w

以一种独特的方式钩入 macrocode 环境：\xmacro@code 负责获取（和标记化）环境的主体。重新定义它以将其获取的内容传递给 __codedoc_xmacro_code:n。这个新宏将所有的 @@ 替换为适当的模块名称。唯一的例外是 <@@=⟨*module*⟩> 行本身，其中 @@ 不应修改。事实上，我们搜索这样的行来自动设置模块名称。我们需要小心：在下面的代码中不应出现任何 <@@=，因为 l3doc 也是使用此代码排版的。每次找到 <@@=，替换它后面的代码中的 ⟨*module*⟩，更新 ⟨*module*⟩，并循环检查是否有进一步的 <@@= 出现。

```
\group_begin:
  \char_set_catcode_other:N \^^A
  \char_set_catcode_active:N \^^S
  \char_set_catcode_active:N \^^B
  \char_set_catcode_other:N \^^L
  \char_set_catcode_other:N \^^R
  \char_set_lccode:nn { `\^^A } { `\% }
  \char_set_lccode:nn { `\^^S } { `\  }
  \char_set_lccode:nn { `\^^B } { `\\ }
  \char_set_lccode:nn { `\^^L } { `\{ }
  \char_set_lccode:nn { `\^^R } { `\} }
  \tex_lowercase:D
    {
      \group_end:
      \cs_set_protected:Npn \xmacro@code
          #1 ^^A ^^S^^S^^S^^S ^^Bend ^^Lmacrocode^^R
        { \__codedoc_xmacro_code:n {#1} \end{macrocode} }
    }
\group_begin:
  \char_set_catcode_active:N \<
  \char_set_catcode_active:N \>
  \cs_new_protected:Npn \__codedoc_xmacro_code:n #1
    {
      \tl_clear:N \l__codedoc_tmpa_tl
      \tl_if_in:nnTF {#1} { < @ @ = }
        { \__codedoc_xmacro_code:w #1 < @ @ = \q_recursion_tail > \q_recursion_stop }
        {
          \tl_set:Nn \l__codedoc_tmpa_tl {#1}
          \__codedoc_detect_internals:N \l__codedoc_tmpa_tl
          \__codedoc_replace_at_at:N \l__codedoc_tmpa_tl
          \tl_use:N \l__codedoc_tmpa_tl
        }
```

```
    }
  \cs_new_protected:Npn \__codedoc_xmacro_code:w #1 < @ @ = #2 >
    {
      % Add code before <@@=...>
      \tl_set:Nn \l__codedoc_tmpb_tl {#1}
      \__codedoc_detect_internals:N \l__codedoc_tmpb_tl
      \__codedoc_replace_at_at:N \l__codedoc_tmpb_tl
      \tl_put_right:NV \l__codedoc_tmpa_tl \l__codedoc_tmpb_tl
      % Check for \q_recursion_tail
      \quark_if_recursion_tail_stop_do:nn {#2}
        { \tl_use:N \l__codedoc_tmpa_tl }
      % Change module name and add <@@=#2> to typeset output
      \tl_gset:Nn \g__codedoc_module_name_tl {#2}
      \tl_put_right:Nn \l__codedoc_tmpa_tl { < \text { \verbatim@font @ @ = #2 } > }
      % Loop
      \__codedoc_xmacro_code:w
    }
\group_end:
```

(*\xmacro@code*, *__codedoc_xmacro_code:n*, 和 *__codedoc_xmacro_code:w* 定义结束。这个函数被记录在第??页。
)

5.13 文档结束时

打印所有已定义和已记录的宏/函数。

```
\iow_new:N \g__codedoc_func_iow

\tl_new:N \l__codedoc_doc_def_tl
\tl_new:N \l__codedoc_doc_undef_tl
\tl_new:N \l__codedoc_undoc_def_tl
\tl_const:Nn \c__codedoc_iow_separator_tl { ---- }
\tl_const:Nn \c__codedoc_iow_midrule_tl  { -- }

\cs_new_protected:Npn \__codedoc_show_functions_defined:
  {
    \bool_lazy_and:nnT
      { \g__codedoc_typeset_implementation_bool } { \g__codedoc_checkfunc_bool }
      {
        \iow_term:e { \c__codedoc_iow_separator_tl \iow_newline: }
        \iow_open:Nn \g__codedoc_func_iow { \c_sys_jobname_str .cmds }

        \tl_clear:N \l__codedoc_doc_def_tl
        \tl_clear:N \l__codedoc_doc_undef_tl
```

```
      \tl_clear:N \l__codedoc_undoc_def_tl
      \seq_gremove_duplicates:N \g_doc_functions_seq
      \seq_gremove_duplicates:N \g_doc_macros_seq
      \seq_map_inline:Nn \g_doc_functions_seq
        {
          \seq_if_in:NnTF \g_doc_macros_seq {##1}
            {
              \tl_put_right:Ne \l__codedoc_doc_def_tl
                { \iow_newline: > ~ ##1 }
            }
            {
              \tl_put_right:Ne \l__codedoc_doc_undef_tl
                { \iow_newline: ! ~ ##1 }
            }
        }
      \seq_map_inline:Nn \g_doc_macros_seq
        {
          \seq_if_in:NnF \g_doc_functions_seq {##1}
            {
              \tl_put_right:Ne \l__codedoc_undoc_def_tl
                { \iow_newline: ? ~ ##1 }
            }
        }
      \__codedoc_functions_typeout:nN
        {
          Functions~both~documented~and~defined: \iow_newline:
          (In~order~of~being~documented)
        }
        \l__codedoc_doc_def_tl
      \__codedoc_functions_typeout:nN
        { Functions~documented~but~not~defined: }
        \l__codedoc_doc_undef_tl
      \__codedoc_functions_typeout:nN
        { Functions~defined~but~not~documented: }
        \l__codedoc_undoc_def_tl

      \iow_close:N \g__codedoc_func_iow
      \iow_term:e { \c__codedoc_iow_separator_tl }
    }
  }
\AtEndDocument { \__codedoc_show_functions_defined: }
```

TODO: 用 `\iow_term:e`.

```
\cs_new_protected:Npn \__codedoc_functions_typeout:nN #1#2
  {
    \tl_if_empty:NF #2
      {
        \iow_now:Ne \g__codedoc_func_iow
          {
            \c__codedoc_iow_midrule_tl \iow_newline:
            #1 \iow_newline:
            \c__codedoc_iow_midrule_tl
            #2
          }
        \tl_clear:N #2
      }
  }
\cs_new_protected:Npn \__codedoc_show_not_tested:
  {
    \bool_if:NT \g__codedoc_checktest_bool
      {
        \tl_clear:N \l__codedoc_tmpa_tl
        \prop_if_empty:NF \g__codedoc_missing_tests_prop
          {
            \cs_set:Npn \__codedoc_tmpa:w ##1##2
              {
                \iow_newline:
                \space\space\space\space \exp_not:n {##1}
                \clist_map_function:nN {##2} \__codedoc_tmpb:w
              }
            \cs_set:Npn \__codedoc_tmpb:w ##1
              {
                \iow_newline:
                \space\space\space\space\space\space * ~ ##1
              }
            \tl_put_right:Ne \l__codedoc_tmpa_tl
              {
                \iow_newline: \iow_newline:
                The~ following~ macro(s)~ have~ incomplete~ tests:
                \iow_newline:
                \prop_map_function:NN
                  \g__codedoc_missing_tests_prop \__codedoc_tmpa:w
              }
          }
        \seq_if_empty:NF \g__codedoc_not_tested_seq
```

```
          {
            \cs_set:Npn \__codedoc_tmpa:w ##1
              { \clist_map_function:nN {##1} \__codedoc_tmpb:w }
            \cs_set:Npn \__codedoc_tmpb:w ##1
              {
                \iow_newline:
                \space\space\space\space ##1
              }
            \tl_put_right:Ne \l__codedoc_tmpa_tl
              {
                \iow_newline:
                \iow_newline:
                The~ following~ macro(s)~ do~ not~ have~ any~ tests:
                \iow_newline:
                \seq_map_function:NN
                  \g__codedoc_not_tested_seq \__codedoc_tmpa:w
              }
          }
        \tl_if_empty:NF \l__codedoc_tmpa_tl
          {
            \int_set:Nn \l__codedoc_tmpa_int { \tex_interactionmode:D }
            \errorstopmode
            \ClassError { l3doc } { \l__codedoc_tmpa_tl } { }
            \int_set:Nn \tex_interactionmode:D { \l__codedoc_tmpa_int }
          }
      }
  }
\AtEndDocument { \__codedoc_show_not_tested: }
```

5.14 索引

5.14.1 必要的修补

以下内容对于设置 hyperref 的目标很有用，例如为了索引的目的。与 hypdoc 不同的是，我们不尝试保存 PDF 目标，因为这会在早期运行时产生太多 pdfTEX 警告。

```
\cs_new_protected:Npn \__codedoc_target:
  {
    \mode_leave_vertical:
    \group_begin:
      \HD@savedestfalse \HD@target
    \group_end:
  }
```

强制在每行代码上创建目标。

```
\cs_set_nopar:Npe \theCodelineNo
  {
    \group_begin:
      \exp_not:N \HD@savedestfalse
      \exp_not:o \theCodelineNo
    \group_end:
  }
```

在目录（以及由 \@starttoc 引入的其他类似列表）中，我们抑制索引。这是因为标题中出现的 \cmd、\cs 或 \tn 只会在第二次运行时排版，并且正确地对其进行索引需要比我们已经需要的运行次数更多。此外，在目录中索引某些命令的用法并不实用。

```
\bool_new:N \l__codedoc_allow_indexing_bool
\bool_set_true:N \l__codedoc_allow_indexing_bool
\use:e
  {
    \exp_not:n { \cs_set_nopar:Npn \@starttoc #1 }
      {
        \group_begin:
          \bool_set_false:N \l__codedoc_allow_indexing_bool
          \exp_not:o { \@starttoc {#1} }
        \group_end:
      }
  }
```

5.14.2 用户空间命令

修复索引（暂时）：

```
\g@addto@macro \theindex { \MakePrivateLetters }
\cs_gset:Npn \verbatimchar {&}

\setcounter { IndexColumns } { 2 }
```

设置索引使用 \part

```
\IndexPrologue
  {
    \part*{Index}
    \markboth{Index}{Index}
    \addcontentsline{toc}{part}{Index}
    The~italic~numbers~denote~the~pages~where~the~
    corresponding~entry~is~described,~
    numbers~underlined~point~to~the~definition,~
```

```
    all~others~indicate~the~places~where~it~is~used.
  }
```

\SpecialIndex 尝试影响出现在 macrocode 环境中的命令在索引中的处理方式。

```
\cs_gset_protected:Npn \SpecialIndex #1
  {
    \@bsphack
    \__codedoc_special_index:nn {#1} { }
    \@esphack
  }
```

(*\SpecialIndex* 定义结束。这个函数被记录在第??页。)

```
\msg_new:nnn { l3doc } { print-index-howto }
  {
    Generate~the~index~by~executing\\
    \iow_indent:n
      { makeindex~-s~gind.ist~-o~\c_sys_jobname_str.ind~\c_sys_jobname_str.idx }
  }
\tl_gput_right:Nn \PrintIndex
  { \AtEndDocument { \msg_info:nn { l3doc } { print-index-howto } } }
```

5.14.3 内部索引命令

\it@is@a 在 macrocode 环境中，一字符命令的索引是通过 \it@is@a ⟨*char*⟩ 来生成的。可以修改该命令。

```
\cs_gset_protected:Npn \it@is@a #1
  {
    \use:e
      {
        \__codedoc_special_index_module:nnnnN
          {#1}
          { \bslash #1 }
          { }
          { }
          \c_false_bool
      }
  }
```

(*\it@is@a* 定义结束。这个函数被记录在第??页。)

__codedoc_special_index:nn

```
\cs_new_protected:Npn \__codedoc_special_index:nn #1#2
```

```
{
  \__codedoc_key_get:n {#1}
  \quark_if_no_value:NF \l__codedoc_override_module_tl
    { \tl_set_eq:NN \l__codedoc_index_module_tl \l__codedoc_override_module_tl }
  \__codedoc_special_index_module:ooonN
    { \l__codedoc_index_key_tl }
    { \l__codedoc_index_macro_tl }
    { \l__codedoc_index_module_tl }
    {#2}
    \l__codedoc_index_internal_bool
}
\cs_generate_variant:Nn \__codedoc_special_index:nn { o }
```

(*__codedoc_special_index:nn* 定义结束。)

__codedoc_special_index_module:nnnnN
__codedoc_special_index_module:ooonN
__codedoc_special_index_aux:nnnnnn
__codedoc_special_index_set:Nn

远程基于 Heiko 的替换方法，可以与 hypdoc 友好地配合使用。我们使用 \verb 或 \verbatim@font 结构，这取决于 #2 中的令牌数是否等于其字符数：如果不等，则表明存在诸如 \meta{...} 这样的结构。

```
\tl_new:N \l__codedoc_index_escaped_macro_tl
\tl_new:N \l__codedoc_index_escaped_key_tl

\cs_new_protected:Npn \__codedoc_special_index_module:nnnnN #1#2#3#4#5
```

#1： key
#2： macro
#3： module
#4： 索引“类型”（main/usage/等等）
#5： 是否是内部命令的布尔值

```
{
  \use:e
    {
      \exp_not:n { \__codedoc_special_index_aux:nnnnnn {#1} {#2} }
        \tl_if_empty:nTF {#3}
          { { } { } { } }
          {
            \str_if_eq:nnTF {#3} { TeX }
              {
                { TeX~and~LaTeX2e }
                { \string\TeX{}~and~\string\LaTeXe{} }
              }
              {
                {#3}
```

```
                  { \string\pkg{#3} }
                }
              { \bool_if:NT #5 { ~internal } ~commands: }
            }
      }
          {#4}
  }

\cs_generate_variant:Nn \__codedoc_special_index_module:nnnnN { ooo }

\cs_new_protected:Npn \__codedoc_special_index_aux:nnnnnn #1#2#3#4#5#6
```

#1： key
#2： macro
#3： 索引子标题字符串
#4： 索引子标题文本
#5： 索引子标题后缀（附加到参数 3 和 4）
#6： 索引“类型”（main/usage/等等）

```
  {
    \tl_set:Nn \l__codedoc_index_escaped_key_tl {#1}
    \__codedoc_quote_special_char:N \l__codedoc_index_escaped_key_tl
    \__codedoc_special_index_set:Nn \l__codedoc_index_escaped_macro_tl {#2}
    \str_if_eq:onTF { \@currenvir } { macrocode }
      { \__codedoc_index_codeline_hc:nn }
      {
        \str_case:nnF {#6}
          {
            { main }  { \__codedoc_index_codeline_hc:nn }
            { usage } { \__codedoc_index_page_hc:nn }
          }
          { \__codedoc_target: \__codedoc_index_page_hc:nn }
      }
      {
        \tl_if_empty:nF { #3 #4 #5 }
          { #3 #5 \actualchar #4 #5 \levelchar }
        \l__codedoc_index_escaped_key_tl
        \actualchar
        {
          \token_to_str:N \verbatim@font \c_space_tl
          \l__codedoc_index_escaped_macro_tl
        }
      }
```

```
      {#6}
  }
```

注意，如果几个连续的代码行以某种方式合并成一个范围，则 `#3` 可能会包含 `MMMMI-` 或 `MMMMV-` 多次。顺便提一下，破折号在我们的某些源文件中是活跃的，比如 `interface3.tex` 或 `source2e.tex`。

\hdpindex
__codedoc_old_hdpindex:nn
\hdclindex
__codedoc_old_hdclindex:nnn
__codedoc_hdindex:nn
\c__codedoc_active_minus_tl
__codedoc_hdindex_aux:nn
__codedoc_hdindex_aux:w

```
\group_begin:
\char_set_active_eq:NN - \scan_stop:
\tl_const:Ne \c__codedoc_active_minus_tl { \char_generate:nn { `- } { 13 } }
\group_end:
\cs_new_eq:NN \__codedoc_old_hdpindex:nn \hdpindex
\cs_new_eq:NN \__codedoc_old_hdclindex:nnn \hdclindex
\cs_gset_protected:Npn \hdpindex #1
  { \__codedoc_hdindex:nn { \__codedoc_old_hdpindex:nn {#1} } }
\cs_gset_protected:Npn \hdclindex #1#2
  { \__codedoc_hdindex:nn { \__codedoc_old_hdclindex:nnn {#1} {#2} } }
\cs_new_protected:Npn \__codedoc_hdindex:nn #1#2
  {
    \tl_set:Nn \l__codedoc_tmpa_tl {#2}
    \tl_replace_all:Nen \l__codedoc_tmpa_tl
      { \exp_not:V \c__codedoc_active_minus_tl \exp_not:V \c__codedoc_active_minus_tl }
      { -- }
    \seq_set_split:NnV \l__codedoc_tmpa_seq { -- } \l__codedoc_tmpa_tl
    \seq_set_map:NNn \l__codedoc_tmpa_seq \l__codedoc_tmpa_seq
      { \__codedoc_hdindex_aux:nn {#1} {##1} }
    \seq_use:Nn \l__codedoc_tmpa_seq { -- }
  }
\cs_new_protected:Npn \__codedoc_hdindex_aux:nn #1#2
  {
    \tl_set:Nn \l__codedoc_tmpa_tl {#2}
    \tl_replace_all:Nnn \l__codedoc_tmpa_tl { MMMM } { \use_none:nn }
    \tl_if_in:NnT \l__codedoc_tmpa_tl { MMMD }
      {
        \tl_replace_all:Nen \l__codedoc_tmpa_tl
          { \exp_not:V \c__codedoc_active_minus_tl MMMD } { - MMMD }
        \tl_replace_all:Nnn \l__codedoc_tmpa_tl { - MMMD } { \__codedoc_hdindex_aux:w }
      }
    \use:e { \exp_not:n {#1} { \exp_not:V \l__codedoc_tmpa_tl } }
  }
\cs_new_protected:Npn \__codedoc_hdindex_aux:w #1 M { }

\cs_new_protected:Npn \__codedoc_special_index_set:Nn #1#2
```

```
{
  \__kernel_tl_set:Ne #1 { \tl_to_str:n {#2} }
  \__codedoc_if_almost_str:nTF {#2}
    {
      \tl_replace_all:Nen #1 { \tl_to_str:n { __ } }
        {
          \verbatimchar
          \token_to_str:N \_ \token_to_str:N \_
          \token_to_str:N \verb * \verbatimchar
        }
      \exp_args:Ne \tl_map_inline:nn
        { \tl_to_str:N \verbatimchar \token_to_str:N _ }
        {
          \tl_replace_all:Nnn #1 {##1}
            {
              \verbatimchar \c_backslash_str ##1
              \token_to_str:N \verb * \verbatimchar
            }
        }
      \__kernel_tl_set:Ne #1
        {
          \token_to_str:N \verb * \verbatimchar
          #1 \verbatimchar
        }
    }
    {
      \tl_set:Nn #1 {#2}
      \tl_replace_all:NVn #1
        \c_backslash_str
        { \token_to_str:N \bslash \c_space_tl }
    }
  \__codedoc_quote_special_char:N #1
}
```

(__codedoc_special_index_module:nnnnN 以及其它的定义结束。这些函数被记录在第??页。)

__codedoc_quote_special_char:N 引用一些特殊字符。

```
\cs_new_protected:Npn \__codedoc_quote_special_char:N #1
  {
    \tl_map_inline:nn { \quotechar \actualchar \encapchar \levelchar \bslash }
      {
        \tl_replace_all:Nen #1
          { \tl_to_str:N ##1 } { \quotechar \tl_to_str:N ##1 }
```

```
      }
  }
```

(*__codedoc_quote_special_char:N 定义结束。*)

5.14.4 查找排序键和模块

__codedoc_key_get:n

从 #1 设置 \l__codedoc_index_macro_tl、\l__codedoc_index_key_tl 和 \l__codedoc_index_module_tl。基本函数由 __codedoc_key_get_base:nN 存储在 \l__codedoc_index_macro_tl 中，如果包含标记或没有签名，则回退到 #1。

⟨*key*⟩ 的起始点是字符串 \l__codedoc_index_key_tl。如果第一个字符是反斜杠，则删除它。然后通过存在 : 或 _ 的方式识别 expl 函数和变量，通过存在 @ 的方式识别 TEX/LATEX 2ε 命令。对于 expl 名称，我们调用 __codedoc_key_func: 或 __codedoc_key_var:，负责删除一些字符并找到模块名称，而对于 TEX/LATEX 2ε 命令，模块名称是 TeX，其他命令的模块名称为空。

```
\cs_new_protected:Npe \__codedoc_key_get:n #1
  {
    \exp_not:N \__codedoc_key_get_base:nN {#1} \exp_not:N \l__codedoc_index_macro_tl
    \__kernel_tl_set:Ne \exp_not:N \l__codedoc_index_key_tl
      { \exp_not:N \tl_to_str:N \exp_not:N \l__codedoc_index_macro_tl }
    \tl_clear:N \exp_not:N \l__codedoc_index_module_tl
    \tl_if_in:NnTF \exp_not:N \l__codedoc_index_key_tl { \tl_to_str:n { __ } }
      { \bool_set_true:N \exp_not:N \l__codedoc_index_internal_bool }
      { \bool_set_false:N \exp_not:N \l__codedoc_index_internal_bool }
    \exp_not:N \tl_if_head_eq_charcode:VNT
      \exp_not:N \l__codedoc_index_key_tl \c_backslash_str
      { \exp_not:N \__codedoc_key_pop: }
    \tl_if_in:NnTF \exp_not:N \l__codedoc_index_key_tl { \token_to_str:N : }
      { \exp_not:N \__codedoc_key_func: }
      {
        \tl_if_in:NnTF \exp_not:N \l__codedoc_index_key_tl { \token_to_str:N _ }
          { \exp_not:N \__codedoc_key_var: }
          {
            \tl_if_in:NnT \exp_not:N \l__codedoc_index_key_tl { \token_to_str:N @ }
              { \tl_set:Nn \exp_not:N \l__codedoc_index_module_tl { TeX } }
          }
      }
  }
\cs_new_protected:Npn \__codedoc_key_pop:
  {
    \__kernel_tl_set:Ne \l__codedoc_index_key_tl
```

```
      { \tl_tail:N \l__codedoc_index_key_tl }
  }
```

(__codedoc_key_get:n 定义结束。)

__codedoc_key_trim_module:n
__codedoc_key_drop_underscores:

辅助函数，从 \l__codedoc_index_module_tl 中删除第一次出现 #1 之后的所有内容。辅助函数，删除 \l__codedoc_index_key_tl 的任何前导下划线。

```
\cs_new_protected:Npn \__codedoc_key_trim_module:n #1
  {
    \cs_set:Npn \__codedoc_tmpa:w ##1 #1 ##2 \q_stop
      { \exp_not:n {##1} }
    \__kernel_tl_set:Ne \l__codedoc_index_module_tl
      { \exp_after:wN \__codedoc_tmpa:w \l__codedoc_index_module_tl #1 \q_stop }
  }
\cs_new_protected:Npn \__codedoc_key_drop_underscores:
  {
    \tl_if_head_eq_charcode:VNT \l__codedoc_index_key_tl _
      { \__codedoc_key_pop: \__codedoc_key_drop_underscores: }
  }
```

(__codedoc_key_trim_module:n 和 __codedoc_key_drop_underscores: 定义结束。)

__codedoc_key_func:

函数 __codedoc_key_func: 用于处理带有冒号的情况，通常用于 expl3 函数或者来自 l3keys 的键。在处理键时，会移除前导的句点（对于后一种情况），以及任何前导下划线，模块名称即是在冒号或下划线之前的部分。

```
\cs_new_protected:Npn \__codedoc_key_func:
  {
    \tl_if_head_eq_charcode:VNT \l__codedoc_index_key_tl .
      { \__codedoc_key_pop: }
    \__codedoc_key_drop_underscores:
    \tl_set_eq:NN \l__codedoc_index_module_tl \l__codedoc_index_key_tl
    \exp_args:No \__codedoc_key_trim_module:n { \token_to_str:N : }
    \exp_args:No \__codedoc_key_trim_module:n { \token_to_str:N _ }
  }
```

(__codedoc_key_func: 定义结束。)

__codedoc_key_var:
__codedoc_key_get_module:

函数 __codedoc_key_var: 处理没有 : 但有 _ 的情况，通常是变量，偶尔是带有下划线的非 expl3 函数（比如 Lua 函数）。首先检测第二个字符：如果是 _，则认为是正确的变量；否则将下划线之前的部分作为模块名称。对于变量，区分 quarks 和 scan marks（以 q 和 s 开头），然后去掉第一个字母（局部/全局/常量标记）和下划线，以改善索引排序。然后获取模块作为第一个（以下划线分隔的）“单词”。以前

我们根据有多少这样的“单词”来区分，以便检测诸如 \c_zero 这样的命令，它应该按照 int 变量排序，以及 \l_tmpa_dim，它应该在 dim 而不是 tmpa 模块中排序。现在第一种情况已经被弃用了一段时间，而 tmpa 和类似的情况是通过下面显式列表特殊处理的。工作原理是，如果模块在一个非有效模块名称的列表中，那么我们尝试最后一个词，如果也失败了（比如在弃用的 \c_one_hundred 中），我们就完全清空模块。

```
\cs_new_protected:Npn \__codedoc_key_var:
  {
    \exp_args:Ne \tl_if_head_eq_charcode:nNTF
      { \exp_args:No \str_tail:n \l__codedoc_index_key_tl } _
      {
        \str_case:en { \str_head:N \l__codedoc_index_key_tl }
          {
            { q } { \tl_set:Nn \l__codedoc_index_module_tl { quark } }
            { s } { \tl_set:Nn \l__codedoc_index_module_tl { scan } }
          }
        \__codedoc_key_pop:
        \__codedoc_key_pop:
        \__codedoc_key_drop_underscores:
        \tl_if_empty:NT \l__codedoc_index_module_tl
          {
            \seq_set_split:NoV \l__codedoc_tmpa_seq
              { \token_to_str:N _ } \l__codedoc_index_key_tl
            \seq_get_left:NN \l__codedoc_tmpa_seq \l__codedoc_index_module_tl
            \clist_if_in:NoT \g__codedoc_non_modules_clist \l__codedoc_index_module_tl
              {
                \seq_get_right:NN \l__codedoc_tmpa_seq \l__codedoc_index_module_tl
                \clist_if_in:NoT \g__codedoc_non_modules_clist \l__codedoc_index_module_tl
                  {
                    \tl_clear:N \l__codedoc_index_module_tl
                  }
              }
          }
      }
      {
        \tl_set_eq:NN \l__codedoc_index_module_tl \l__codedoc_index_key_tl
        \exp_args:No \__codedoc_key_trim_module:n { \token_to_str:N _ }
      }
  }
```

(*__codedoc_key_var:* 和 *__codedoc_key_get_module:* 定义结束。)

\g__codedoc_non_modules_clist 包含在 expl3 命令中出现的第一个单词列表，但它们不是真正的模块，因此在索引中应该按不同方式排序。

```
\clist_new:N \g__codedoc_non_modules_clist
\clist_gset:Ne \g__codedoc_non_modules_clist
  {
    \tl_to_str:n
      {

        alignment, ampersand, atsign, backslash, catcode, circumflex,
        code, colon, document, dollar, e, empty, false, hash, inf,
        initex, job, left, log, math, mark, max, minus, nan, nil, no,
        novalue, other, parameter, percent, pi, recursion, right, space,
        stop, term, tilde, tmpa, tmpb, true, underscore, zero, one, two,
        three, four, five, six, seven, eight, nine, ten, eleven, twelve,
        thirteen, fourteen, fifteen, sixteen, thirty, hundred

      }
  }
```

(*\g__codedoc_non_modules_clist 定义结束。*)

5.15 历次更新

将变更历史设置为使用 \part。允许在此处将控制命令连字符化...

```
\GlossaryPrologue
  {
    \part*{Change~History}
    {\GlossaryParms\ttfamily\hyphenchar\font=`\-}
    \markboth{Change~History}{Change~History}
    \addcontentsline{toc}{part}{Change~History}
  }

\msg_new:nnn { l3doc } { print-changes-howto }
  {
    Generate~the~change~list~by~executing\\
    \iow_indent:n
      { makeindex~-s~gglo.ist~-o~\c_sys_jobname_str.gls~\c_sys_jobname_str.glo }
  }
\tl_gput_right:Nn \PrintChanges
  { \AtEndDocument { \msg_info:nn { l3doc } { print-changes-howto } } }
```

5.16 默认配置

```
\bool_if:NTF \g__codedoc_typeset_implementation_bool
  {
    \RecordChanges
    \CodelineIndex
    \EnableCrossrefs
    \AlsoImplementation
  }
  {
    \CodelineNumbered
    \DisableCrossrefs
    \OnlyDescription
  }
```

⟨/class⟩

5.17 LaTeX3 源文件的内部宏

这些定义仅供 LaTeX3 文档使用；对于 l3doc 的第三方用户来说并非必需。以后这将被拆分成一个专门在各种 expl3 模块中加载的单独包。

⟨*cfg⟩

负责人。

```
\tl_const:Nn \Team
  {
    The~\LaTeX3~Project\thanks
      {\url{https://www.latex-project.org/latex3/}}
  }
\NewDocumentCommand{\ExplMakeTitle}{mm}
  {
    \title
      {
       The~\pkg{#1}~package \\ #2
      }
    \author
      {
       The~\LaTeX3~Project\thanks{E-mail:~
       \href{mailto:latex-l@listserv.uni-heidelberg.de}
            {latex-l@listserv.uni-heidelberg.de}}
      }
    \date{Released~\ExplFileDate}
    \maketitle
  }
```

5.18 数学扩展

用于 l3fp。

```
\AtBeginDocument
  {
    \clist_map_inline:nn
      {
       asin, acos, atan, acot,
       asinh, acosh, atanh, acoth, round, floor, ceil
      }
      { \exp_args:Nc \DeclareMathOperator{#1}{#1} }
  }
```

\nan

```
\NewDocumentCommand { \nan } { } { \text { \texttt { nan } } }
```

(*\nan* 定义结束。这个函数被记录在第??页。)

⟨/cfg⟩

5.19 Makeindex 配置

⟨*docist⟩

用 `l3doc.ist` 样式文件代替通常的 `gind.ist`，以确保在序列 `I J K` 中使用 `I` 而不是默认的 makeindex 行为 `I II III`。

Will: 我们需要这个吗?

Frank: 目前我们不分发或生成此文件。我们使用 `gind.ist`。

```
actual '='
quote '!'
level '>'
preamble
"\n \\begin{theindex} \n \\makeatletter\\scan@allowedfalse\n"
postamble
"\n\n \\end{theindex}\n"
item_x1   "\\efill \n \\subitem "
item_x2   "\\efill \n \\subsubitem "
delim_0   "\\pfill "
delim_1   "\\pfill "
delim_2   "\\pfill "
% The next lines will produce some warnings when
```

```
% running Makeindex as they try to cover two different
% versions of the program:
lethead_prefix    "{\\bfseries\\hfil "
lethead_suffix    "\\hfil}\\nopagebreak\n"
lethead_flag        1
heading_prefix    "{\\bfseries\\hfil "
heading_suffix    "\\hfil}\\nopagebreak\n"
headings_flag       1

% and just for source3:
% Remove R so I is treated in sequence I J K not I II III
page_precedence "rnaA"
```

(定义结束。)

⟨/docist⟩

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。

D

E

F

G

H

I

K

L

M

N

O

P

T

U

V