LaTeX 的钩子管理*

Frank Mittelbach† 【著】
张泓知 【译】

2024 年 1 月 8 日

目 录

*该模块版本号 v1.1f 日期为 2023/10/02,

†Phelype Oleinik 做了代码改进以使速度更快以及其它好处。

1 介绍

钩子（Hooks）是命令或环境代码中的处理点，在这些点上可以添加处理代码到现有命令中。不同的包可以对同一命令进行处理，为了确保安全处理，需要将不同包添加的代码块按合适的顺序进行排序。

包通过 `\AddToHook` 添加代码块，并使用默认的包名作为标签对其进行标记。

在 `\begin{document}` 处，所有钩子的代码根据一些规则（由 `\DeclareHookRule` 给出）进行排序，以实现快速执行，避免额外的处理开销。如果后续修改了钩子代码（或更改了规则），将生成新的用于快速处理的版本。

一些钩子已在文档的导言部分使用。如果在此时已经使用了钩子，钩子将被准备（并排序）以便执行。

2 包作者接口

钩子管理系统提供了一组 CamelCase 命令，用于传统的 LaTeX 2_ε 包（以及必要时在文档导言部分使用），同时也提供了用于现代包的 `expl3` 命令，这些现代包使用了 LaTeX 的 L3 编程层。在幕后，访问的是一组单一的数据结构，使得来自两个世界的包可以共存并访问其他包中的钩子。

2.1 LaTeX 2_ε 接口

2.1.1 声明钩子

除了少数例外，钩子必须在使用前声明。这些例外包括命令和环境的通用钩子（在 `\begin` 和 `\end` 执行）以及加载文件时运行的钩子（参见第 3.1节）。

`\NewHook` `\NewHook {⟨hook⟩}`

创建一个新的 ⟨*hook*⟩。如果这个钩子在一个包内声明，建议其名称总是结构化的，形式为：⟨*package-name*⟩/⟨*hook-name*⟩。如果需要，您可以通过添加更多的 `/` 部分来进一步细分名称。如果钩子名称已经存在，将引发错误并且不会创建该钩子。

⟨*hook*⟩ 可以使用点语法指定为当前包的名称。请参见第 2.1.4节。

`\NewReversedHook` `\NewReversedHook {⟨hook⟩}`

类似于 `\NewHook` 声明一个新的 ⟨*hook*⟩。不同之处在于，该钩子的代码块默认按相反顺序排列（最后添加的先执行）。钩子的任何规则都将在默认排序之后应用。详细内容请参见第 2.3 和 2.4 节。

⟨*hook*⟩ 可以使用点语法指定为当前包的名称。请参见第 2.1.4 节。

\NewMirroredHookPair `\NewMirroredHookPair {⟨hook-1⟩} {⟨hook-2⟩}`

是 `\NewHook{⟨hook-1⟩}\NewReversedHook{⟨hook-2⟩}` 的简写。

⟨hook⟩ 可以使用点语法指定为当前包的名称。请参见第 2.1.4 节。

\NewHookWithArguments `\NewHookWithArguments {⟨hook⟩} {⟨number⟩}`

创建一个具有 ⟨number⟩ 个参数的新 ⟨hook⟩,在其他方面与 `\NewHook` 完全相同。第 2.7 节详细解释了带参数的钩子。

⟨hook⟩ 可以使用点语法指定为当前包的名称。请参见第 2.1.4 节。

\NewReversedHookWithArguments `\NewReversedHookWithArguments {⟨hook⟩} {⟨number⟩}`

类似于 `\NewReversedHook`，但创建的钩子的代码带有 ⟨number⟩ 个参数。第 2.7 节详细解释了带参数的钩子。

⟨hook⟩ 可以使用点语法指定为当前包的名称。请参见第 2.1.4 节。

\NewMirroredHookPairWithArguments `\NewMirroredHookPairWithArguments {⟨hook-1⟩} {⟨hook-2⟩} {⟨number⟩}`

是 `\NewHookWithArguments{⟨hook-1⟩}{⟨number⟩}`
`\NewReversedHookWithArguments{⟨hook-2⟩}{⟨number⟩}` 的简写。第 2.7 节详细解释了带参数的钩子。

⟨hook⟩ 可以使用点语法指定为当前包的名称。请参见第 2.1.4 节。

2.1.2 通用钩子的特殊声明

此处的声明通常不应该被使用。它们提供了对主要涉及通用命令钩子的特殊用例的支持。

\DisableGenericHook `\DisableGenericHook {⟨hook⟩}`

在此声明之后[1]，⟨hook⟩ 将不再可用：进一步尝试向其添加代码将导致错误，任何使用，例如 `\UseHook`，都将什么也不做。

这主要用于通用命令钩子（参见 `ltcmdhooks-doc`），因为根据命令的定义，这些通用钩子可能不可用。如果已知此情况，包开发人员可以提前禁用这些钩子。

⟨hook⟩ 可以使用点语法指定为当前包的名称。请参见第 2.1.4 节。

\ActivateGenericHook `\ActivateGenericHook {⟨hook⟩}`

此声明激活了包/类提供的通用钩子（例如，在使用 `\UseHook` 或 `\UseOneTimeHook` 代码中使用的钩子），而无需显式使用 `\NewHook` 进行声明）。此命令撤销了 `\DisableGenericHook` 的效果。如果钩子已经被激活，此命令将不做任何操作。

请参见第 2.6 节，了解何时使用此声明。

[1] 在 2020/06 版本中，此命令称为 `\DisableHook`，但该名称是误导性的，因为它不应用于禁用非通用钩子。

2.1.3 在代码中使用钩子

\UseHook `\UseHook {⟨hook⟩}`

执行存储在 ⟨*hook*⟩ 中的代码。

在 `\begin{document}` 之前，并未设置钩子的快速执行代码，因此在那里使用钩子时，需要显式地首先进行初始化。由于这涉及到赋值，在这些时刻使用钩子并非与在 `\begin{document}` 后完全相同。

无法使用点语法指定 ⟨*hook*⟩。其前面的 . 将被视为文字字符。

\UseHookWithArguments `\UseHookWithArguments {⟨hook⟩} {⟨number⟩} {⟨arg1⟩} … {⟨argn⟩}`

执行存储在 ⟨*hook*⟩ 中的代码，并将 {⟨arg_1⟩} 至 {⟨arg_n⟩} 参数传递给 ⟨*hook*⟩。否则，其行为与 `\UseHook` 完全相同。⟨*number*⟩ 应该是钩子声明的参数数量。如果钩子未声明，此命令将不执行任何操作，并将从输入中删除 ⟨*number*⟩ 个项目。第 2.7 节解释了带参数的钩子。

无法使用点语法指定 ⟨*hook*⟩。其前面的 . 将被视为文字字符。

\UseOneTimeHook `\UseOneTimeHook {⟨hook⟩}`

一些钩子仅在一个地方使用（并且只能在一个地方使用），例如，在 `\begin{document}` 或 `\end{document}` 中的钩子。从那时起，通过已定义的 `\⟨addto-cmd⟩` 命令（例如，`\AddToHook` 或 `\AtBeginDocument` 等）向钩子添加内容将不起作用（就像在钩子代码内部使用这样的命令一样）。因此，习惯上重新定义 `\⟨addto-cmd⟩` 以简单地处理其参数，即本质上使其行为类似于 `\@firstofone`。

`\UseOneTimeHook` 就是这样做的：它记录钩子已被消耗，任何进一步尝试向其添加内容都将导致立即执行要添加的代码。

多次使用 `\UseOneTimeHook` 对同一个 {⟨*hook*⟩} 意味着它只在第一次使用时执行。例如，如果它在可以被多次调用的命令中使用，则该钩子仅在该命令的*第一次*调用时执行；这允许其用作“初始化钩子”。

应避免混合使用 `\UseHook` 和 `\UseOneTimeHook` 用于同一个 {⟨*hook*⟩}，但如果这样做了，那么在第一次 `\UseOneTimeHook` 后，两者都不会再执行。

无法使用点语法指定 ⟨*hook*⟩。其前面的 . 将被视为文字字符。详见第 2.1.4 节。

\UseOneTimeHookWithArguments `\UseOneTimeHookWithArguments {⟨hook⟩} {⟨number⟩} {⟨arg$_1$⟩} … {⟨arg$_n$⟩}`

与 `\UseOneTimeHook` 完全相同，但将参数 {⟨arg$_1$⟩} 至 {⟨arg$_n$⟩} 传递给 ⟨hook⟩。⟨number⟩ 应该是钩子声明的参数数量。如果钩子未声明，此命令将不执行任何操作，并将从输入中删除 ⟨number⟩ 个项目。

应注意，一次性钩子使用后，将不再可能使用 `\AddToHookWithArguments` 或类似方法添加内容到该钩子。`\AddToHook` 仍然正常工作。第 2.7 节解释了带参数的钩子。

无法使用点语法指定 ⟨hook⟩。其前面的 . 将被视为文字字符。详见第 2.1.4 节。

\AddToHook `\AddToHook {⟨hook⟩}[⟨label⟩]{⟨code⟩}`

向标记为 ⟨label⟩ 的 ⟨hook⟩ 添加 ⟨code⟩。当不提供可选参数 ⟨label⟩ 时，将使用 ⟨默认标签⟩（参见第 2.1.4 节）。如果 `\AddToHook` 在包/类中使用，则 ⟨默认标签⟩ 为包/类名，否则为 `top-level`（`top-level` 标签处理方式不同：详见第 2.1.5 节）。

如果 ⟨label⟩ 下已存在代码，则新的 ⟨code⟩ 将附加到现有代码中（即使这是一个反向钩子）。如果要替换 ⟨label⟩ 下的现有代码，请先应用 `\RemoveFromHook`。

钩子不必存在即可向其添加代码。但是，如果未声明，则显然添加的 ⟨code⟩ 将永远不会执行。这使得钩子能够在不考虑包装载顺序的情况下工作，并使得包装可以从其他包装中向钩子添加内容，而无需担心它们实际上是否在当前文档中使用。详见第 2.1.7 节。

可以使用点语法指定 ⟨hook⟩ 和 ⟨label⟩。详见第 2.1.4 节。

\AddToHookWithArguments `\AddToHookWithArguments {⟨hook⟩}[⟨label⟩]{⟨code⟩}`

与 `\AddToHook` 完全相同，但 ⟨code⟩ 可以访问通过 `#1`、`#2`、…、`#n`（与钩子声明的参数数量相符）传递给钩子的参数。如果 ⟨code⟩ 中包含不希望被理解为钩子参数的参数符号（`#`），则应将这些符号加倍。例如，使用 `\AddToHook` 可以写成：

```
\AddToHook{myhook}{\def\foo#1{Hello, #1!}}
```

但是要使用 `\AddToHookWithArguments` 实现相同效果，应写成：

```
\AddToHookWithArguments{myhook}{\def\foo##1{Hello, ##1!}}
```

因为在后一种情况中，`#1` 指的是钩子 `myhook` 的第一个参数。第 2.7 节解释了带参数的钩子。

可以使用点语法指定 ⟨hook⟩ 和 ⟨label⟩。详见第 2.1.4 节。

\RemoveFromHook `\RemoveFromHook {⟨hook⟩}[⟨label⟩]`

从 ⟨*hook*⟩ 中删除由 ⟨*label*⟩ 标记的任何代码。当不提供可选参数 ⟨*label*⟩ 时，将使用 ⟨*default label*⟩ （参见第 2.1.4 节）。

如果在 ⟨*hook*⟩ 中不存在 ⟨*label*⟩ 下的代码，或者 ⟨*hook*⟩ 不存在，则在尝试 `\RemoveFromHook` 时发出警告，并忽略该命令。仅当您确切地了解钩子中有哪些标签时，才应使用 `\RemoveFromHook`。通常情况下，这将是当某个包将某些代码添加到钩子中时，然后同一个包稍后删除此代码时。如果您想阻止来自另一个包的代码执行，则应使用 `voids` 规则（参见第 2.1.6 节）。

如果可选的 ⟨*label*⟩ 参数是 `*`，则会删除所有代码块。这相当危险，因为它可能会删除其他包的代码（可能不为人所知）；因此，它不应在包中使用，而只应在文档导言中使用！

可以使用点符号语法指定 ⟨*hook*⟩ 和 ⟨*label*⟩，以表示当前包名称。参见第 2.1.4 节。

与 `\DeclareHookRule` 中两个标签之间的 `voids` 关系相比，这是一种破坏性的操作，因为标记的代码已从钩子数据结构中删除，而关系设置可以通过稍后提供不同的关系来撤消。

此声明在文档主体内的一个有用应用是当您想临时添加代码到钩子中，然后稍后再次删除它时，例如，

```
\AddToHook{env/quote/before}{\small}
\begin{quote}
  A quote set in a smaller typeface
\end{quote}
...
\RemoveFromHook{env/quote/before}
... now back to normal for further quotes
```

请注意，您无法通过以下方式取消设置：

```
\AddToHook{env/quote/before}{}
```

因为这只是“添加”了一个空的代码块到钩子中。添加 `\normalsize` 是可行的，但这意味着钩子中包含了 `\small\normalsize`，这意味着没有充分理由进行两次字体大小更改。

上述操作仅在想要以较小字体排版多个引用时才需要。如果钩子仅需要一次使用，那么 `\AddToHookNext` 更简单，因为它在使用一次后会重置自身。

\AddToHookNext `\AddToHookNext {⟨hook⟩}{⟨code⟩}`

向下一次 ⟨*hook*⟩ 调用中添加 ⟨*code*⟩。该代码在常规钩子代码执行完毕后执行，并且仅执行一次，即在使用后删除。

使用此声明是全局操作，即使声明在组内使用，并且钩子的下一次调用发生在该组结束之后，代码也不会丢失。如果在执行钩子之前多次使用声明，则所有代码将按照声明的顺序执行。[2]

如果此声明与一次性钩子一起使用，则仅当声明在钩子调用之前时才会使用代码。这是因为与 `\AddToHook` 相比，在钩子调用已经发生时，此声明中的代码不会立即执行——换句话说，此代码仅在下一次钩子调用时真正执行（对于一次性钩子，没有这样的“下一次调用”）。这给您一个选择：我的代码应该始终执行，还是仅在一次性钩子使用时执行（如果不可能则不执行）？对于这两种可能性，都存在使用情况。

可以使用相同钩子（或不同钩子）嵌套此声明，例如，

`\AddToHookNext{⟨hook⟩}{⟨code-1⟩\AddToHookNext{⟨hook⟩}{⟨code-2⟩}}`

将在下一次使用 ⟨*hook*⟩ 时执行 ⟨*code-1*⟩，并在那时将 ⟨*code-2*⟩ 放入 ⟨*hook*⟩ 中，以便在下次运行钩子时执行它。

钩子不一定存在才能向其添加代码。这使得钩子可以独立于包加载顺序工作。参见第 2.1.7 节。

可以使用点符号语法指定 ⟨*hook*⟩，以表示当前包名称。参见第 2.1.4 节。

\AddToHookNextWithArguments `\AddToHookNextWithArguments {⟨hook⟩}{⟨code⟩}`

功能与 `\AddToHookNext` 完全相同，但 ⟨*code*⟩ 可包含对 ⟨*hook*⟩ 参数的引用，正如上面对 `\AddToHookWithArguments` 的描述。第 2.7 节解释了带参数的钩子。

可以使用点符号语法指定 ⟨*hook*⟩，以表示当前包名称。参见第 2.1.4 节。

\ClearHookNext `\ClearHookNext{⟨hook⟩}`

通常，仅当您准确知道它将应用在何处以及为何需要一些额外代码时，才会使用 `\AddToHookNext`。然而，在某些情况下，需要取消这种声明，例如，使用 `\DiscardShipoutBox` 丢弃页面时（但甚至在这种情况下也不总是如此），在这种情况下可以使用 `\ClearHookNext`。

2.1.4 钩子名称和默认标签

在包或类中最好使用 `\AddToHook`，不指定 ⟨*label*⟩，因为这样可以自动使用包或类名称，如果需要规则，则会很有帮助，并避免了输入错误的 ⟨*label*⟩。

[2] 没有重新排序此类代码块的机制（或删除它们）。

仅在非常特定的情况下才需要使用显式的 ⟨*label*⟩，例如，如果要将多个代码块添加到单个钩子中，并希望将它们放置在钩子的不同部分（通过提供一些规则）。

另一个情况是当您开发具有多个子包的大型包时。在这种情况下，您可能希望在整个子包中使用相同的 ⟨*label*⟩，以避免在内部重新组织代码时标签发生变化。

除了 `\UseHook`、`\UseOneTimeHook` 和 `\IfHookEmptyTF`（及其 expl3 接口 `\hook_use:n`、`\hook_use_once:n` 和 `\hook_if_empty:nTF`）之外，所有 ⟨*hook*⟩ 和 ⟨*label*⟩ 参数的处理方式相同：首先，对参数周围的空格进行修剪，然后完全展开，直到只剩下字符记号。如果 ⟨*hook*⟩ 或 ⟨*label*⟩ 的完全展开包含一个不可展开的非字符记号，将引发低级 TEX 错误（即，使用 TEX 的 `\csname…\endcsname` 展开 ⟨*hook*⟩，因此 ⟨*hook*⟩ 和 ⟨*label*⟩ 参数中允许使用 Unicode 字符）。`\UseHook`、`\UseOneTimeHook` 和 `\IfHookEmptyTF` 的参数处理方式基本相同，只是不会修剪参数周围的空格，以获得更好的性能。

虽然不是强制要求，但强烈建议由包定义的钩子和用于向其他钩子添加代码的 ⟨*label*⟩，包含包名称，以便轻松识别代码块的来源并防止冲突。这应该是标准做法，因此此钩子管理代码提供了一个快捷方式，用于在 ⟨*hook*⟩ 名称和 ⟨*label*⟩ 中引用当前包。如果 ⟨*hook*⟩ 名称或 ⟨*label*⟩ 仅由一个单独的点（`.`）或以点开头，后跟斜杠（`./`），则该点表示 ⟨*default label*⟩（通常是当前包或类名称——参见 `\SetDefaultHookLabel`）。“`.`” 或 “`./`” 在 ⟨*hook*⟩ 或 ⟨*label*⟩ 的任何其他位置都会被按原样处理，不会被替换。

例如，在名为 `mypackage.sty` 的包中，默认标签是 `mypackage`，因此以下说明：

```
\NewHook   {./hook}
\AddToHook {./hook}[.]{code}     % Same as \AddToHook{./hook}{code}
\AddToHook {./hook}[./sub]{code}
\DeclareHookRule{begindocument}{.}{before}{babel}
\AddToHook {file/foo.tex/after}{code}
```

等价于：

```
\NewHook   {mypackage/hook}
\AddToHook {mypackage/hook}[mypackage]{code}
\AddToHook {mypackage/hook}[mypackage/sub]{code}
\DeclareHookRule{begindocument}{mypackage}{before}{babel}
\AddToHook {file/foo.tex/after}{code}                    % unchanged
```

⟨*default label*⟩ 在包加载时自动设置为当前包或类的名称。如果挂钩命令在包外使用，或者当前文件没有使用 `\usepackage` 或 `\documentclass` 加载，那么将使用 `top-level` 作为 ⟨*default label*⟩。这可能会有例外情况——参见 `\PushDefaultHookLabel`。

此语法适用于所有 ⟨*label*⟩ 参数和大多数 ⟨*hook*⟩ 参数，无论是在 LATEX 2_ε 接口中，还是在第 2.2 节描述的 LATEX3 接口中。

重要：
点语法在 \UseHook 和一些通常在代码中使用的其他命令中**不可用**！

注意，但要注意，当执行挂钩命令时，. 被 ⟨*default label*⟩ 替换，因此在包结束后某种程度上执行的操作，如果使用了点语法，将会有错误的 ⟨*default label*⟩。出于这个原因，这种语法在 \UseHook（和 \hook_use:n）中不可用，因为大多数情况下，挂钩在定义它的包文件之外使用。这种语法也不适用于挂钩条件语句 \IfHookEmptyTF（和 \hook_if_empty:nTF），因为这些条件语句在挂钩管理代码的一些性能关键部分中使用，并且通常用于引用其他包的挂钩，因此点语法并不太合适。

在某些情况下，例如在大型包中，可能希望将代码分离为逻辑部分，但仍然使用主包名称作为 ⟨*label*⟩，那么可以使用 \PushDefaultHookLabel{...} … \PopDefaultHookLabel 或 \SetDefaultHookLabel{...} 设置 ⟨*default label*⟩。

\PushDefaultHookLabel
\PopDefaultHookLabel

```
\PushDefaultHookLabel {⟨default label⟩}
  ⟨code⟩
\PopDefaultHookLabel
```

\PushDefaultHookLabel 设置当前 ⟨*default label*⟩ 以在 ⟨*label*⟩ 参数或替换前导的“.”时使用。\PopDefaultHookLabel 将 ⟨*default label*⟩ 恢复为其先前的值。

在包或类中，⟨*default label*⟩ 等于包或类名称，除非显式更改。在其他任何地方，⟨*default label*⟩ 是 top-level（参见第 2.1.5 节），除非显式更改。

\PushDefaultHookLabel 的效果持续到下一个 \PopDefaultHookLabel。\usepackage（以及 \RequirePackage 和 \documentclass）内部使用

```
\PushDefaultHookLabel{⟨package name⟩}
  ⟨package code⟩
\PopDefaultHookLabel
```

来设置包或类文件的 ⟨*default label*⟩。在 ⟨*package code*⟩ 中，也可以使用 \SetDefaultHookLabel 更改 ⟨*default label*⟩。\input 和其他从 LaTeX 核心中输入文件的命令不使用 \PushDefaultHookLabel，因此由这些命令加载的文件中的代码不会获得专用的 ⟨*label*⟩！（也就是说，⟨*default label*⟩ 是加载文件时的当前活动标签。）

提供自己类似包的接口的包（例如 TikZ 的 \usetikzlibrary）可以使用 \PushDefaultHookLabel 和 \PopDefaultHookLabel 设置专用标签，并在这些上下文中模拟类似 \usepackage 的挂钩行为。

top-level 标签处理方式不同，并保留给用户文档，因此不允许将 ⟨*default label*⟩ 更改为 top-level。

\SetDefaultHookLabel \SetDefaultHookLabel {⟨*default label*⟩}

\SetDefaultHookLabel 与 \PushDefaultHookLabel 类似，将当前 ⟨*default label*⟩ 设置为在 ⟨*label*⟩ 参数中使用,或替换前导的“.”时使用。其效果持续到标签再次更改或到下一个 \PopDefaultHookLabel。\PushDefaultHookLabel 和 \SetDefaultHookLabel 的区别在于后者不保存当前 ⟨*default label*⟩。

当一个大型包由几个较小的包组成，但所有这些包都应具有相同的 ⟨*label*⟩ 时，\SetDefaultHookLabel 可以在每个包文件的开头使用以设置正确的标签。

在主文档中不允许使用 \SetDefaultHookLabel,其中 ⟨*default label*⟩ 是 top-level，且没有 \PopDefaultHookLabel 来结束其效果。同样不允许将 ⟨*default label*⟩ 更改为 top-level。

2.1.5 top-level 标签

为从主文档中添加的代码分配的 top-level 标签与其他标签不同。添加到导言区挂钩（通常是 \AtBeginDocument）的代码几乎总是用于更改包定义的内容，因此应该放在挂钩的最末端。

因此，添加在 top-level 的代码始终在挂钩的末尾执行，无论它在何处声明。如果挂钩被反转（参见 \NewReversedHook），则 top-level 代码块将在最开始执行。

关于 top-level 的规则不起作用：如果用户想为代码块设置特定规则，应该为该代码块使用不同的标签，并为该标签提供规则。

top-level 标签专属于用户，因此试图从包中使用该标签添加代码将导致错误。

2.1.6 定义挂钩代码之间的关系

默认假设是由不同包添加到挂钩的代码是独立的，并且它们执行的顺序是不相关的。虽然在许多情况下这是正确的，但在其他情况下显然是错误的。

在引入挂钩管理系统之前，包必须采取复杂的预防措施来确定其他包是否也被加载（在前面或后面），并找到一些方法相应地更改其行为。此外，通常用户需要负责以正确的顺序加载包，以使添加到挂钩的代码以正确的顺序添加，有些情况即使更改加载顺序也无法解决冲突。

使用新的挂钩管理系统，现在可以定义（即关系）不同包添加的代码块之间的规则，并明确描述它们应该被处理的顺序。

\DeclareHookRule `\DeclareHookRule {⟨hook⟩}{⟨label1⟩}{⟨relation⟩}{⟨label2⟩}`

为给定的 ⟨*hook*⟩ 定义 ⟨*label1*⟩ 和 ⟨*label2*⟩ 之间的关系。如果 ⟨*hook*⟩ 是 `??`，则为使用这两个标签的所有挂钩定义了默认关系，即具有标记为 ⟨*label1*⟩ 和 ⟨*label2*⟩ 的代码块的挂钩。对于特定挂钩的规则优先于使用 `??` 作为 ⟨*hook*⟩ 的默认规则。

目前，支持的关系有以下几种：

`before` 或 `<` ⟨*label1*⟩ 的代码出现在 ⟨*label2*⟩ 的代码之前。

`after` 或 `>` ⟨*label1*⟩ 的代码出现在 ⟨*label2*⟩ 的代码之后。

`incompatible-warning` 只能出现 ⟨*label1*⟩ 或 ⟨*label2*⟩ 的代码（表示两个包或其部分不兼容）。如果两个标签同时出现在同一个挂钩中，会发出警告。

`incompatible-error` 类似于 `incompatible-warning`，但是不会发出警告，而是引发 LaTeX 错误，并在冲突解决前从该挂钩中删除两个标签的代码。

`voids` ⟨*label1*⟩ 的代码覆盖了 ⟨*label2*⟩ 的代码。更确切地说，在该挂钩中会删除 ⟨*label2*⟩ 的代码。例如，如果一个包在功能上是另一个包的超集，因此希望撤消某个挂钩中的代码并用自己的版本替换，则可以使用此选项。

`unrelated` ⟨*label1*⟩ 和 ⟨*label2*⟩ 的代码顺序无关紧要。此规则用于撤销之前指定的不正确规则。

对于给定挂钩的两个标签之间只能存在一个关系，即后续的 `\DeclareHookRule` 会覆盖任何先前的声明。

可以使用点语法指定 ⟨*hook*⟩ 和 ⟨*label*⟩，以表示当前包名称。请参阅第 2.1.4 节。

\ClearHookRule `\ClearHookRule{⟨hook⟩}{⟨label1⟩}{⟨label2⟩}`

这是一种简化的写法，表示给定的 ⟨*hook*⟩ 中 ⟨*label1*⟩ 和 ⟨*label2*⟩ 之间无关联。

\DeclareDefaultHookRule `\DeclareDefaultHookRule{⟨label1⟩}{⟨relation⟩}{⟨label2⟩}`

这为所有挂钩设置了 ⟨*label1*⟩ 和 ⟨*label2*⟩ 之间的关系，除非特定挂钩被另一个规则覆盖。适用于一个包与另一个包有特定关系的情况，例如，是 `incompatible` 或总是需要特殊顺序 `before` 或 `after`。（技术上，这只是使用 `\DeclareHookRule` 并将 `??` 作为挂钩名称的简写。）

声明默认规则仅在文档导言部分支持。[3]

可以使用点语法指定 ⟨*label*⟩，以表示当前包名称。请参阅第 2.1.4 节。

[3] 尝试这样做，例如通过使用 `??` 的 `\DeclareHookRule`，会产生不良的副作用，并且不受支持（尽管出于性能原因未显式捕获）。

2.1.7 查询挂钩

简单的数据类型，比如记号列表，有三种可能的状态：

- 存在但为空；
- 存在且非空；以及
- 不存在（此时不存在空的概念）。

挂钩稍微复杂一些：一个挂钩可以存在也可以不存在，独立于此，它可以是空的也可以是非空的。这意味着即使一个挂钩不存在，它也可能是非空的，而且它也可以被禁用。

这种看似奇怪的状态可能发生在这样的情况下，例如，包 A 定义了挂钩 `A/foo`，而包 B 向该挂钩添加了一些代码。然而，文档可能在加载包 A 之前加载了包 B，或者根本没有加载包 A。在这两种情况下，一些代码被添加到了挂钩 `A/foo` 中，但该挂钩尚未定义，因此该挂钩被认为是非空的，但实际上它并不存在。因此，查询挂钩的存在性并不意味着它的空值，反之亦然。

由于代码或规则可以添加到一个挂钩，即使它还不存在，所以查询其存在性没有实际用途（与其他变量不同，其他变量只有在已经声明的情况下才能更新）。因此，只有对空值的测试具有公共接口。

当没有代码添加到挂钩的永久代码池或其“next”记号列表时，挂钩被认为空。挂钩不需要被声明为具有代码池。当使用 `\NewHook` 或其变体声明挂钩时，该挂钩被认为存在。当向其添加代码时，通用挂钩如 `file` 和 `env` 会自动声明。

`\IfHookEmptyTF` ⋆ `\IfHookEmptyTF {`⟨*hook*⟩`} {`⟨*true code*⟩`} {`⟨*false code*⟩`}`

检测 ⟨*hook*⟩ 是否为空（即没有使用 `\AddToHook` 或 `\AddToHookNext` 添加代码，或者通过 `\RemoveFromHook` 将代码移除），根据结果分别执行 ⟨*true code*⟩ 或 ⟨*false code*⟩。

无法使用点语法指定 ⟨*hook*⟩。前导的 . 会被视为字面量。

2.1.8 显示挂钩代码

如果需要使用挂钩规则调整挂钩中的代码执行顺序，了解挂钩相关信息、当前顺序和现有规则将会很有帮助。

\ShowHook \ShowHook {⟨*hook*⟩}
\LogHook \LogHook {⟨*hook*⟩}

显示关于 ⟨*hook*⟩ 的信息，例如：

- 挂钩中添加的代码块（及其标签），
- 任何用于排序的设置规则，
- 计算出的代码块执行顺序，
- 仅在下一次调用时执行的任何代码。

\LogHook 将信息打印到 .log 文件中，而 \ShowHook 将其打印到终端/命令窗口，并在 \errorstopmode 下启动 TEX 的提示，等待用户操作。

可以使用点语法指定 ⟨*hook*⟩，以表示当前包名称。请参阅第 2.1.4 节。

假设有一个名为 example-hook 的钩子，其 \ShowHook{example-hook} 的输出如下：

```
1   -> The hook 'example-hook':
2   > Code chunks:
3   >     foo -> [code from package 'foo']
4   >     bar -> [from package 'bar']
5   >     baz -> [package 'baz' is here]
6   > Document-level (top-level) code (executed last):
7   >     -> [code from 'top-level']
8   > Extra code for next invocation:
9   >     -> [one-time code]
10  > Rules:
11  >     foo|baz with relation >
12  >     baz|bar with default relation <
13  > Execution order (after applying rules):
14  >     baz, foo, bar.
```

在上面的列表中，第 3 到第 5 行展示了添加到钩子的三个代码片段及其相应的标签，格式如下：

⟨*label*⟩ -> ⟨*code*⟩

第 7 行展示了用户在主文档中添加的代码片段（标记为 top-level），格式如下：(labeled top-level) in the format

```
Document-level (top-level) code (executed ⟨first|last⟩):
  -> ⟨top-level code⟩
```

这段代码将是钩子执行的第一个或最后一个代码（如果钩子是正常的，则为 `last`，如果是反向的，则为 `first`）。这个代码块不受规则影响，也不参与排序。

第 9 行展示了下一次钩子执行时的代码片段格式，如下：

```
-> ⟨next-code⟩
```

这段代码将在下一次 `\UseHook{example-hook}` 时使用并消失，与之前提到的代码片段相反，这些代码片段只能通过 `\RemoveFromHook{⟨label⟩}[example-hook]` 从钩子中移除。

第 11 和第 12 行展示了影响该钩子的声明规则的格式，如下：

```
⟨label-1⟩|⟨label-2⟩ with ⟨default?⟩ relation ⟨relation⟩
```

这意味着 ⟨*relation*⟩ 应用于 ⟨*label-1*⟩ 和 ⟨*label-2*⟩，按照 `\DeclareHookRule` 中的详细说明顺序执行。如果关系是 `default`，则意味着此规则适用于所有钩子中的 ⟨*label-1*⟩ 和 ⟨*label-2*⟩（除非被非默认关系覆盖）。

最后，第 14 行按顺序列出了排序后钩子中的标签；即，在使用钩子时它们将被执行的顺序。

2.1.9 调试钩子代码

`\DebugHooksOn` `\DebugHooksOn`
`\DebugHooksOff`

打开或关闭钩子代码的调试。这会显示对钩子数据结构的大部分更改。输出相当粗糙，不适合正常使用。

2.2 L3 层的编程 (expl3) 接口

这是关于与 `expl3` 写的包一起使用的 LaTeX3 编程接口的快速摘要。与 LaTeX 2_ε 接口不同，它们始终仅使用必需的参数，例如，您总是必须为代码片段指定 ⟨*label*⟩。因此，我们建议即使在 `expl3` 包中也使用前面讨论过的声明，但选择权在您手中。

`\hook_new:n` `\hook_new:n {⟨hook⟩}`
`\hook_new_reversed:n` `\hook_new_reversed:n {⟨hook⟩}`
`\hook_new_pair:nn` `\hook_new_pair:nn {⟨hook-1⟩} {⟨hook-2⟩}`

创建一个具有正常或反向代码顺序的新 ⟨*hook*⟩。`\hook_new_pair:nn` 创建了一对此类钩子，其中 `{⟨hook-2⟩}` 是一个反向钩子。如果钩子名称已经被使用，将引发错误并且不会创建该钩子。

可以使用点号语法来指定 ⟨*hook*⟩，表示当前包的名称。参见第 2.1.4 节。

\hook_new_with_args:nn `\hook_new_with_args:nn {⟨hook⟩} {⟨number⟩}`
\hook_new_reversed_with_args:nn `\hook_new_reversed_with_args:nn {⟨hook⟩} {⟨number⟩}`
\hook_new_pair_with_args:nnn `\hook_new_pair_with_args:nnn {⟨hook-1⟩} {⟨hook-2⟩} {⟨number⟩}`

创建一个具有正常或反向代码顺序的新 ⟨*hook*⟩，在使用时从输入流中获取 ⟨*number*⟩ 个参数。`\hook_new_pair_with_args:nn` 创建了一对此类钩子，其中 {⟨*hook-2*⟩} 是一个反向钩子。如果钩子名称已经被使用，将引发错误并且不会创建该钩子。

可以使用点号语法来指定 ⟨*hook*⟩，表示当前包的名称。参见第 2.1.4 节。

\hook_disable_generic:n `\hook_disable_generic:n {⟨hook⟩}`

将 {⟨*hook*⟩} 标记为已禁用。任何进一步尝试向其添加代码或声明都将导致错误，并且任何对 `\hook_use:n` 的调用都将不起作用。

此声明旨在用于通用钩子，如果它们接收到代码，则已知它们无法正常工作（参见 `ltcmdhooks-doc`）。

可以使用点号语法来指定 ⟨*hook*⟩，表示当前包的名称。参见第 2.1.4 节。

\hook_activate_generic:n `\hook_activate_generic:n {⟨hook⟩}`

这类似于 `\hook_new:n`，但如果钩子之前使用 `\hook_new:n` 声明过，则不会执行任何操作。此声明应仅在特殊情况下使用，例如，当来自另一个包的命令需要更改，而不清楚是否已经先前显式声明了通用的 `cmd` 钩子（用于该命令）时。

通常情况下，应该使用 `\hook_new:n` 而不是这个声明。

\hook_use:n `\hook_use:n {⟨hook⟩}`
\hook_use:nnw `\hook_use:nnw {⟨hook⟩} {⟨number⟩} {⟨arg1⟩} … {⟨argn⟩}`

执行 {⟨*hook*⟩} 代码，然后执行（如果设置了）下一次调用的代码，随后清空该下一次调用的代码。对于使用参数声明的钩子，应使用 `\hook_use:nnw`，并且后面应跟着与声明的参数数量相同的大括号组。⟨*number*⟩ 应该是钩子声明的参数数量。如果钩子未声明，则此命令不起作用，并且将从输入中移除 ⟨*number*⟩ 个项目。

⟨*hook*⟩ 不能 使用点号语法指定。开头的 . 将被视为字面量处理。

\hook_use_once:n `\hook_use_once:n {⟨hook⟩}`
\hook_use_once:nnw `\hook_use_once:nnw {⟨hook⟩} {⟨number⟩} {⟨arg1⟩} … {⟨argn⟩}`

改变 {⟨*hook*⟩} 的状态，从现在开始，任何添加到钩子代码的操作都会立即执行。然后执行已设置的任何 {⟨*hook*⟩} 代码。对于使用参数声明的钩子，应使用 `\hook_use_once:nnw`，并且后面应跟着与声明的参数数量相同的大括号组。⟨*number*⟩ 应该是钩子声明的参数数量。如果钩子未声明，则此命令不起作用，并且将从输入中移除 ⟨*number*⟩ 个项目。

⟨*hook*⟩ 不能 使用点号语法指定。开头的 . 将被视为字面量处理。

\hook_gput_code:nnn `\hook_gput_code:nnn {⟨hook⟩} {⟨label⟩} {⟨code⟩}`
\hook_gput_code_with_args:nnn `\hook_gput_code_with_args:nnn {⟨hook⟩} {⟨label⟩} {⟨code⟩}`

将一段 ⟨*code*⟩ 添加到标记为 ⟨*label*⟩ 的 ⟨*hook*⟩ 中。如果标签已经存在，则将 ⟨*code*⟩ 追加到已有的代码后面。

如果使用了 `\hook_gput_code_with_args:nnn`，那么 ⟨*code*⟩ 可以访问传递给 `\hook_use:nnw`（或 `\hook_use_once:nnw`）的参数，使用 `#1`、`#2`、…、`#n`（最多为钩子声明的参数数量）。在这种情况下，如果要将实际参数标记添加到代码中，应该使用两个相同的参数标记。

如果要向外部的 ⟨*hook*⟩（例如内核或其他包）添加代码，那么约定是使用包名称作为 ⟨*label*⟩，而不是某个内部模块名称或其他任意字符串。

可以使用点号语法来指定 ⟨*hook*⟩ 和 ⟨*label*⟩，表示当前包的名称。参见第 2.1.4 节。

\hook_gput_next_code:nn `\hook_gput_next_code:nn {⟨hook⟩} {⟨code⟩}`
\hook_gput_next_code_with_args:nn

添加一段 ⟨*code*⟩，仅在下一次 ⟨*hook*⟩ 调用中使用。使用后即消失。

如果使用了 `\hook_gput_next_code_with_args:nn`，那么 ⟨*code*⟩ 可以访问传递给 `\hook_use:nnw`（或 `\hook_use_once:nnw`）的参数，使用 `#1`、`#2`、…、`#n`（最多为钩子声明的参数数量）。在这种情况下，如果要将实际参数标记添加到代码中，应该使用两个相同的参数标记。

这比 `\hook_gput_code:nnn` 更简单，代码将按照声明的顺序简单地附加到钩子末尾，即，在所有标准代码执行完毕后。因此，如果需要撤销标准操作，必须将其作为 ⟨*code*⟩ 的一部分处理。

可以使用点号语法来指定 ⟨*hook*⟩，表示当前包的名称。参见第 2.1.4 节。

\hook_gclear_next_code:n `\hook_gclear_next_code:n {⟨hook⟩}`

撤销任何之前的 `\hook_gput_next_code:nn`。

\hook_gremove_code:nn `\hook_gremove_code:nn {⟨hook⟩} {⟨label⟩}`

移除标记为 ⟨*label*⟩ 的 ⟨*hook*⟩ 中的任何代码。

如果在 ⟨*hook*⟩ 中没有 ⟨*label*⟩ 下的代码，或者 ⟨*hook*⟩ 不存在，尝试使用 `\hook_gremove_code:nn` 时将发出警告，并且命令将被忽略。

如果第二个参数是 `*`，则会移除所有代码块。这相当危险，因为会删除其他包中的代码，可能会影响到你不清楚的代码，请在使用之前三思！

可以使用点号语法来指定 ⟨*hook*⟩ 和 ⟨*label*⟩，表示当前包的名称。参见第 2.1.4 节。

\hook_gset_rule:nnnn `\hook_gset_rule:nnnn {⟨hook⟩} {⟨label1⟩} {⟨relation⟩} {⟨label2⟩}`

在 ⟨*hook*⟩ 中使用 ⟨*label1*⟩ 和 ⟨*label2*⟩ 进行关联。查看 `\DeclareHookRule` 获取允许的 ⟨*relation*⟩。如果 ⟨*hook*⟩ 是 `??`，则指定默认规则。

可以使用点号语法来指定 ⟨*hook*⟩ 和 ⟨*label*⟩，表示当前包的名称。参见第 2.1.4 节。点号语法在两个 ⟨*label*⟩ 参数中都进行解析，但通常只在其中一个参数中使用才有意义。

\hook_if_empty_p:n ⋆ `\hook_if_empty:nTF {⟨hook⟩} {⟨true code⟩} {⟨false code⟩}`

\hook_if_empty:nTF ⋆

检测 ⟨*hook*⟩ 是否为空（即，未使用 `\AddToHook` 或 `\AddToHookNext` 添加代码），并根据结果分别执行 ⟨*true code*⟩ 或 ⟨*false code*⟩。

⟨*hook*⟩ 不能 使用点号语法指定。开头的 . 将被视为字面量处理。

\hook_show:n `\hook_show:n {⟨hook⟩}`

\hook_log:n `\hook_log:n {⟨hook⟩}`

显示关于 ⟨*hook*⟩ 的信息，例如

- 添加到其中的代码块（及其标签），
- 设定的任何用于排序的规则，
- 计算出的代码块执行顺序，
- 仅在下一次调用时执行的任何代码。

`\hook_log:n` 将信息打印到 `.log` 文件，而 `\hook_show:n` 将其打印到终端/命令窗口，并启动 TEX 的提示符（仅在 `\errorstopmode`）等待用户操作。

可以使用点号语法来指定 ⟨*hook*⟩，表示当前包的名称。参见第 2.1.4 节。

\hook_debug_on: `\hook_debug_on:`

\hook_debug_off:

打开或关闭钩子代码的调试。这会显示钩子数据的变化。

2.3 关于钩子代码执行顺序

如果在不设置特殊规则的情况下，⟨*hook*⟩ 下不同标签的代码块被视为独立的，这意味着你不能对执行顺序做出假设！

假设你有以下声明：

```
\NewHook{myhook}
\AddToHook{myhook}[packageA]{\typeout{A}}
\AddToHook{myhook}[packageB]{\typeout{B}}
\AddToHook{myhook}[packageC]{\typeout{C}}
```

使用 \UseHook 执行钩子将按顺序产生类型输出 A B C。换句话说，执行顺序计算为 packageA、packageB、packageC，可以使用 \ShowHook{myhook} 进行验证：

```
-> The hook 'myhook':
> Code chunks:
>     packageA -> \typeout {A}
>     packageB -> \typeout {B}
>     packageC -> \typeout {C}
> Document-level (top-level) code (executed last):
>     ---
> Extra code for next invocation:
>     ---
> Rules:
>     ---
> Execution order:
>     packageA, packageB, packageC.
```

原因在于代码块被内部保存在属性列表中，属性列表的初始顺序是添加键-值对的顺序。但是，这仅在除添加之外没有其他操作时才成立！

举个例子，假设你想替换 packageA 的代码块，比如说，

```
\RemoveFromHook{myhook}[packageA]
\AddToHook{myhook}[packageA]{\typeout{A alt}}
```

那么你的顺序变成了 packageB、packageC、packageA，因为标签从属性列表中移除，然后重新添加（放在末尾）。

虽然这可能不太令人惊讶，但如果添加了多余的规则，例如，如果指定了

```
\DeclareHookRule{myhook}{packageA}{before}{packageB}
```

而不是之前我们得到的那些行

```
-> The hook 'myhook':
> Code chunks:
>     packageA -> \typeout {A}
>     packageB -> \typeout {B}
>     packageC -> \typeout {C}
> Document-level (top-level) code (executed last):
>     ---
> Extra code for next invocation:
```

```
>     ---
> Rules:
>     packageB|packageA with relation >
> Execution order (after applying rules):
>     packageA, packageC, packageB.
```

当你看到代码块时，仍然是相同的顺序，但是在标签 packageB 和 packageC 的执行顺序已经交换了。原因是，根据规则，有两种满足条件的顺序，而排序算法恰好选择了与没有规则的情况不同的顺序（在没有规则的情况下，算法根本不会运行，因为没有需要解决的内容）。顺便说一下，如果我们改为指定多余的规则

```
\DeclareHookRule{myhook}{packageB}{before}{packageC}
```

执行顺序就不会改变了。

总结：除非存在部分或完全定义顺序的规则（你可以依赖它们被满足），否则无法依赖执行顺序。

2.4 使用“反转”钩子

也许您想知道为什么可以用 \NewReversedHook 声明一个“反转”钩子以及它到底是做什么的。

简而言之：一个没有任何规则的反转钩子的执行顺序与使用 \NewHook 声明的钩子顺序完全相反。

如果您有一对期望添加涉及分组的代码的钩子，比如在第一个钩子中开始一个环境，在第二个钩子中关闭该环境，这将非常有帮助。举个有些牵强的例子[4]，假设有一个包添加了以下内容：

```
\AddToHook{env/quote/before}[package-1]{\begin{itshape}}
\AddToHook{env/quote/after} [package-1]{\end{itshape}}
```

结果是，所有引用将呈现为斜体。现在再假设另一个 package-too 也使引用变为蓝色，因此添加了以下内容：

```
\usepackage{color}
\AddToHook{env/quote/before}[package-too]{\begin{color}{blue}}
\AddToHook{env/quote/after} [package-too]{\end{color}}
```

现在，如果 env/quote/after 钩子是一个普通的钩子，那么在两个钩子中我们将得到相同的执行顺序，即：

[4] 有更简单的方法实现相同的效果。

```
package-1, package-too
```

（或相反）结果将是：

```
\begin{itshape}\begin{color}{blue} ...
\end{itshape}\end{color}
```

并且会出现一个错误消息，指出 `\begin{color}` 被 `\end{itshape}` 结束了。如果将 `env/quote/after` 声明为反转钩子，执行顺序就会反转，因此所有环境都以正确的顺序关闭，`\ShowHook` 将给出以下输出：

```
-> The hook 'env/quote/after':
> Code chunks:
>     package-1 -> \end {itshape}
>     package-too -> \end {color}
> Document-level (top-level) code (executed first):
>     ---
> Extra code for next invocation:
>     ---
> Rules:
>     ---
> Execution order (after reversal):
>     package-too, package-1.
```

执行顺序的反转发生在应用任何规则之前，因此如果您更改顺序，则可能必须在两个钩子中都进行更改，而不仅仅是一个，但这取决于用例。

2.5 “普通”钩子与“一次性”钩子的区别

在执行钩子时，开发人员可以选择使用 `\UseHook` 或 `\UseOneTimeHook`（或它们的 expl3 等效命令 `\hook_use:n` 和 `\hook_use_once:n`）。这个选择影响了在钩子第一次执行后如何处理 `\AddToHook`。

对于普通钩子，通过 `\AddToHook` 添加代码意味着代码块被添加到钩子数据结构中，然后每次调用 `\UseHook` 时都会使用它。

对于一次性钩子，处理方式略有不同：在调用 `\UseOneTimeHook` 后，任何进一步尝试通过 `\AddToHook` 向钩子添加代码的操作都将立即执行 ⟨*code*⟩。

这有一些需要注意的后果：

- 如果在钩子执行后向普通钩子添加 ⟨*code*⟩，并且由于某种原因它再也不会执行，则新的 ⟨*code*⟩ 将永远不会被执行。

- 相比之下，如果这种情况发生在一次性钩子上，则 ⟨*code*⟩ 会立即执行。

具体来说，这意味着类似以下结构的构建：

```
\AddToHook{myhook}
          { ⟨code-1⟩ \AddToHook{myhook}{⟨code-2⟩} ⟨code-3⟩ }
```

对于一次性钩子来说是有效的[5]（三个代码块依次执行），但对于普通钩子来说则意义不大，因为对于普通钩子，第一次执行 `\UseHook{myhook}` 时将会：

- 执行 ⟨*code-1*⟩，
- 然后执行 `\AddToHook{myhook}{code-2}`，将代码块 ⟨*code-2*⟩ 添加到下一次调用时使用的钩子中，
- 最后执行 ⟨*code-3*⟩。

第二次调用 `\UseHook` 时，它将执行上述操作，并且额外执行 ⟨*code-2*⟩，因为此时它已被作为代码块添加到钩子中。因此，每次使用钩子时都会添加另一个副本的 ⟨*code-2*⟩，所以该代码块将被执行 $\langle \#\ of\ invocations \rangle - 1$ 次。

2.6 包提供的通用钩子

钩子管理系统还实现了一类称为“通用钩子”的钩子。通常，钩子在可以在代码中使用之前必须显式声明。这确保了不同的包不会为不相关的目的使用相同的钩子名称——这会导致绝对混乱。然而，有一些“标准”钩子，对于它们事先声明是不合理的，例如，每个命令（理论上）都有一个关联的 `before` 和 `after` 钩子。在这种情况下，即对于命令、环境或文件钩子，可以通过 `\AddToHook` 简单地向其中添加代码来使用它们。对于更专门的通用钩子，例如 babel 提供的那些，您需要使用下面解释的 `\ActivateGenericHook` 进行额外的启用。

LaTeX 提供的通用钩子包括 `cmd`、`env`、`file`、`include`、`package` 和 `class`，所有这些都可以直接使用：您只需使用 `\AddToHook` 来添加代码，但不必在您的代码中添加 `\UseHook` 或 `\UseOneTimeHook`，因为这已经为您完成了（或者在 `cmd` 钩子的情况下，在必要时会在 `\begin{document}` 处对命令代码进行修补）。

但是，如果您想在自己的代码中提供进一步的通用钩子，情况稍有不同。为此，您应该使用 `\UseHook` 或 `\UseOneTimeHook`，但是不要使用 `\NewHook` 声明钩子。如前所述，对未声明的钩子名称调用 `\UseHook` 不起任何作用。因此，作为额外的设置步骤，您需要显式激活您的通用钩子。请注意，以这种方式生成的通用钩子始终是普通钩子。

[5]这有时会用于 `\AtBeginDocument`，这就是为什么它被支持的原因。

对于真正的通用钩子，在钩子名称中包含可变部分的提前激活将是困难或不可能的，因为您通常不知道真实文档中可能出现的可变部分的类型。

例如，babel 提供了诸如 babel/⟨*language*⟩/afterextras 的钩子。然而，babel 中的语言支持通常是通过外部语言包完成的。因此，在核心 babel 代码中为所有语言执行激活并不可行。相反，需要由每个语言包执行（或者由希望使用特定钩子的用户执行）。

由于这些钩子没有使用 \NewHook 声明，因此它们的名称应谨慎选择，以确保它们（可能）是唯一的。最佳做法是包括包或命令名称，就像上面 babel 的示例中所做的那样。

通过这种方式定义的通用钩子始终是普通钩子（即，您不能以这种方式实现反转钩子）。这是一个故意的限制，因为它大大加快了处理速度。

2.7 带参数的钩子

有时需要向钩子传递上下文信息，并且由于某种原因，无法将此类信息存储在宏中。为了满足这个目的，可以声明带参数的钩子，以便程序员可以传递钩子中代码所需的数据。

带参数的钩子的工作原理基本上与常规钩子相同，大多数适用于常规钩子的命令也适用于带参数的钩子。不同之处在于钩子的声明（使用 \NewHookWithArguments 而不是 \NewHook），然后可以使用 \AddToHook 和 \AddToHookWithArguments 添加代码，以及钩子的使用（使用 \UseHookWithArguments 而不是 \UseHook）。

带参数的钩子必须像常规钩子一样在首次使用前（所有常规钩子一样）声明，使用 \NewHookWithArguments{⟨*hook*⟩}{⟨*number*⟩}。然后添加到该钩子的所有代码都可以使用 #1 访问第一个参数，#2 访问第二个参数，依此类推，直到声明的参数数量。但是，仍然可以添加带有对尚未声明的钩子参数的引用的代码（稍后我们将讨论这一点）。钩子本质上是宏，所以 TEX 的 9 个参数限制适用，并且如果尝试引用不存在的参数号码，则会引发低级 TEX 错误。

要使用带参数的钩子，只需写 \UseHookWithArguments{⟨*hook*⟩}{⟨*number*⟩}，然后接着是参数的大括号列表。例如，如果钩子 test 需要三个参数，写法如下：

```
\UseHookWithArguments{test}{3}{arg-1}{arg-2}{arg-3}
```

然后，在钩子的 ⟨*code*⟩ 中，所有的 #1 将被替换为 arg-1，#2 将被替换为 arg-2，以此类推。如果在使用时，程序员提供的参数多于钩子声明的参数，则超出的参数将被钩子简单地忽略。如果提供的参数过少，则行为是不可预测的[6]。如果钩子未被声明，⟨*number*⟩ 个参数将从输入流中移除。

[6]钩子 将采用声明的参数数量，发生了什么取决于被抓取的内容以及钩子代码对其参数的处理。

使用 `\AddToHookWithArguments` 可以像常规 `\AddToHook` 一样向带参数的钩子添加代码，以实现不同的结果。在这种情况下，向钩子添加代码的主要区别在于首先可以访问钩子的参数，当然还有参数标记（`#6`）的处理方式。

在带参数的钩子中使用 `\AddToHook` 将像对所有其他钩子一样工作。这允许包开发人员向本来没有参数的钩子添加参数，而无需担心兼容性问题。这意味着，例如：

```
\AddToHook{test}{\def\foo#1{Hello, #1!}}
```

无论钩子 `test` 是否带参数，都会定义相同的宏 `\foo`。

使用 `\AddToHookWithArguments` 允许向添加的 ⟨*code*⟩ 访问钩子的参数，如 `#1`、`#2` 等，直到钩子声明的参数数量。这意味着，如果想要在 ⟨*code*⟩ 中添加一个 `#6`，那个标记必须在输入中重复。上面的相同定义，使用 `\AddToHookWithArguments`，需要重写为：

```
\AddToHookWithArguments{test}{\def\foo##1{Hello, ##1!}}
```

将上述示例扩展为使用钩子参数，我们可以重写类似以下内容的代码（现在从声明到使用，以获得完整的画面）：

```
\NewHookWithArguments{test}{1}
\AddToHookWithArguments{test}{%
  \typeout{Defining foo with "#1"}
  \def\foo##1{Hello, ##1! Some text after: #1}%
}
\UseHook{test}{Howdy!}
\ShowCommand\foo
```

上述代码运行后会在终端打印：

```
Defining foo with "Howdy!"
> \foo=macro:
#1->Hello, #1! Some text after: Howdy!.
```

请注意，在对 `\AddToHookWithArguments` 的调用中，`##1` 变为了 `#1`，而 `#1` 被传递给钩子的参数。如果再次使用钩子并提供不同的参数，定义自然会发生变化。

在声明钩子和确定钩子数量固定之前，可以添加引用钩子参数的代码。但是，如果钩子中添加的某些代码引用的参数多于将为该钩子声明的参数数量，则在钩子声明时会出现低级 TEX 错误，指示“非法参数编号”，这将很难追踪，因为在这一点上 TEX 无法知道引起问题的代码来自何处。因此，包编写者明确记录每个钩子可以接受多少个参数（如果有的话）是非常重要的，以便使用这些包的用户知道可以引用多少个参数，同样重要的是，了解每个参数的含义。

2.8 私有的 LaTeX 核心钩子

有几个地方对于 LaTeX 正确运行而言绝对至关重要，需要按照精确定义的顺序执行代码。即使可以通过钩子管理实现这一点（通过添加各种规则来确保与包添加的其他代码的适当排序），但这会使每个文档变得不必要地缓慢，因为即使结果是预先确定的，也必须进行排序。此外，这会强迫包作者不必要地为钩子添加进一步的规则（或者破坏 LaTeX）。

出于这个原因，此类代码不使用钩子管理，而是直接在公共钩子之前或之后使用私有内核命令,命名约定如下:`\@kernel@before@`⟨*hook*⟩ 或 `\@kernel@after@`⟨*hook*⟩。例如，在 `\enddocument` 中你会找到:

```
\UseHook{enddocument}%
\@kernel@after@enddocument
```

这意味着首先执行用户/包可访问的 `enddocument` 钩子，然后执行内部核心钩子。正如它们的名称所示，这些内核命令不应由第三方包更改，请不要这样做，这样有利于稳定性，而是使用其旁边的公共钩子。[7]

2.9 遗留的 LaTeX 2_ε 接口

LaTeX 2_ε 提供了一小部分钩子以及用于向其添加代码的命令。它们在这里列出，并保留了向后兼容性。

使用新的钩子管理机制，LaTeX 添加了几个额外的钩子，未来还将添加更多。请参见下一节以了解已经可用的内容。

[7]与 TeX 中的所有内容一样，没有强制执行此规则，通过查看代码很容易发现内核如何向其添加内容。因此，这个部分的主要目的是说：“请不要这样做，这是不可配置的代码!”

\AtBeginDocument `\AtBeginDocument [⟨label⟩] {⟨code⟩}`

如果不使用可选参数 ⟨*label*⟩,它基本上与以前一样,即将 ⟨*code*⟩ 添加到 `begindocument` 钩子（在 `\begin{document}` 内执行）。但是，通过这种方式添加的所有代码都使用标签 `top-level` 进行标记（参见第 2.1.5 节），如果在包或类之外进行，或者使用包/类名称，如果在这样的文件内部调用（参见第 2.1.4 节）。

这样，使用 `\AddToHook` 或 `\AtBeginDocument` 使用不同的标签显式地按照需要排序代码块，例如，在另一个包的代码之前或之后运行某些代码。当使用可选参数时，该调用等效于运行 `\AddToHook {begindocument} [`⟨*label*⟩`] {`⟨*code*⟩`}`。

`\AtBeginDocument` 是 `begindocument` 钩子（参见第 3.2 节）的包装器，它是一个一次性钩子。因此，在 `begindocument` 钩子在 `\begin{document}` 处执行后，任何尝试使用 `\AtBeginDocument` 或 `\AddToHook` 向该钩子添加 ⟨*code*⟩ 的操作都将导致该 ⟨*code*⟩ 立即执行。有关一次性钩子的更多信息，请参见第 2.5 节。

对于具有已知顺序要求的重要包，我们可能会随着时间的推移向内核（或这些包）添加规则，以便它们不受文档加载顺序的影响而工作。

\AtEndDocument `\AtEndDocument [⟨label⟩] {⟨code⟩}`

Like `\AtBeginDocument` but for the `enddocument` hook.

在 LaTeX 2$_\varepsilon$ 中之前存在的少量钩子在内部使用诸如 `\@begindocumenthook` 之类的命令，有时包直接增强它们而不是通过 `\AtBeginDocument` 进行操作。出于这个原因，目前支持这样做，也就是说，如果系统检测到这样一个内部传统钩子命令包含代码，则将其添加到新的钩子系统中，并使用标签 `legacy` 进行标记，以防止丢失。

然而, 随着时间的推移, 剩余的直接使用情况需要更新, 因为在未来的某个 LaTeX 发布中，我们将关闭此传统支持，因为它会不必要地减慢处理速度。

3 LaTeX 2$_\varepsilon$ 命令和由钩子增强的环境

在本节中，我们描述了现在由 LaTeX 提供的标准钩子，或者提供了指向其他文档的指针，其中对它们进行了描述。本节将随时间而增长（并且可能最终会转移到 usrguide3）。

3.1 通用钩子

正如前面所述，除了通用钩子之外，所有钩子在使用之前都必须使用 `\NewHook` 声明。所有通用钩子的名称都采用以下形式：“⟨*type*⟩/⟨*name*⟩/⟨*position*⟩”, 其中 ⟨*type*⟩ 取自下面预定义的列表，⟨*name*⟩ 是其含义将取决于 ⟨*type*⟩ 的变量部分。最后一个组成部分 ⟨*position*⟩ 具有更复杂的可能性：它始终可以是 `before` 或 `after`；对于 `env`

钩子，还可以是 begin 或 end；对于 include 钩子，还可以是 end。每个特定的钩子在下面或 ltcmdhooks-doc.pdf 或 ltfilehook-doc.pdf 中有文档记录。

LaTeX 提供的通用钩子属于以下六种类型：

env 在环境之前和之后执行的钩子——⟨*name*⟩ 是环境的名称，⟨*position*⟩ 的可用值为 before、begin、end 和 after；

cmd 添加到命令之前和之后执行的钩子——⟨*name*⟩ 是命令的名称，⟨*position*⟩ 的可用值为 before 和 after；

file 在读取文件之前和之后执行的钩子——⟨*name*⟩ 是文件的名称（带有扩展名），⟨*position*⟩ 的可用值为 before 和 after；

package 在加载包之前和之后执行的钩子——⟨*name*⟩ 是包的名称，⟨*position*⟩ 的可用值为 before 和 after；

class 在加载类之前和之后执行的钩子——⟨*name*⟩ 是类的名称，⟨*position*⟩ 的可用值为 before 和 after；

include 在 \include 包含的文件之前和之后执行的钩子——⟨*name*⟩ 是包含的文件的名称（不包含 .tex 扩展名），⟨*position*⟩ 的可用值为 before、end 和 after。

下面详细介绍了上述每个钩子，并提供了链接的文档。

3.1.1 所有环境的通用钩子

每个环境 ⟨*env*⟩ 现在都有四个与之关联的钩子：

env/⟨*env*⟩/before 这个钩子作为 \begin 的一部分执行，特别是在开始环境组之前。因此，它的范围不受环境的限制。

env/⟨*env*⟩/begin 这个钩子作为 \begin 的一部分直接位于特定于环境开始的代码之前（例如，\newenvironment 的第二个参数）。它的范围是环境主体。

env/⟨*env*⟩/end 这个钩子作为 \end 的一部分直接位于特定于环境结束的代码之前（例如，\newenvironment 的第三个参数）。

env/⟨*env*⟩/after 这个钩子作为 \end 的一部分，在环境结束的代码和环境组结束之后执行。因此，它的范围不受环境的限制。

该钩子实现为一个反向钩子，因此，如果两个包向 env/⟨*env*⟩/before 和 env/⟨*env*⟩/after 添加代码，它们可以添加周围的环境，且关闭它们的顺序是正确的。

通用环境钩子即使对于只能在文档中出现一次的环境也不是一次性钩子。[8] 与其他钩子不同，也不需要使用 \NewHook 声明它们。

这些钩子只有在使用 \begin{⟨*env*⟩} 和 \end{⟨*env*⟩} 时才会执行。如果环境代码是通过对 \⟨*env*⟩ 和 \end⟨*env*⟩ 进行低级调用（例如，为了避免环境组），则它们不可用。如果要在使用此方法的代码中使用它们，您需要自己添加它们，即编写类似以下内容的代码：

```
\UseHook{env/quote/before}\quote
    ...
\endquote\UseHook{env/quote/after}
```

以添加外部钩子等。

为了与现有包的兼容性，还提供了以下四个命令来设置环境钩子；但对于新的包，我们建议直接使用钩子名称和 \AddToHook。

\BeforeBeginEnvironment `\BeforeBeginEnvironment [⟨*label*⟩] {⟨*env*⟩} {⟨*code*⟩}`

此声明使用 ⟨*label*⟩ 将代码添加到 env/⟨*env*⟩/before 钩子中。如果未给出 ⟨*label*⟩，则使用 ⟨*default label*⟩（参见第 2.1.4 节）。

\AtBeginEnvironment `\AtBeginEnvironment [⟨*label*⟩] {⟨*env*⟩} {⟨*code*⟩}`

这类似于 \BeforeBeginEnvironment，但它将代码添加到 env/⟨*env*⟩/begin 钩子中。

\AtEndEnvironment `\AtEndEnvironment [⟨*label*⟩] {⟨*env*⟩} {⟨*code*⟩}`

这类似于 \BeforeBeginEnvironment，但它将代码添加到 env/⟨*env*⟩/end 钩子中。

\AfterEndEnvironment `\AfterEndEnvironment [⟨*label*⟩] {⟨*env*⟩} {⟨*code*⟩}`

这类似于 \BeforeBeginEnvironment，但它将代码添加到 env/⟨*env*⟩/after 钩子中。

3.1.2 命令的通用钩子

与环境类似，现在（至少在理论上）对于任何 LaTeX 命令都有两个通用钩子可用。它们是：

cmd/⟨*name*⟩/before 此钩子在命令执行的开头执行。

cmd/⟨*name*⟩/after 此钩子在命令体的最后执行。它实现为一个反向钩子。

实际上有一些限制，尤其是 after 钩子仅适用于一部分命令。有关这些限制的详细信息可以在 ltcmdhooks-doc.pdf 中找到，或者在 ltcmdhooks-code.pdf 中查看代码。

[8] 因此，如果在处理环境之后添加代码，只有在环境再次出现且不会发生代码执行时，该代码才会被执行。

3.1.3 文件加载操作提供的通用钩子

在通过其高级接口加载文件(例如 \input、\include、\usepackage、\RequirePackage 等）时，LaTeX 添加了几个钩子。这些钩子在 ltfilehook-doc.pdf 中有文档说明，或者可以在 ltfilehook-code.pdf 中查看代码。

3.2 \begin{document} 提供的钩子

直到 2020 年，\begin{document} 仅提供了一个可通过 \AtBeginDocument 添加的钩子。多年的经验表明，在一个地方使用这个单一的钩子是不够的，因此，在添加通用钩子管理系统的过程中，在此处添加了许多其他的钩子。这些钩子的位置被选择为提供与外部包（例如 etoolbox 和其他增强 \document 以获得更好控制的包）所提供的支持相同。

现在支持以下钩子（它们都是一次性钩子）：

begindocument/before 此钩子在 \document 开始时执行，可以将其视为位于导言区末尾的代码的钩子，这就是 etoolbox 的 \AtEndPreamble 使用它的方式。

这是一个一次性钩子，因此在执行后，所有进一步尝试添加代码的操作都将立即执行该代码（参见第 2.5 节）。

begindocument 这个钩子是通过使用 \AddToHook{begindocument} 或使用 \AtBeginDocument 添加的，它在读取 .aux 文件和大多数初始化完成后执行，因此可以被钩子代码修改和检查。它后面紧跟一些不应该被更改的进一步初始化，因此稍后会出现。

该钩子不应该用于添加排版素材，因为我们仍然处于 LaTeX 的初始化阶段，而不是文档主体。如果需要将此类素材添加到文档主体中，请改用下一个钩子。

这是一个一次性钩子，因此在执行后，所有进一步尝试添加代码的操作都将立即执行该代码（参见第 2.5 节）。

begindocument/end 此钩子在 \document 代码结束时执行，换句话说，在文档主体开始时执行。其后唯一的命令是 \ignorespaces。

这是一个一次性钩子，因此在执行后，所有进一步尝试添加代码的操作都将立即执行该代码（参见第 2.5 节）。

\begin 执行的通用钩子也存在，即 env/document/before 和 env/document/begin，但对于此特殊环境，最好使用上述专用的一次性钩子。

3.3 \end{document} 提供的钩子

LaTeX 2_ε 一直提供 \AtEndDocument 来添加代码到 \end{document}，就在通常执行的代码前面。尽管这对于 LaTeX 2.09 的情况是一个很大的改进，但对于许多用

例来说并不够灵活，因此，诸如 etoolbox、atveryend 等包对 \enddocument 进行了补丁，以添加额外的代码挂载点。

使用包进行补丁总是有问题的，因为会导致冲突（代码可用性、补丁的顺序、不兼容的补丁等）。因此，在 \enddocument 代码中添加了一些额外的钩子，允许包以受控的方式在各个地方添加代码，而无需覆盖或补丁核心代码。

现在支持以下钩子（它们都是一次性钩子）：

enddocument 与 \AtEndDocument 相关联的钩子。它在 \enddocument 开始时立即调用。

当执行此钩子时，可能仍有未处理的素材（例如推迟列表上的浮动体），而钩子可能会添加进一步要排版的素材。之后，调用 \clearpage 来确保所有这样的素材都被排版。如果没有等待的素材，则 \clearpage 没有效果。

这是一个一次性钩子，因此在执行后，所有进一步尝试添加代码的操作都将立即执行该代码（参见第 2.5 节）。

enddocument/afterlastpage 如名称所示，此钩子不应该接收生成更多页面素材的代码。这是做一些最终的收尾工作的正确位置，可能要向 .aux 文件写一些信息（在此时，该文件仍然打开以接收数据，但由于不会再有页面，您需要使用 \immediate\write 来写入它）。这也是设置任何在下一步重新读取 .aux 文件时运行的测试代码的正确位置。

执行此钩子后，.aux 文件将关闭写入，并重新读取以进行一些测试（例如查找缺失引用或重复标签等）。

这是一个一次性钩子，因此在执行后，所有进一步尝试添加代码的操作都将立即执行该代码（参见第 2.5 节）。

enddocument/afteraux 此时，.aux 文件已经被重新处理，因此这是进行最终检查和向用户显示信息的可能位置。但是，对于后者，您可能更喜欢下一个钩子，这样您的信息会显示在（可能较长的）文件列表之后，如果通过 \listfiles 请求的话。

这是一个一次性钩子，因此在执行后，所有进一步尝试添加代码的操作都将立即执行该代码（参见第 2.5 节）。

enddocument/info 此钩子用于接收向终端写入最终信息消息的代码。它紧随上一个钩子之后执行（因此两者可以合并，但是然后添加更多代码的包始终需要提供显式规则来指定它应该放在何处）。

此钩子已经包含内核添加的一些代码（标签重复警告、缺失引用、字体替换等），即在使用 \listfiles 时列出的文件列表和警告信息。

这是一个一次性钩子，因此在执行后，所有进一步尝试添加代码的操作都将立即执行该代码（参见第 2.5 节）。

enddocument/end 最后，此钩子在最终调用 \@@end 前执行。

这是一个一次性钩子，因此在执行后，所有进一步尝试添加代码的操作都将立即执行该代码（参见第 2.5 节）。甚至在此之后添加代码可能吗?

还有一个名为 shipout/lastpage 的钩子。此钩子作为文档中最后一个 \shipout 的一部分执行，以允许包将最终的 \special 添加到该页面。此钩子相对于上述列表中的钩子的执行时间可以因文档而异。此外，要正确确定哪个 \shipout 是最后一个，需要多次运行 LaTeX，因此最初它可能在错误的页面上执行。有关详细信息，请参阅第 3.4 节。

还可以使用通用的 env/document/end 钩子, 它是由 \end 执行的, 即在上述的第一个钩子前执行。但是请注意,另一个通用的 \end 环境钩子,即 env/document/after，永远不会被执行，因为此时 LaTeX 已经完成了文档处理。

3.4 \shipout 操作提供的钩子

在 LaTeX 生成页面的过程中添加了几个钩子和机制。这些内容在 ltshipout-doc.pdf 中有详细记录，或者在 ltshipout-code.pdf 中有相关代码。

3.5 段落提供的钩子

段落处理已经增加了一些内部和公共钩子。这些内容在 ltpara-doc.pdf 中有详细记录，或者在 ltpara-code.pdf 中有相关代码。

3.6 NFSS 命令提供的钩子

对于需要同时支持多个脚本（因此有几套字体，例如支持拉丁字体和日文字体）的语言，NFSS 字体命令如 \sffamily 需要同时切换拉丁字体为“Sans Serif”，并且额外修改第二套字体。

为了支持这一点，几个 NFSS 命令都有钩子来添加这种支持。

rmfamily 在 \rmfamily 执行了其初始检查并准备字体系列更新后，此钩子在 \selectfont 之前执行。

sffamily 这类似于 rmfamily 钩子，但用于 \sffamily 命令。

ttfamily 这类似于 rmfamily 钩子，但用于 \ttfamily 命令。

normalfont `\normalfont` 命令将字体编码、系列和形状重置为文档默认值。然后执行此钩子，最后调用 `\selectfont`。

expand@font@defaults 内部命令 `\expand@font@defaults` 展开并保存当前的元系列（rm/sf/tt）和元系列（bf/md）的默认值。如果为了中文或日文等增加了 NFSS 机制，则可能需要在此时设置进一步的默认值。这可以在此钩子中完成，在此宏的末尾执行。

bfseries/defaults, bfseries 如果用户显式更改了 `\bfdefault` 的值，则在调用 `\bfseries` 时将其新值用于设置元系列（rm/sf/tt）的 bf 系列默认值。在这种情况下，`bfseries/defaults` 钩子允许进一步进行调整。如果检测到这样的更改，则仅执行此钩子。相反，`bfseries` 钩子总是在调用 `\selectfont` 以更改新系列之前执行。

mdseries/defaults, mdseries 这两个钩子与上面的类似，但是在 `\mdseries` 命令中。

selectfont 此钩子在 `\selectfont` 内执行，用于评估当前的编码、系列、形状和大小，并选择新的字体（如果必要则加载）。在此钩子执行后，NFSS 仍会执行任何必要的更新以适应新的大小（例如更改 `\strut` 的大小）和更改编码。

此钩子用于在主要字体更改的同时，处理其他字体的情况（例如在处理多种不同字母表的 CJK 处理中）。

3.7 标记机制提供的钩子

详细内容请参阅 `ltmarks-doc.pdf`。

insertmark 此钩子允许在 `\InsertMark` 插入标记时进行特殊设置。它在分组中执行，因此局部更改仅适用于被插入的标记。

4 代码实现

```
1 ⟨@@=hook⟩
2 ⟨*2ekernel | latexrelease⟩
3 \ExplSyntaxOn
4 ⟨latexrelease⟩\NewModuleRelease{2020/10/01}{lthooks}
5 ⟨latexrelease⟩                    {The~hook~management~system}
```

4.1 调试

`\g__hook_debug_bool` 保持当前调试状态。

```
6 \bool_new:N \g__hook_debug_bool
```

(*\g__hook_debug_bool* 定义结束。)

\hook_debug_on:
\hook_debug_off:
__hook_debug:n
__hook_debug_gset:

利用重定义 __hook_debug:n 来打开或关闭调试。

```
\cs_new_eq:NN \__hook_debug:n \use_none:n
\cs_new_protected:Npn \hook_debug_on:
  {
    \bool_gset_true:N \g__hook_debug_bool
    \__hook_debug_gset:
  }
\cs_new_protected:Npn \hook_debug_off:
  {
    \bool_gset_false:N \g__hook_debug_bool
    \__hook_debug_gset:
  }
\cs_new_protected:Npn \__hook_debug_gset:
  {
    \cs_gset_protected:Npx \__hook_debug:n ##1
      { \bool_if:NT \g__hook_debug_bool {##1} }
  }
```

(*\hook_debug_on:* 以及其它的定义结束。这些函数被记录在第18页。)

4.2 从其他内核模块的内部借用

__hook_str_compare:nn

Private copy of __str_if_eq:nn

```
\cs_new_eq:NN \__hook_str_compare:nn \__str_if_eq:nn
```

(*__hook_str_compare:nn* 定义结束。)

4.3 声明

\l__hook_tmpa_bool

整个包中都使用的临时布尔值。

```
\bool_new:N \l__hook_tmpa_bool
```

(*\l__hook_tmpa_bool* 定义结束。)

\l__hook_return_tl
\l__hook_tmpa_tl
\l__hook_tmpb_tl

整个包中使用的临时变量。

```
\tl_new:N \l__hook_return_tl
\tl_new:N \l__hook_tmpa_tl
\tl_new:N \l__hook_tmpb_tl
```

(*\l__hook_return_tl*, *\l__hook_tmpa_tl*, 和 *\l__hook_tmpb_tl* 定义结束。)

\g__hook_all_seq 在一些地方，我们需要所有曾经定义的钩子名称列表，因此我们在这个序列中对它们进行跟踪。

```
\seq_new:N \g__hook_all_seq
```

(*\g__hook_all_seq 定义结束。*)

\l__hook_cur_hook_tl 存储当前正在排序的钩子的名称。

```
\tl_new:N \l__hook_cur_hook_tl
```

(*\l__hook_cur_hook_tl 定义结束。*)

\l__hook_work_prop 一个属性列表，保存正在排序的钩子的 \g__hook_⟨*hook*⟩_code_prop 的副本，以便对其进行操作，以免对钩子数据结构造成破坏性影响。

```
\prop_new:N \l__hook_work_prop
```

(*\l__hook_work_prop 定义结束。*)

\g__hook_used_prop 所有接收代码的钩子（用于调试显示）。

```
\prop_new:N \g__hook_used_prop
```

(*\g__hook_used_prop 定义结束。*)

\g__hook_hook_curr_name_tl 用于钩子命令的默认标签，并且用一个堆栈来跟踪包中的包。

\g__hook_name_stack_seq

```
\tl_new:N \g__hook_hook_curr_name_tl
\seq_new:N \g__hook_name_stack_seq
```

(*\g__hook_hook_curr_name_tl 和 \g__hook_name_stack_seq 定义结束。*)

__hook_tmp:w 通用的临时宏。

```
\cs_new_eq:NN \__hook_tmp:w ?
```

(*__hook_tmp:w 定义结束。*)

\c__hook_empty_tl 一个空的记号列表和一个包含九个参数的列表。

\c__hook_nine_parameters_tl

```
\tl_const:Nn \c__hook_empty_tl { }
\tl_const:Nn \c__hook_nine_parameters_tl { #1#2#3#4#5#6#7#8#9 }
```

(*\c__hook_empty_tl 和 \c__hook_nine_parameters_tl 定义结束。*)

\tl_gremove_once:Nx expl3 函数的一些变体。

\tl_show:x

\tl_log:x

\tl_set:Ne

\cs_replacement_spec:c

\prop_put:Nne

\str_count:e

FMi: 可能应该移到 expl3

```
\cs_generate_variant:Nn \tl_gremove_once:Nn { Nx }
\cs_generate_variant:Nn \tl_show:n { x }
\cs_generate_variant:Nn \tl_log:n { x }
\cs_generate_variant:Nn \tl_set:Nn { Ne }
\cs_generate_variant:Nn \cs_replacement_spec:N { c }
\cs_generate_variant:Nn \prop_put:Nnn { Nne }
\cs_generate_variant:Nn \str_count:n { e }
```

(*\tl_gremove_once:Nx 以及其它的定义结束。*)

\s__hook_mark 用于定界参数的扫描标记。

```
\scan_new:N \s__hook_mark
```

(*\s__hook_mark 定义结束。*)

__hook_use_none_delimit_by_s_mark:w
__hook_use_i_delimit_by_s_mark:nw

移除直到下一个\s__hook_mark的记号。

```
\cs_new:Npn \__hook_use_none_delimit_by_s_mark:w #1 \s__hook_mark { }
\cs_new:Npn \__hook_use_i_delimit_by_s_mark:nw #1 #2 \s__hook_mark {#1}
```

(*__hook_use_none_delimit_by_s_mark:w 和 __hook_use_i_delimit_by_s_mark:nw 定义结束。*)

__hook_tl_set:cn expl3 函数的私有副本。l3debug 只会给公共名称添加调试，而不会对这些副本进行调试，所以我们不必在每处都使用\debug_suspend: 和\debug_resume:。

像__hook_tl_set:Nn这样的函数必须被重新定义，而不是复制，因为在 expl3 中，它们使用__kernel_tl_(g)set:Nx，这也被 l3debug 所修补。

```
\cs_new_protected:Npn \__hook_tl_set:cn #1#2
  { \cs_set_nopar:cpx {#1} { \__kernel_exp_not:w {#2} } }
```

(*__hook_tl_set:cn 定义结束。*)

__hook_tl_gset:Nn
__hook_tl_gset:Nx
__hook_tl_gset:cn
__hook_tl_gset:co
__hook_tl_gset:cx

与上述相同。

```
\cs_new_protected:Npn \__hook_tl_gset:Nn #1#2
  { \cs_gset_nopar:Npx #1 { \__kernel_exp_not:w {#2} } }
\cs_new_protected:Npn \__hook_tl_gset:Nx #1#2
  { \cs_gset_nopar:Npx #1 {#2} }
\cs_generate_variant:Nn \__hook_tl_gset:Nn { c, co }
\cs_generate_variant:Nn \__hook_tl_gset:Nx { c }
```

(*__hook_tl_gset:Nn 定义结束。*)

__hook_tl_gput_right:Nn
__hook_tl_gput_right:Ne
__hook_tl_gput_right:cn

与上述相同。

```
\cs_new_protected:Npn \__hook_tl_gput_right:Nn #1#2
  { \__hook_tl_gset:Nx #1 { \__kernel_exp_not:w \exp_after:wN { #1 #2 } } }
\cs_generate_variant:Nn \__hook_tl_gput_right:Nn { Ne, cn }
```

(*__hook_tl_gput_right:Nn* 定义结束。)

__hook_tl_gput_left:Nn 与上述相同。

```
\cs_new_protected:Npn \__hook_tl_gput_left:Nn #1#2
  {
    \__hook_tl_gset:Nx #1
      { \__kernel_exp_not:w {#2} \__kernel_exp_not:w \exp_after:wN {#1} }
  }
```

(*__hook_tl_gput_left:Nn* 定义结束。)

__hook_tl_gset_eq:NN 与上述相同。

```
\cs_new_eq:NN \__hook_tl_gset_eq:NN \tl_gset_eq:NN
```

(*__hook_tl_gset_eq:NN* 定义结束。)

__hook_tl_gclear:N
__hook_tl_gclear:c 与上述相同。

```
\cs_new_protected:Npn \__hook_tl_gclear:N #1
  { \__hook_tl_gset_eq:NN #1 \c_empty_tl }
\cs_generate_variant:Nn \__hook_tl_gclear:N { c }
```

(*__hook_tl_gclear:N* 定义结束。)

4.4 提供新的钩子

4.4.1 钩子的数据结构

\g_@@_⟨*hook*⟩_code_prop
\@@␣⟨*hook*⟩
\g_@@_⟨*hook*⟩_reversed_tl
\g_@@_⟨*hook*⟩_declared_tl
\g_@@_⟨*hook*⟩_parameter_tl
\@@_next␣⟨*hook*⟩
\@@_toplevel␣⟨*hook*⟩

钩子有一个名称（在下面的描述中称为⟨*hook*⟩），对于每个钩子，我们必须提供一些数据结构。这些包括：

\g__hook_⟨*hook*⟩_code_prop 一个属性列表，保存钩子的代码，分开存储。键默认为添加代码的包名称，但包可以定义其他键。

\g__hook_⟨*hook*⟩_rule_⟨*label1*⟩|⟨*label2*⟩_tl 一个记号列表,保存⟨*hook*⟩中⟨*label1*⟩和⟨*label2*⟩之间的关系。这些⟨*label*⟩按词法（逆序）排序，以确保两个标签始终指向相同的记号列表。对于全局规则，⟨*hook*⟩名称为??。

__hook␣⟨*hook*⟩ 当文档中调用钩子时实际执行的代码存储在这个记号列表中。它是由应用信息的代码块构建而成。这个记号列表命名为这样是为了在钩子内部出现错误时，报告的错误记号列表更短，并且使得在__hook_make_name:n中规范化钩子名称更简单。

\g__hook_⟨*hook*⟩_reversed_tl 一些钩子是“反转”的。这个记号列表存储了这样的钩子的-，以便识别。使用-字符是因为 ⟨*reversed*⟩1 对于普通钩子是 +1，对于反转的钩子是 −1。

\g__hook_⟨*hook*⟩_declared_tl 这个记号列表作为标记用于正式声明钩子。如果尝试进行另一个声明，则会检查其存在性并引发错误。

\c__hook_⟨*hook*⟩_parameter_tl 这个记号列表保存已声明钩子的参数文本(它的存在几乎完全与上面的记号列表相交)，用于管理带参数的钩子。

__hook_toplevel␣⟨*hook*⟩ 这个记号列表存储从用户文档中插入到钩子中的代码，位于top-level标签中。这个标签是特殊的，不参与排序。相反，所有代码都被附加到它，并在（如果钩子是反转的，则在）正常的钩子代码之后（或之前）执行，但在next代码块之前。

__hook_next␣⟨*hook*⟩ 最后，有一些额外的代码（通常为空），用于下一次调用钩子时使用（然后删除)。这可用于在文档内部针对单个情况定义一些特殊行为。这个记号列表遵循与主__hook␣⟨*hook*⟩记号列表相同的命名方案。它被称为__hook_next␣⟨*hook*⟩,而不是__hook␣next_⟨*hook*⟩,因为否则名为next_⟨*hook*⟩的钩子将与称为⟨*hook*⟩的钩子的下一个代码记号列表冲突。

4.4.2 关于钩子的存在性

一个钩子可能处于不同的存在状态。在这里，我们给出了设置钩子的内部命令的概述，并解释了如何区分不同的状态。实际的实现随后在后续的章节中进行。

我们需要解决的一个问题是，即使代码尚未声明，我们也需要能够向钩子（例如，使用\AddToHook）添加代码。例如，一个包需要写入另一个包的钩子，但是该包可能不会被加载，或者仅在稍后加载。另一个问题是，大多数钩子（但不是通用钩子）需要进行声明。

因此,我们区分了以下钩子的状态,由四个不同的测试管理:结构存在(__hook_if_structure_exist:nTF)、创建（__hook_if_usable:nTF)、声明（__hook_if_declared:nTF）和禁用或未禁用（__hook_if_disabled:nTF)。

不存在 目前对于钩子还一无所知。可以使用__hook_if_structure_exist:nTF(使用假分支）来检测到此状态。在此状态下，可以声明钩子、禁用钩子、定义规则或向其中添加代码，但无法使用\UseHook。

基本数据结构设置 当钩子的基本数据结构设置完成时(使用__hook_init_structure:n),钩子处于此状态。数据结构设置发生在自动进行命令（例如\AddToHook）使用时，此时钩子处于“不存在”状态。在此状态下，四个测试的结果如下:

__hook_if_structure_exist:nTF 返回 true。

__hook_if_usable:nTF 返回 false。

__hook_if_declared:nTF 返回 false。

__hook_if_disabled:nTF 返回 false。

允许的操作与“不存在”状态下相同。

已声明 当钩子未被禁用且显式声明（例如，使用\NewHook）时，钩子处于此状态。在此情况下，四个测试的结果如下：

__hook_if_structure_exist:nTF 返回 true。

__hook_if_usable:nTF 返回 true。

__hook_if_declared:nTF 返回 true。

__hook_if_disabled:nTF 返回 false。

可用 当钩子未被禁用且未显式声明，但仍允许使用（使用\UseHook或\hook_use:n）时，钩子处于此状态。此状态仅适用于通用钩子，因为它们无需声明。因此，这样的钩子直接从“不存在”状态转移到“可用”状态，这是因为诸如\AddToHook之类的声明要添加到钩子数据结构。在此状态下，测试的结果如下：

__hook_if_structure_exist:nTF 返回 true。

__hook_if_usable:nTF 返回 true。

__hook_if_declared:nTF 返回 false。

__hook_if_disabled:nTF 返回 false。

禁用 任何状态中的通用钩子在使用\DisableGenericHook后移动到此状态。这会改变测试的结果如下：

__hook_if_structure_exist:nTF 不变。

__hook_if_usable:nTF 返回 false。

__hook_if_declared:nTF 返回 true。

__hook_if_disabled:nTF 返回 true。

结构测试不变（如果钩子以前未知，则为false，否则为true）。可用测试返回false，因此从现在开始，任何\UseHook都将跳过该钩子。声明测试返回true，因此任何进一步的\NewHook将生成错误，并且禁用测试返回true，以便\AddToHook可以返回错误。

FMi: 也许它只应该在文档开始后执行？

4.4.3 设置钩子

\hook_new:n
\hook_new_with_args:nn
__hook_new:nn

\hook_new:n声明一个新的钩子,并期望钩子名称⟨*name*⟩作为其参数,例如,begindocument.

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\hook_new_with_args:nn}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \hook_new:n #1
  { \__hook_normalize_hook_args:Nn \__hook_new:nn {#1} { 0 } }
\cs_new_protected:Npn \hook_new_with_args:nn #1 #2
  { \__hook_normalize_hook_args:Nn \__hook_new:nn {#1} {#2} }

\cs_new_protected:Npn \__hook_new:nn #1 #2
  {
```

我们检查钩子是否已经被显式使用\hook_new:n声明过，如果已存在，则发出警告，否则为钩子设置“created”标志，以便下次使用\hook_new:n时产生错误。

```
    \__hook_if_declared:nTF {#1}
      { \msg_error:nnn { hooks } { exists } {#1} }
      {
        \tl_new:c { g__hook_#1_declared_tl }
        \cs_undefine:c { __hook~#1 }
        \cs_undefine:c { c__hook_#1_parameter_tl }
        \__hook_make_usable:nn {#1} {#2}
```

如果钩子中已经有代码，但尚未声明，运行__hook_update_hook_code:n以使其准备好执行（参见测试 lthooks-034）。

```
        \__hook_update_hook_code:n {#1}
      }
  }
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\hook_new_with_args:nn}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_gset_protected:Npn \hook_new:n #1
⟨latexrelease⟩  { \__hook_normalize_hook_args:Nn \__hook_new:n {#1} }
⟨latexrelease⟩\cs_undefine:N \__hook_new:nn
⟨latexrelease⟩\cs_gset_protected:Npn \__hook_new:n #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_if_declared:nTF {#1}
⟨latexrelease⟩      { \msg_error:nnn { hooks } { exists } {#1} }
⟨latexrelease⟩      {
⟨latexrelease⟩        \tl_new:c { g__hook_#1_declared_tl }
⟨latexrelease⟩        \__hook_make_usable:n {#1}
⟨latexrelease⟩      }
```

```
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_gset_protected:Npn \hook_new_with_args:nn #1 { }
⟨latexrelease⟩\EndIncludeInRelease
```

(\hook_new:n，\hook_new_with_args:nn，和 __hook_new:nn 定义结束。这些函数被记录在第15页。)

__hook_make_usable:nn

这为钩子初始化了所有钩子数据结构，但如果单独使用，则不会将钩子标记为已声明（如\hook_new:n所做的那样，因此后续对该钩子的\hook_new:n不会导致错误。此命令在向通用钩子添加代码时由\hook_gput_code:nnn内部使用。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_make_usable:nn}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \__hook_make_usable:nn #1 #2
  {
```

现在我们检查钩子的数据结构是否可以安全地创建，而不会引发 expl3 的错误，然后将钩子名称添加到所有钩子的列表中，并为新钩子分配必要的数据结构，否则什么也不做。

```
    \__hook_if_usable:nF {#1}
      {
        \seq_gput_right:Nn \g__hook_all_seq {#1}
```

在这里，我们将定义\c__hook_⟨hook⟩_parameter_tl，以保存参数的一系列直到钩子的参数数量（#2）。

```
        \__kernel_cs_parm_from_arg_count:nnF
          { \tl_const:cn { c__hook_#1_parameter_tl } } {#2}
          {
            \msg_error:nnnn { hooks } { too-many-args } {#1} {#2}
            \tl_const:cx { c__hook_#1_parameter_tl }
              { \exp_not:V \c__hook_nine_parameters_tl }
          }
```

在此之后，使用 __hook_normalise_cs_args:nn 来修正宏的参数数量 __hook_toplevel␣⟨hook⟩ 和 __hook_next␣⟨hook⟩。我们需要能够向一个钩子添加带有参数的代码，而不需要提前知道该钩子的参数数量，因此 lthooks 假定为 9，直到该钩子被正确声明并知道参数数量为止。__hook_normalise_cs_args:nn 使用刚刚上面定义的 \c__hook_⟨hook⟩_parameter_tl 进行规范化。

```
    \__hook_normalise_cs_args:nn { _toplevel } {#1}
    \__hook_normalise_cs_args:nn { _next } {#1}
```

这仅由当前钩子的实际代码使用，因此正常声明它：

```
    \__hook_code_gset:nn {#1} { }
```

现在确保钩子的基本数据结构存在：

```
    \__hook_init_structure:n {#1}
```

调用 `\__hook_normalise_code_pool:n` 将纠正对钩子中不存在的参数的引用，引发低级 TEX 错误并使有问题的参数标记翻倍。必须在 `\__hook_init_structure:n` 之后执行，因为它作用于 `\g__hook_⟨hook⟩_code_prop`。

```
    \__hook_normalise_code_pool:n {#1}
```

`\g__hook_⟨hook⟩_labels_clist` 存储了标签的排序列表（一旦排序）。这仅用于调试。这些是有条件地定义的，以防正在使用 `\__hook_make_usable:nn` 重新定义一个钩子。

```
      \clist_if_exist:cF { g__hook_#1_labels_clist }
        {
          \clist_new:c { g__hook_#1_labels_clist }
```

一些钩子应该反转代码块的默认顺序。为了表示这一点，我们有一个令牌列表，对于普通钩子为空，对于反转的钩子包含一个 -。

```
    \tl_new:c { g__hook_#1_reversed_tl }
    }
```

上述内容都是遵循 L3 规范，但我们还提供了一个接口来处理类似 LATEX 2_ε 旧版钩子的形式，如 `\@...hook`，例如 `\@begindocumenthook`。有一些这样的钩子，并且它们已经被添加到使用 `\g@addto@macro`。如果存在一个与新钩子名称匹配的这样的宏，即 `\@⟨hook-name⟩hook`，并且它不为空，则我们将其内容作为代码块添加到标签 `legacy` 下。

警告：这种支持将在未来版本中消失！

```
      \__hook_include_legacy_code_chunk:n {#1}
    }
  }
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_make_usable:nn}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_undefine:N \__hook_make_usable:nn
⟨latexrelease⟩\cs_gset_protected:Npn \__hook_make_usable:n #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \tl_if_exist:cF { __hook~#1 }
⟨latexrelease⟩      {
⟨latexrelease⟩        \seq_gput_right:Nn \g__hook_all_seq {#1}
⟨latexrelease⟩        \tl_new:c { __hook~#1 }
⟨latexrelease⟩        \__hook_init_structure:n {#1}
⟨latexrelease⟩        \clist_new:c { g__hook_#1_labels_clist }
⟨latexrelease⟩        \tl_new:c { g__hook_#1_reversed_tl }
⟨latexrelease⟩        \__hook_include_legacy_code_chunk:n {#1}
```

```
⟨latexrelease⟩      }
⟨latexrelease⟩  }
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_make_usable:nn* 定义结束。)

__hook_init_structure:n

这个函数声明了一个钩子的基本数据结构，而不是显式地声明钩子本身。这是为了允许向未声明的钩子添加内容。在这里，不需要检查所有变量是否存在，因为所有三个变量同时被声明（它们要么都存在，要么都不存在）。

它创建了钩子代码池（\g__hook_⟨*hook*⟩_code_prop），以及 top-level 和 next 令牌列表。第一次向钩子添加内容时，使用 __hook_init_structure:n 对钩子进行初始化。只使用 __hook_init_structure:n 初始化钩子并不能使其能够被 \hook_use:n 使用。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_init_structure:n}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \__hook_init_structure:n #1
  {
    \__hook_if_structure_exist:nF {#1}
      {
        \prop_new:c { g__hook_#1_code_prop }
        \__hook_toplevel_gset:nn {#1} { }
        \__hook_next_gset:nn {#1} { }
      }
  }
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_init_structure:n}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_gset_protected:Npn \__hook_init_structure:n #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_if_structure_exist:nF {#1}
⟨latexrelease⟩      {
⟨latexrelease⟩        \prop_new:c { g__hook_#1_code_prop }
⟨latexrelease⟩        \tl_new:c { __hook_toplevel~#1 }
⟨latexrelease⟩        \tl_new:c { __hook_next~#1 }
⟨latexrelease⟩      }
⟨latexrelease⟩  }
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_init_structure:n* 定义结束。)

\hook_new_reversed:n
\hook_new_reversed_with_args:nn
__hook_new_reversed:nn

声明一个新的钩子。代码块的默认顺序是反向的，通过将令牌列表设置为减号来表示。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\hook_new_reversed_with_args:nn}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \hook_new_reversed:n #1
  { \__hook_normalize_hook_args:Nn \__hook_new_reversed:nn {#1} { 0 } }
\cs_new_protected:Npn \hook_new_reversed_with_args:nn #1 #2
  { \__hook_normalize_hook_args:Nn \__hook_new_reversed:nn {#1} {#2} }
\cs_new_protected:Npn \__hook_new_reversed:nn #1 #2
  {
    \__hook_if_declared:nTF {#1}
      { \msg_error:nnn { hooks } { exists } {#1} }
      {
        \__hook_new:nn {#1} {#2}
        \tl_gset:cn { g__hook_#1_reversed_tl } { - }
      }
  }
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\hook_new_reversed_with_args:nn}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_gset_protected:Npn \hook_new_reversed:n #1
⟨latexrelease⟩  { \__hook_normalize_hook_args:Nn \__hook_new_reversed:n {#1} }
⟨latexrelease⟩\cs_undefine:N \__hook_new_reversed:nn
⟨latexrelease⟩\cs_gset_protected:Npn \__hook_new_reversed:n #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_new:n {#1}
⟨latexrelease⟩    \tl_gset:cn { g__hook_#1_reversed_tl } { - }
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_undefine:N \__hook_new_reversed:nn
⟨latexrelease⟩\cs_gset_protected:Npn \hook_new_reversed_with_args:nn #1 #2 { }
⟨latexrelease⟩\EndIncludeInRelease
```

(*\hook_new_reversed:n*, *\hook_new_reversed_with_args:nn*, 和 *__hook_new_reversed:nn* 定义结束。这些函数被记录在第15页。)

\hook_new_pair:nn
\hook_new_pair_with_args:nnn

一种一次性声明普通和（匹配的）反向钩子的简写方式。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\hook_new_pair_with_args:nnn}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \hook_new_pair:nn #1#2
  { \__hook_normalize_hook_args:Nnn \__hook_new_pair:nnn {#1} {#2} { 0 } }
\cs_new_protected:Npn \hook_new_pair_with_args:nnn #1#2#3
  { \__hook_normalize_hook_args:Nnn \__hook_new_pair:nnn {#1} {#2} {#3} }
\cs_new_protected:Npn \__hook_new_pair:nnn #1 #2 #3
  {
```

```
    \__hook_if_declared:nTF {#1}
      { \msg_error:nnn { hooks } { exists } {#1} }
      {
        \__hook_if_declared:nTF {#2}
          { \msg_error:nnn { hooks } { exists } {#2} }
          {
            \__hook_new:nn {#1} {#3}
            \__hook_new_reversed:nn {#2} {#3}
          }
      }
  }
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\hook_new_pair_with_args:nnn}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_gset_protected:Npn \hook_new_pair:nn #1#2
⟨latexrelease⟩  {
⟨latexrelease⟩    \hook_new:n {#1}
⟨latexrelease⟩    \hook_new_reversed:n {#2}
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_gset_protected:Npn \hook_new_pair_with_args:nnn #1#2#3
⟨latexrelease⟩  { }
⟨latexrelease⟩\EndIncludeInRelease
```

(*\hook_new_pair:nn* 和 *\hook_new_pair_with_args:nnn* 定义结束。这些函数被记录在第15页。)

__hook_include_legacy_code_chunk:n

LaTeX 遗留钩子的概念在代码中使用以下命名方案：\@...hook。

如果此宏不为空，我们将其添加到当前钩子的标签 legacy 下，然后在全局范围内将其清空。这样，直接操作诸如 \@begindocumenthook 等命令的包或类仍然可以添加其钩子数据。

警告：这种支持将在未来版本中消失！

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_include_legacy_code_chunk:n}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \__hook_include_legacy_code_chunk:n #1
  {
```

如果宏不存在（通常情况下是这样），则不需要进行任何操作。

```
    \tl_if_exist:cT { @#1hook }
      {
```

当然，如果遗留钩子存在但为空，就没有必要在遗留标签下添加任何内容。

```
        \tl_if_empty:cF { @#1hook }
          {
```

在这里，我们设置 __hook_replacing_args_false:，因为没有遗留代码会引用钩子参数。

```
            \__hook_replacing_args_false:
            \use:e
              {
                \__hook_hook_gput_code_do:nnn {#1} { legacy }
                  { \exp_not:v { @#1hook } }
              }
            \__hook_replacing_args_reset:
```

添加到钩子后，我们需要将其清除，否则如果钩子数据更新，可能会在以后再次添加。

```
            \__hook_tl_gclear:c { @#1hook }
          }
      }
  }
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_include_legacy_code_chunk:n}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_gset_protected:Npn \__hook_include_legacy_code_chunk:n #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \tl_if_exist:cT { @#1hook }
⟨latexrelease⟩      {
⟨latexrelease⟩        \tl_if_empty:cF { @#1hook }
⟨latexrelease⟩          {
⟨latexrelease⟩            \exp_args:Nnnv \__hook_hook_gput_code_do:nnn
⟨latexrelease⟩              {#1} { legacy } { @#1hook }
⟨latexrelease⟩            \__hook_tl_gclear:c { @#1hook }
⟨latexrelease⟩          }
⟨latexrelease⟩      }
⟨latexrelease⟩  }
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_include_legacy_code_chunk:n* 定义结束。)

4.4.4 禁用和提供钩子

\hook_disable_generic:n
__hook_disable:n
__hook_if_disabled_p:n
__hook_if_disabled:nTF

通过创建其 \g__hook_⟨*hook*⟩_declared_tl 来禁用一个钩子，以便在使用 \hook_new:n 时出错，然后取消定义 __hook␣⟨*hook*⟩，以防止其执行。

这并不清除可能已经存储在钩子结构中的任何代码，但不允许添加更多代码。__hook_if_disabled:nTF 使用该特定组合来检查钩子是否已禁用。

```
⟨latexrelease⟩\IncludeInRelease{2021/06/01}{\hook_disable_generic:n}
⟨latexrelease⟩                  {Disable~hooks}
```

```
\cs_new_protected:Npn \hook_disable_generic:n #1
  { \__hook_normalize_hook_args:Nn \__hook_disable:n {#1} }
\cs_new_protected:Npn \__hook_disable:n #1
  {
    \tl_gclear_new:c { g__hook_#1_declared_tl }
    \cs_undefine:c { __hook~#1 }
  }
\prg_new_conditional:Npnn \__hook_if_disabled:n #1 { p, T, F, TF }
  {
    \bool_lazy_and:nnTF
        { \tl_if_exist_p:c { g__hook_#1_declared_tl } }
        { ! \cs_if_exist_p:c { __hook~#1 } }
      { \prg_return_true: }
      { \prg_return_false: }
  }
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\hook_disable_generic:n}
⟨latexrelease⟩                  {Disable~hooks}
⟨latexrelease⟩
⟨latexrelease⟩\cs_new_protected:Npn \hook_disable_generic:n #1 {}
⟨latexrelease⟩
⟨latexrelease⟩\EndIncludeInRelease
```

(*\hook_disable_generic:n*，*__hook_disable:n*，和 *__hook_if_disabled:nTF* 定义结束。这个函数被记录在第16页。)

\hook_activate_generic:n
__hook_activate_generic:n

\hook_activate_generic:n 声明了一个新的钩子，如果它还没有被声明的话，此时它仅检查已经存在的钩子是否不是一个反向钩子。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\hook_activate_generic:n}
⟨latexrelease⟩                  {Providing~hooks}

\cs_new_protected:Npn \hook_activate_generic:n #1
  { \__hook_normalize_hook_args:Nn \__hook_activate_generic:nn {#1} {   } }

\cs_new_protected:Npn \__hook_activate_generic:nn #1 #2
  {
```

如果要激活的钩子已被禁用，我们发出警告（目前是这样的 — 这可能会改变）。

```
    \__hook_if_disabled:nTF {#1}
      { \msg_warning:nnn { hooks } { activate-disabled } {#1} }
```

否则，我们检查钩子是否未被声明，如果是，则确定它是反向的还是非反向的，然后相应地进行声明。

```
      {
        \__hook_if_declared:nF {#1}
          {
            \tl_new:c { g__hook_#1_declared_tl }
            \__hook_make_usable:nn {#1} { 0 }
            \tl_gset:cx { g__hook_#1_reversed_tl }
              { \__hook_if_generic_reversed:nT {#1} { - } }
```

反映我们已激活通用钩子并设置其执行代码。

```
            \__hook_update_hook_code:n {#1}
          }
      }
  }

⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2021/06/01}{\hook_activate_generic:n}
⟨latexrelease⟩                  {Providing~hooks}
⟨latexrelease⟩\cs_gset_protected:Npn \__hook_activate_generic:nn #1 #2
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_if_disabled:nTF {#1}
⟨latexrelease⟩      { \msg_warning:nnn { hooks } { activate-disabled } {#1} }
⟨latexrelease⟩      {
⟨latexrelease⟩        \__hook_if_declared:nF {#1}
⟨latexrelease⟩          {
⟨latexrelease⟩            \tl_new:c { g__hook_#1_declared_tl }
⟨latexrelease⟩            \__hook_make_usable:n {#1}
⟨latexrelease⟩            \tl_gset:cx { g__hook_#1_reversed_tl }
⟨latexrelease⟩              { \__hook_if_generic_reversed:nT {#1} { - } }
⟨latexrelease⟩            \__hook_update_hook_code:n {#1}
⟨latexrelease⟩          }
⟨latexrelease⟩      }
⟨latexrelease⟩  }
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\hook_activate_generic:n}
⟨latexrelease⟩                  {Providing~hooks}
⟨latexrelease⟩\cs_gset_protected:Npn \hook_activate_generic:n #1 { }
⟨latexrelease⟩\EndIncludeInRelease
```

(*\hook_activate_generic:n* 和 *__hook_activate_generic:n* 定义结束。这个函数被记录在第16页。)

4.5 解析标签

__hook_parse_label_default:n

此宏检查是否提供了标签（不是 \c_novalue_tl），如果提供了标签，则尝试解析标签以查找一个前导的 .，将其替换为 __hook_currname_or_default:。

```
\cs_new:Npn \__hook_parse_label_default:n #1
  {
    \tl_if_novalue:nTF {#1}
      { \__hook_currname_or_default: }
      { \tl_trim_spaces_apply:nN {#1} \__hook_parse_dot_label:n }
  }
```

（*__hook_parse_label_default:n 定义结束。*）

__hook_parse_dot_label:n
__hook_parse_dot_label:w
__hook_parse_dot_label_cleanup:w
__hook_parse_dot_label_aux:w

首先检查标签是否为空，如果为空则引发错误，并使用回退值。如果不为空，在可能的情况下将标签分割为 ./，并检查在 ./ 前是否没有任何标记，或者前面的唯一字符是 .。如果满足这些要求，则将前导的 . 替换为 __hook_currname_or_default:。否则，将标签不变地返回。

```
\cs_new:Npn \__hook_parse_dot_label:n #1
  {
    \tl_if_empty:nTF {#1}
      {
        \msg_expandable_error:nn { hooks } { empty-label }
        \__hook_currname_or_default:
      }
      {
        \str_if_eq:nnTF {#1} { . }
          { \__hook_currname_or_default: }
          { \__hook_parse_dot_label:w #1 ./ \s__hook_mark }
      }
  }
\cs_new:Npn \__hook_parse_dot_label:w #1 ./ #2 \s__hook_mark
  {
    \tl_if_empty:nTF {#1}
      { \__hook_parse_dot_label_aux:w #2 \s__hook_mark }
      {
        \tl_if_empty:nTF {#2}
          { \__hook_make_name:n {#1} }
          { \__hook_parse_dot_label_cleanup:w #1 ./ #2 \s__hook_mark }
      }
  }
\cs_new:Npn \__hook_parse_dot_label_cleanup:w #1 ./ \s__hook_mark {#1}
\cs_new:Npn \__hook_parse_dot_label_aux:w #1 ./ \s__hook_mark
```

```
{ \__hook_currname_or_default: / \__hook_make_name:n {#1} }
```

(*__hook_parse_dot_label:n 以及其它的定义结束。*)

__hook_currname_or_default:

如果设置了 \g__hook_hook_curr_name_tl，则使用它；否则尝试使用 \@currname。如果两者都没有设置，则引发错误并使用回退值 label-missing。

```
\cs_new:Npn \__hook_currname_or_default:
  {
    \tl_if_empty:NTF \g__hook_hook_curr_name_tl
      {
        \tl_if_empty:NTF \@currname
          {
            \msg_expandable_error:nnn { latex2e } { should-not-happen }
              { Empty~default~label. }
            \__hook_make_name:n { label-missing }
          }
          { \@currname }
      }
      { \g__hook_hook_curr_name_tl }
  }
```

(*__hook_currname_or_default: 定义结束。*)

__hook_make_name:n
__hook_make_name:w

这提供了钩子名称的标准清理。它使用 \cs:w 构建钩子名称的控制序列，然后使用 \cs_to_str:N 获取其字符串表示，不包括转义字符。使用 \cs:w 的扩展方式，而不是 e 的方式，因为 Unicode 字符在 \expanded 内部的行为不佳。该宏将 __hook␣ 前缀添加到钩子名称，以重用钩子的代码标记列表来构建控制序列，并避免在 TeX 内存中留下“公共”控制序列（如 \relax）。

```
\cs_new:Npn \__hook_make_name:n #1
  {
    \exp_after:wN \exp_after:wN \exp_after:wN \__hook_make_name:w
    \exp_after:wN \token_to_str:N \cs:w __hook~ #1 \cs_end:
  }
\exp_last_unbraced:NNNNo
\cs_new:Npn \__hook_make_name:w #1 \tl_to_str:n { __hook~ } { }
```

(*__hook_make_name:n 和 __hook_make_name:w 定义结束。*)

__hook_normalize_hook_args:Nn
__hook_normalize_hook_args:Nnn
__hook_normalize_hook_rule_args:Nnnnn
__hook_normalize_hook_args_aux:Nn

这是规范化钩子和标签参数的标准途径。主要宏在一个组内执行整个操作，这样通过 __hook_make_name:n 创建的控制序列在继续之前会被清除。这意味着此函数不能用于 \hook_use:n!

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

```
  {
    \group_begin:
    \use:e
      {
        \group_end:
        \exp_not:N #1 #2
      }
  }
\cs_new_protected:Npn \__hook_normalize_hook_args:Nn #1 #2
  {
    \__hook_normalize_hook_args_aux:Nn #1
      { { \__hook_parse_label_default:n {#2} } }
  }
\cs_new_protected:Npn \__hook_normalize_hook_args:Nnn #1 #2 #3
  {
    \__hook_normalize_hook_args_aux:Nn #1
      {
        { \__hook_parse_label_default:n {#2} }
        { \__hook_parse_label_default:n {#3} }
      }
  }
\cs_new_protected:Npn \__hook_normalize_hook_rule_args:Nnnnn #1 #2 #3 #4 #5
  {
    \__hook_normalize_hook_args_aux:Nn #1
      {
        { \__hook_parse_label_default:n {#2} }
        { \__hook_parse_label_default:n {#3} }
        { \tl_trim_spaces:n {#4} }
        { \__hook_parse_label_default:n {#5} }
      }
  }
```

(*__hook_normalize_hook_args:Nn 以及其它的定义结束。*)

__hook_curr_name_push:n
__hook_curr_name_push_aux:n
__hook_curr_name_pop:
__hook_end_document_label_check:

符号列表 `\g__hook_hook_curr_name_tl` 存储当前要用作钩子默认标签的包名/文件名。提供一致的接口很棘手，因为包可以在包中加载，并且一些包可能不使用 `\SetDefaultHookLabel` 来更改默认标签（这种情况下使用 `\@currname`）。

为了实现这一点，我们保持一个堆栈，其中包含每个输入级别的默认标签。堆栈底部包含 `top-level` 的默认标签（此堆栈不应为空）。如果正在构建格式，则将默认标签设置为 `top-level`：

```
\tl_gset:Nn \g__hook_hook_curr_name_tl { top-level }
```

然后，在我们位于 latexrelease 中的情况下，我们在堆栈上推送一些内容以支持向前滚动。但在一些罕见的情况下，latexrelease 可能会在另一个包中加载（尤其是 platexrelease），因此我们首先推送 top-level 条目：

```
⟨latexrelease⟩\seq_if_empty:NT \g__hook_name_stack_seq
⟨latexrelease⟩  { \seq_gput_right:Nn \g__hook_name_stack_seq { top-level } }
```

然后我们分析 \@currnamestack，将 \@currname 添加到堆栈：

```
⟨latexrelease⟩\cs_set_protected:Npn \__hook_tmp:w #1 #2 #3
⟨latexrelease⟩  {
⟨latexrelease⟩    \quark_if_recursion_tail_stop:n {#1}
⟨latexrelease⟩    \seq_gput_right:Nn \g__hook_name_stack_seq {#1}
⟨latexrelease⟩    \__hook_tmp:w
⟨latexrelease⟩  }
⟨latexrelease⟩\exp_after:wN \__hook_tmp:w \@currnamestack
⟨latexrelease⟩  \q_recursion_tail \q_recursion_tail
⟨latexrelease⟩  \q_recursion_tail \q_recursion_stop
```

最后将默认标签设置为 \@currname：

```
⟨latexrelease⟩\tl_gset:Nx \g__hook_hook_curr_name_tl { \@currname }
⟨latexrelease⟩\seq_gpop_right:NN \g__hook_name_stack_seq \l__hook_tmpa_tl
```

有两个命令来跟踪堆栈：当输入文件时，__hook_curr_name_push:n 将当前默认标签推送到堆栈上并设置新的默认标签（一次完成所有操作）：

```
\cs_new_protected:Npn \__hook_curr_name_push:n #1
  { \exp_args:Nx \__hook_curr_name_push_aux:n { \__hook_make_name:n {#1} } }
\cs_new_protected:Npn \__hook_curr_name_push_aux:n #1
  {
    \tl_if_blank:nTF {#1}
      { \msg_error:nn { hooks } { no-default-label } }
      {
        \str_if_eq:nnTF {#1} { top-level }
          {
            \msg_error:nnnnn { hooks } { set-top-level }
              { to } { PushDefaultHookLabel } {#1}
          }
          {
            \seq_gpush:NV \g__hook_name_stack_seq \g__hook_hook_curr_name_tl
            \tl_gset:Nn \g__hook_hook_curr_name_tl {#1}
          }
      }
  }
```

当输入结束时，堆栈的顶部项目将被弹出，因为该标签将不再使用，并且 \g__hook_hook_curr_name_tl 被更新为现在堆栈的顶部项目：

```
\cs_new_protected:Npn \__hook_curr_name_pop:
  {
    \seq_gpop:NNTF \g__hook_name_stack_seq \l__hook_return_tl
      { \tl_gset_eq:NN \g__hook_hook_curr_name_tl \l__hook_return_tl }
      { \msg_error:nn { hooks } { extra-pop-label } }
  }
```

在文档结束时，我们想要检查是否有没有匹配的 __hook_curr_name_push:n 和 __hook_curr_name_pop:（这不是一个关键错误，但可能表明其他地方存在问题）：

```
\tl_gput_right:Nn \@kernel@after@enddocument@afterlastpage
  { \__hook_end_document_label_check: }
\cs_new_protected:Npn \__hook_end_document_label_check:
  {
    \seq_gpop:NNT \g__hook_name_stack_seq \l__hook_return_tl
      {
        \msg_error:nnx { hooks } { missing-pop-label }
          { \g__hook_hook_curr_name_tl }
        \tl_gset_eq:NN \g__hook_hook_curr_name_tl \l__hook_return_tl
        \__hook_end_document_label_check:
      }
  }
```

符号列表 \g__hook_hook_curr_name_tl 仅是堆栈顶部的镜像。

现在定义一个包装器，用参数替换堆栈的顶部项目，并相应地更新 \g__hook_hook_curr_name_tl。

__hook_set_default_hook_label:n
__hook_set_default_label:n

```
\cs_new_protected:Npn \__hook_set_default_hook_label:n #1
  {
    \seq_if_empty:NTF \g__hook_name_stack_seq
      {
        \msg_error:nnnnn { hooks } { set-top-level }
          { for } { SetDefaultHookLabel } {#1}
      }
      { \exp_args:Nx \__hook_set_default_label:n { \__hook_make_name:n {#1} } }
  }
\cs_new_protected:Npn \__hook_set_default_label:n #1
  {
    \str_if_eq:nnTF {#1} { top-level }
      {
        \msg_error:nnnnn { hooks } { set-top-level }
```

```
          { to } { SetDefaultHookLabel } {#1}
      }
      { \tl_gset:Nn \g__hook_hook_curr_name_tl {#1} }
  }
```

(*__hook_curr_name_push:n* 以及其它的定义结束。)

4.6 添加或移除钩子代码

\hook_gput_code:nnn
\hook_gput_code_with_args:nnn
__hook_gput_code:nnn
__hook_gput_code_store:nnn
__hook_hook_gput_code_do:nnn
__hook_prop_gput_labeled_cleanup:nnn
__hook_prop_gput_labeled_do:Nnnn

使用 \hook_gput_code:nnn{⟨*hook*⟩}{⟨*label*⟩}{⟨*code*⟩}，将一段 ⟨*code*⟩ 添加到已存在的带有 ⟨*label*⟩ 标签的 ⟨*hook*⟩ 中。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\hook_gput_code:nnn}
⟨latexrelease⟩                          {Hooks~with~args}
\cs_new_protected:Npn \hook_gput_code:nnn #1 #2 #3
  {
    \__hook_replacing_args_false:
    \__hook_normalize_hook_args:Nnn \__hook_gput_code:nnn {#1} {#2} {#3}
    \__hook_replacing_args_reset:
  }
\cs_new_protected:Npn \hook_gput_code_with_args:nnn #1 #2 #3
  {
    \__hook_replacing_args_true:
    \__hook_normalize_hook_args:Nnn \__hook_gput_code:nnn {#1} {#2} {#3}
    \__hook_replacing_args_reset:
  }
```

如果使用了 \AddToHookWithArguments，进行一些合理性检查，如果此时不能使用参数，通过使用 __hook_replacing_args_false: 回退到常规的 \AddToHook。

```
\cs_new_protected:Npn \__hook_gput_code:nnn #1 #2 #3
  {
    \__hook_chk_args_allowed:nn {#1} { AddToHook }
```

然后检查代码是否应立即执行，而不是存储：

```
    \__hook_if_execute_immediately:nTF {#1}
      {
```

\AddToHookWithArguments 不能用于一次性钩子（已经使用过的）。

```
        \__hook_if_replacing_args:TF
          {
            \msg_error:nnnn { hooks } { one-time-args }
              {#1} { AddToHook }
          }
```

```
          { }
        \use:n
      }
      { \__hook_gput_code_store:nnn {#1} {#2} }
        {#3}
  }
\cs_new_protected:Npn \__hook_gput_code_store:nnn #1 #2 #3
  {
```

然后检查钩子是否可用。

```
    \__hook_if_usable:nTF {#1}
```

如果可用，我们只需将新代码添加（或附加）到保存钩子不同代码块的属性列表中。在 `\begin{document}` 时，这些代码将被排序到一个令牌列表中以便快速执行。

```
      {
        \__hook_hook_gput_code_do:nnn {#1} {#2} {#3}
```

但是，如果文档中有更新，我们需要修改此执行代码，这通过 `\__hook_update_hook_code:n` 完成。在导言部分，这将不起作用。

```
        \__hook_update_hook_code:n {#1}
      }
```

如果钩子不可用，在放弃之前，检查它是否未禁用，否则尝试将其声明为通用钩子，如果其名称匹配有效模式之一。

```
      {
        \__hook_if_disabled:nTF {#1}
          { \msg_error:nnn { hooks } { hook-disabled } {#1} }
          { \__hook_try_declaring_generic_hook:nnn {#1} {#2} {#3} }
      }
  }
```

此宏将无条件地向给定的钩子添加一段代码。

```
\cs_new_protected:Npn \__hook_hook_gput_code_do:nnn #1 #2 #3
  {
```

但首先，如果启用了调试功能，显示一些调试信息：

```
    \__hook_debug:n{\iow_term:x{****~ Add~ to~
                        \__hook_if_usable:nF {#1} { undeclared~ }
                        hook~ #1~ (#2)
                        \on@line\space <-- \tl_to_str:n{#3}} }
```

然后尝试从钩子中获取标记为 `#2` 的代码块。如果已经有代码存在，那么将 `#3` 附加到其中，否则只是放置 `#3`。如果当前标签为 `top-level`，则代码将被添加到一个专

用的令牌列表 __hook_toplevel␣⟨*hook*⟩ 中，该列表位于钩子的末尾（或者对于反向钩子，位于开头），就在 __hook_next␣⟨*hook*⟩ 之前。

```
    \str_if_eq:nnTF {#2} { top-level }
      {
        \str_if_eq:eeTF { top-level } { \__hook_currname_or_default: }
          {
```

如果钩子的基本结构不存在，则需要用 __hook_init_structure:n 声明它。

```
            \__hook_init_structure:n {#1}
```

然后将其附加到钩子的 _toplevel 容器中。

```
            \__hook_cs_gput_right:nnn { _toplevel } {#1} {#3}
          }
          { \msg_error:nnn { hooks } { misused-top-level } {#1} }
      }
      {
```

在添加到代码池时，如果使用了 \AddToHook（replacing_args 为 false），我们必须双重哈希，这样以后它会变成一个单参数令牌，而不是钩子宏的参数。

```
        \exp_args:Nx \__hook_prop_gput_labeled_cleanup:nnn
          {
            \__hook_if_replacing_args:TF
              { \exp_not:n }
              { \__hook_double_hashes:n }
                {#3}
          }
          {#1} {#2}
      }
  }
```

向钩子的代码池中添加代码。

```
\cs_new_protected:Npn \__hook_prop_gput_labeled_cleanup:nnn #1 #2 #3
  {
    \tl_set:Nn \l__hook_return_tl {#1}
    \__hook_if_replacing_args:TF
      {
        \__hook_if_usable:nT {#2}
          {
            \__hook_set_normalise_fn:nn {#2}
              { Invalid~code~added~\msg_line_context: }
            \__hook_normalise_fn:nn {#3} {#1}
            \prop_get:NnN \l__hook_work_prop {#3} \l__hook_return_tl
          }
      }
```

```
      { }
    \exp_args:NcV \__hook_prop_gput_labeled_do:Nnn
      { g__hook_#2_code_prop } \l__hook_return_tl {#3}
  }
\cs_new_protected:Npn \__hook_prop_gput_labeled_do:Nnn #1 #2 #3
  {
    \prop_get:NnNTF #1 {#3} \l__hook_return_tl
      { \prop_gput:Nno #1 {#3} { \l__hook_return_tl #2 } }
      { \prop_gput:Nnn #1 {#3} {#2} }
  }

⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\hook_gput_code:nnn}
⟨latexrelease⟩                  {Providing~hooks}
⟨latexrelease⟩\cs_gset_protected:Npn \hook_gput_code:nnn #1 #2
⟨latexrelease⟩  { \__hook_normalize_hook_args:Nnn \__hook_gput_code:nnn {#1} {#2} }
⟨latexrelease⟩\cs_gset_protected:Npn \__hook_gput_code:nnn #1 #2 #3
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_if_execute_immediately:nTF {#1}
⟨latexrelease⟩      {#3}
⟨latexrelease⟩      {
⟨latexrelease⟩        \__hook_if_usable:nTF {#1}
⟨latexrelease⟩          {
⟨latexrelease⟩            \__hook_hook_gput_code_do:nnn {#1} {#2} {#3}
⟨latexrelease⟩            \__hook_update_hook_code:n {#1}
⟨latexrelease⟩          }
⟨latexrelease⟩          {
⟨latexrelease⟩            \__hook_if_disabled:nTF {#1}
⟨latexrelease⟩              { \msg_error:nnn { hooks } { hook-disabled } {#1} }
⟨latexrelease⟩              { \__hook_try_declaring_generic_hook:nnn {#1} {#2} {#3} }
⟨latexrelease⟩          }
⟨latexrelease⟩      }
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_gset_protected:Npn \__hook_hook_gput_code_do:nnn #1 #2 #3
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_debug:n{\iow_term:x{****~ Add~ to~
⟨latexrelease⟩                      \__hook_if_usable:nF {#1} { undeclared~ }
⟨latexrelease⟩                      hook~ #1~ (#2)
⟨latexrelease⟩                      \on@line\space <-~ \tl_to_str:n{#3}} }
⟨latexrelease⟩    \str_if_eq:nnTF {#2} { top-level }
⟨latexrelease⟩      {
⟨latexrelease⟩        \str_if_eq:eeTF { top-level } { \__hook_currname_or_default: }
```

```
⟨latexrelease⟩          {
⟨latexrelease⟩            \__hook_init_structure:n {#1}
⟨latexrelease⟩            \__hook_tl_gput_right:cn { __hook_toplevel~#1 } {#3}
⟨latexrelease⟩          }
⟨latexrelease⟩          { \msg_error:nnn { hooks } { misused-top-level } {#1} }
⟨latexrelease⟩      }
⟨latexrelease⟩      {
⟨latexrelease⟩        \prop_get:cnNTF { g__hook_#1_code_prop } {#2} \l__hook_return_tl
⟨latexrelease⟩          {
⟨latexrelease⟩            \prop_gput:cno { g__hook_#1_code_prop } {#2}
⟨latexrelease⟩              { \l__hook_return_tl #3 }
⟨latexrelease⟩          }
⟨latexrelease⟩          { \prop_gput:cnn { g__hook_#1_code_prop } {#2} {#3} }
⟨latexrelease⟩      }
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_gset_protected:Npn \hook_gput_code_with_args:nnn #1#2#3 { }
⟨latexrelease⟩\EndIncludeInRelease
```

（\hook_gput_code:nnn 以及其它的定义结束。这些函数被记录在第17页。）

__hook_chk_args_allowed:nn

此宏检查是否可以向钩子 #1 添加带有引用的代码。仅在运行函数为 replacing_args 时才执行操作。如果钩子已声明且不带参数，将引发错误，然后设置 __hook_replacing_args_false:，以使调用它的宏可以正常添加代码。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_chk_args_allowed:nn}
⟨latexrelease⟩                 {Hooks~with~args}
\cs_new_protected:Npn \__hook_chk_args_allowed:nn #1 #2
  {
    \__hook_if_replacing_args:TF
      {
        \__hook_if_declared:nT {#1}
          { \tl_if_empty:cT { c__hook_#1_parameter_tl } { \use_ii:nn } }
        \use_none:n
          {
            \msg_error:nnnn { hooks } { without-args } {#1} {#2}
            \__hook_replacing_args_false:
          }
      }
      { }
  }
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_chk_args_allowed:nn}
⟨latexrelease⟩                 {Hooks~with~args}
```

```
⟨latexrelease⟩\cs_undefine:N \__hook_chk_args_allowed:nn
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_chk_args_allowed:nn* 定义结束。)

__hook_gput_undeclared_hook:nnn

经常会出现这样的情况，一个包 *A* 定义了一个钩子 foo，但是在加载 *A* 之前加载了向该钩子添加代码的包 *B*。在这种情况下，我们需要在其声明之前向钩子添加代码。隐式声明的钩子没有参数（原则上），所以在这里使用 \c_false_bool。

```
\cs_new_protected:Npn \__hook_gput_undeclared_hook:nnn #1 #2 #3
  {
    \__hook_init_structure:n {#1}
    \__hook_hook_gput_code_do:nnn {#1} {#2} {#3}
  }
```

(*__hook_gput_undeclared_hook:nnn* 定义结束。)

__hook_try_declaring_generic_hook:nnn
__hook_try_declaring_generic_next_hook:nn

这些入口级宏只是将参数传递给通用的 __hook_try_declaring_generic_hook:nNNnn，其中包含在采取某些操作时执行的正确函数。

包装器 __hook_try_declaring_generic_hook:nnn 接着推迟 \hook_gput_code:nnn，如果通用钩子已声明，否则推迟到 __hook_gput_undeclared_hook:nnn（在此之前已经测试了钩子是否存在，所以此时如果它不是通用的，它就不存在）。

包装器 __hook_try_declaring_generic_next_hook:nn 用于 next-execution 钩子，它做的事情类似：如果通用钩子已声明，就推迟代码到 \hook_gput_next_code:nn，否则推迟到 __hook_gput_next_do:nn。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_try_declaring_generic_hook:nnn}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \__hook_try_declaring_generic_hook:nnn #1
  {
    \__hook_try_declaring_generic_hook:wnTF #1 / / / \scan_stop: {#1}
      \__hook_gput_code:nnn
      \__hook_gput_undeclared_hook:nnn
        {#1}
  }
\cs_new_protected:Npn \__hook_try_declaring_generic_next_hook:nn #1
  {
    \__hook_try_declaring_generic_hook:wnTF #1 / / / \scan_stop: {#1}
      \__hook_gput_next_code:nn
      \__hook_gput_next_do:nn
        {#1}
  }
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2021/11/15}{\__hook_try_declaring_generic_hook:nnn}
```

```
⟨latexrelease⟩                {Standardise~generic~hook~names}
⟨latexrelease⟩\cs_gset_protected:Npn \__hook_try_declaring_generic_hook:nnn #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_try_declaring_generic_hook:wnTF #1 / / / \scan_stop: {#1}
⟨latexrelease⟩      \hook_gput_code:nnn
⟨latexrelease⟩      \__hook_gput_undeclared_hook:nnn
⟨latexrelease⟩        {#1}
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_gset_protected:Npn \__hook_try_declaring_generic_next_hook:nn #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_try_declaring_generic_hook:wnTF #1 / / / \scan_stop: {#1}
⟨latexrelease⟩      \hook_gput_next_code:nn
⟨latexrelease⟩      \__hook_gput_next_do:nn
⟨latexrelease⟩        {#1}
⟨latexrelease⟩  }
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_try_declaring_generic_hook:nnn}
⟨latexrelease⟩                {Standardise~generic~hook~names}
⟨latexrelease⟩\cs_new_protected:Npn \__hook_try_declaring_generic_hook:nnn #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_try_declaring_generic_hook:nNNnn {#1}
⟨latexrelease⟩      \hook_gput_code:nnn \__hook_gput_undeclared_hook:nnn
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_new_protected:Npn \__hook_try_declaring_generic_next_hook:nn #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_try_declaring_generic_hook:nNNnn {#1}
⟨latexrelease⟩      \hook_gput_next_code:nn \__hook_gput_next_do:nn
⟨latexrelease⟩  }
```

（*__hook_try_declaring_generic_hook:nnn* 和 *__hook_try_declaring_generic_next_hook:nn* 定义结束。）

__hook_try_declaring_generic_hook:nNNnn
hook_try_declaring_generic_hook_split:nNNnn

__hook_try_declaring_generic_hook:nNNnn 现在在第一个 / 处（如果有）分割钩子名称，并首先使用 __hook_if_file_hook:wTF 检查是否是特定于文件的钩子（它们需要一些规范化）。如果不是，则检查它是否是预定义集合中的一个通用名称。我们还分离出第二个组件，看看是否需要创建一个反向钩子。在任一情况下，该函数对于通用钩子返回 ⟨*true*⟩，对于其他情况返回 ⟨*false*⟩。

```
⟨latexrelease⟩\cs_new_protected:Npn \__hook_try_declaring_generic_hook:nNNnn #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_if_file_hook:wTF #1 / \s__hook_mark
⟨latexrelease⟩      {
⟨latexrelease⟩        \exp_args:Ne \__hook_try_declaring_generic_hook_split:nNNnn
⟨latexrelease⟩          { \exp_args:Ne \__hook_file_hook_normalize:n {#1} }
```

```
⟨latexrelease⟩       }
⟨latexrelease⟩       { \__hook_try_declaring_generic_hook_split:nNNnn {#1} }
⟨latexrelease⟩   }

⟨latexrelease⟩\cs_new_protected:Npn \__hook_try_declaring_generic_hook_split:nNNnn #1 #2 #3
⟨latexrelease⟩   {
⟨latexrelease⟩     \__hook_try_declaring_generic_hook:wnTF #1 / / / \scan_stop: {#1}
⟨latexrelease⟩       { #2 }
⟨latexrelease⟩       { #3 } {#1}
⟨latexrelease⟩   }
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_try_declaring_generic_hook:nNNnn* 和 *__hook_try_declaring_generic_hook_split:nNNnn* 定义结束。)

__hook_try_declaring_generic_hook:wn*TF*

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_try_declaring_generic_hook:wn}
⟨latexrelease⟩                  {Hooks~with~args}
\prg_new_protected_conditional:Npnn \__hook_try_declaring_generic_hook:wn
    #1 / #2 / #3 / #4 \scan_stop: #5 { TF }
  {
    \__hook_if_generic:nTF {#5}
      {
        \__hook_if_usable:nF {#5}
          {
```

如果这个钩子还不存在，我们会检查它是否是一个 cmd 钩子，如果是，我们会尝试修补这个命令，除了声明这个钩子。

对于某些命令，这是不可能的，这种情况下，__hook_patch_cmd_or_delay:Nnn（在 ltcmdhooks 中定义）将生成一个相应的错误消息。

```
            \str_if_eq:nnT {#1} { cmd }
              {
                \__hook_try_put_cmd_hook:n {#5}
                \__hook_make_usable:nn {#5} { 9 }
                \use_none:nnn
              }
```

即使无法真正使用这个钩子（错误消息将在其他地方生成），也要声明这个钩子。

在这里，我们使用 __hook_make_usable:nn，这样以后仍然可以使用 \hook_new:n。通用钩子（除了 cmd 钩子）不接受参数，所以将零作为第二个参数。

```
            \__hook_make_usable:nn {#5} { 0 }
          }
        \__hook_if_generic_reversed:nT {#5}
```

```
          { \tl_gset:cn { g__hook_#5_reversed_tl } { - } }
        \prg_return_true:
      }
      {
```

通用钩子都被命名为 ⟨*type*⟩/⟨*name*⟩/⟨*place*⟩，其中 ⟨*type*⟩ 和 ⟨*place*⟩ 是预定义的（\c__hook_generic_⟨*type*⟩/./⟨*place*⟩_tl），⟨*name*⟩ 是可变的部分。旧版本中有一些钩子的 ⟨*name*⟩ 在第三部分，所以下面的代码一段时间内支持这种语法，并会发出警告。

\exp_after:wN ... \exp:w 的技巧是为了删除 __hook_try_declaring_generic_hook:wnTF 插入的条件结构，从而允许访问后续的标记，这对于保持事务正常进行是必要的。

当淘汰周期结束时，下面的行应该全部替换为 \prg_return_false:。

```
        \__hook_if_deprecated_generic:nTF {#5}
          {
            \__hook_deprecated_generic_warn:n {#5}
            \exp_after:wN \__hook_declare_deprecated_generic:NNn
            \exp:w % \exp_end:
          }
          { \prg_return_false: }
      }
  }
```

__hook_deprecated_generic_warn:n 会针对给定的钩子发出弃用警告，并标记该钩子，以确保不会再次发出警告（可以发出多个警告，但每个钩子只发出一次）。

__hook_deprecated_generic_warn:Nn
__hook_deprecated_generic_warn:Nw

```
\cs_new_protected:Npn \__hook_deprecated_generic_warn:n #1
  { \__hook_deprecated_generic_warn:w #1 \s__hook_mark }
\cs_new_protected:Npn \__hook_deprecated_generic_warn:w
    #1 / #2 / #3 \s__hook_mark
  {
    \if_cs_exist:w __hook~#1/#2/#3 \cs_end: \else:
      \msg_warning:nnnnn { hooks } { generic-deprecated } {#1} {#2} {#3}
    \fi:
    \cs_gset_eq:cN { __hook~#1/#2/#3 } \scan_stop:
  }
```

现在用户已经被告知了这个废弃情况，我们通过交换 ⟨*name*⟩ 和 ⟨*place*⟩ 并将代码添加到正确的钩子中进行下一步处理。

__hook_do_deprecated_generic:Nn
__hook_do_deprecated_generic:Nw
__hook_declare_deprecated_generic:NNw
__hook_declare_deprecated_generic:NNw

```
\cs_new_protected:Npn \__hook_do_deprecated_generic:Nn #1 #2
  { \__hook_do_deprecated_generic:Nw #1 #2 \s__hook_mark }
\cs_new_protected:Npn \__hook_do_deprecated_generic:Nw #1
```

```
          #2 / #3 / #4 \s__hook_mark
  { #1 { #2 / #4 / #3 } }
\cs_new_protected:Npn \__hook_declare_deprecated_generic:NNn #1 #2 #3
  { \__hook_declare_deprecated_generic:NNw #1 #2 #3 \s__hook_mark }
\cs_new_protected:Npn \__hook_declare_deprecated_generic:NNw #1 #2
    #3 / #4 / #5 \s__hook_mark
  {
    \__hook_try_declaring_generic_hook:wnTF #3 / #5 / #4 / \scan_stop:
        { #3 / #5 / #4 }
      #1 #2 { #3 / #5 / #4 }
  }
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2021/11/15}{\__hook_try_declaring_generic_hook:wn}
⟨latexrelease⟩                  {Standardise~generic~hook~names}
⟨latexrelease⟩\prg_new_protected_conditional:Npnn \__hook_try_declaring_generic_hook:wn
⟨latexrelease⟩    #1 / #2 / #3 / #4 \scan_stop: #5 { TF }
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_if_generic:nTF {#5}
⟨latexrelease⟩      {
⟨latexrelease⟩        \__hook_if_usable:nF {#5}
⟨latexrelease⟩          {
⟨latexrelease⟩            \str_if_eq:nnT {#1} { cmd }
⟨latexrelease⟩              { \__hook_try_put_cmd_hook:n {#5} }
⟨latexrelease⟩            \__hook_make_usable:n {#5}
⟨latexrelease⟩          }
⟨latexrelease⟩        \__hook_if_generic_reversed:nT {#5}
⟨latexrelease⟩          { \tl_gset:cn { g__hook_#5_reversed_tl } { - } }
⟨latexrelease⟩        \prg_return_true:
⟨latexrelease⟩      }
⟨latexrelease⟩      {
⟨latexrelease⟩        \__hook_if_deprecated_generic:nTF {#5}
⟨latexrelease⟩          {
⟨latexrelease⟩            \__hook_deprecated_generic_warn:n {#5}
⟨latexrelease⟩            \exp_after:wN \__hook_declare_deprecated_generic:NNn
⟨latexrelease⟩            \exp:w % \exp_end:
⟨latexrelease⟩          }
⟨latexrelease⟩          { \prg_return_false: }
⟨latexrelease⟩      }
⟨latexrelease⟩  }
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2021/06/01}{\__hook_try_declaring_generic_hook:wn}
```

```
⟨latexrelease⟩                {Support~cmd~hooks}
⟨latexrelease⟩\prg_new_protected_conditional:Npnn \__hook_try_declaring_generic_hook:wn
⟨latexrelease⟩    #1 / #2 / #3 / #4 \scan_stop: #5 { TF }
⟨latexrelease⟩  {
⟨latexrelease⟩    \tl_if_empty:nTF {#2}
⟨latexrelease⟩      { \prg_return_false: }
⟨latexrelease⟩      {
⟨latexrelease⟩        \prop_if_in:NnTF \c__hook_generics_prop {#1}
⟨latexrelease⟩          {
⟨latexrelease⟩            \__hook_if_usable:nF {#5}
⟨latexrelease⟩              {
⟨latexrelease⟩                \str_if_eq:nnT {#1} { cmd }
⟨latexrelease⟩                  { \__hook_try_put_cmd_hook:n {#5} }
⟨latexrelease⟩                \__hook_make_usable:n {#5}
⟨latexrelease⟩              }
⟨latexrelease⟩            \prop_if_in:NnTF \c__hook_generics_reversed_ii_prop {#2}
⟨latexrelease⟩              { \tl_gset:cn { g__hook_#5_reversed_tl } { - } }
⟨latexrelease⟩              {
⟨latexrelease⟩                \prop_if_in:NnT \c__hook_generics_reversed_iii_prop {#3}
⟨latexrelease⟩                  { \tl_gset:cn { g__hook_#5_reversed_tl } { - } }
⟨latexrelease⟩              }
⟨latexrelease⟩            \prg_return_true:
⟨latexrelease⟩          }
⟨latexrelease⟩          { \prg_return_false: }
⟨latexrelease⟩      }
⟨latexrelease⟩  }
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_try_declaring_generic_hook:wn}
⟨latexrelease⟩                {Support~cmd~hooks}
⟨latexrelease⟩\prg_new_protected_conditional:Npnn \__hook_try_declaring_generic_hook:wn
⟨latexrelease⟩    #1 / #2 / #3 / #4 \scan_stop: #5 { TF }
⟨latexrelease⟩  {
⟨latexrelease⟩    \tl_if_empty:nTF {#2}
⟨latexrelease⟩      { \prg_return_false: }
⟨latexrelease⟩      {
⟨latexrelease⟩        \prop_if_in:NnTF \c__hook_generics_prop {#1}
⟨latexrelease⟩          {
⟨latexrelease⟩            \__hook_if_declared:nF {#5} { \hook_new:n {#5} }
⟨latexrelease⟩            \prop_if_in:NnTF \c__hook_generics_reversed_ii_prop {#2}
⟨latexrelease⟩              { \tl_gset:cn { g__hook_#5_reversed_tl } { - } }
⟨latexrelease⟩              {
⟨latexrelease⟩                \prop_if_in:NnT \c__hook_generics_reversed_iii_prop {#3}
```

```
⟨latexrelease⟩                  { \tl_gset:cn { g__hook_#5_reversed_tl } { - } }
⟨latexrelease⟩              }
⟨latexrelease⟩            \prg_return_true:
⟨latexrelease⟩          }
⟨latexrelease⟩          { \prg_return_false: }
⟨latexrelease⟩      }
⟨latexrelease⟩  }
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_try_declaring_generic_hook:wnTF* 以及其它的定义结束。)

__hook_if_file_hook_p:w
__hook_if_file_hook:wTF

__hook_if_file_hook:wTF 检查参数是否是有效的特定文件钩子(例如不是 file/before，而是 file/foo.tex/before)。如果是特定文件钩子，则执行 ⟨*true*⟩ 分支，否则执行 ⟨*false*⟩ 分支。

```
⟨latexrelease⟩\IncludeInRelease{2021/11/15}{\__hook_if_file_hook:w}
⟨latexrelease⟩                  {Standardise~generic~hook~names}
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_if_file_hook:w}
⟨latexrelease⟩                  {Standardise~generic~hook~names}
⟨latexrelease⟩\prg_new_conditional:Npnn \__hook_if_file_hook:w
⟨latexrelease⟩    #1 / #2 / #3 \s__hook_mark { TF }
⟨latexrelease⟩  {
⟨latexrelease⟩    \str_if_eq:nnTF {#1} { file }
⟨latexrelease⟩      {
⟨latexrelease⟩        \bool_lazy_or:nnTF
⟨latexrelease⟩            { \tl_if_empty_p:n {#3} }
⟨latexrelease⟩            { \str_if_eq_p:nn {#3} { / } }
⟨latexrelease⟩          { \prg_return_false: }
⟨latexrelease⟩          {
⟨latexrelease⟩            \prop_if_in:NnTF \c__hook_generics_file_prop {#2}
⟨latexrelease⟩              { \prg_return_true: }
⟨latexrelease⟩              { \prg_return_false: }
⟨latexrelease⟩          }
⟨latexrelease⟩      }
⟨latexrelease⟩      { \prg_return_false: }
⟨latexrelease⟩  }
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_if_file_hook:wTF* 定义结束。)

__hook_file_hook_normalize:n
__hook_strip_double_slash:n
__hook_strip_double_slash:w

```
⟨latexrelease⟩\IncludeInRelease{2021/11/15}{\__hook_file_hook_normalize:n}
⟨latexrelease⟩                  {Standardise~generic~hook~names}
```

```
⟨latexrelease⟩\EndIncludeInRelease
```

当找到一个特定于文件的钩子时，在声明之前会被 `\__hook_file_hook_normalize:n` 轻微规范化。当前的实现仅将两个连续的斜杠（//）替换为一个斜杠，以处理用户执行类似于 `\def\input@path{{./mypath/}}` 的简单情况，在这种情况下，一个钩子将会是 `\AddToHook{file/./mypath//file.tex/after}`。

```
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_file_hook_normalize:n}
⟨latexrelease⟩                  {Standardise~generic~hook~names}
⟨latexrelease⟩\cs_new:Npn \__hook_file_hook_normalize:n #1
⟨latexrelease⟩  { \__hook_strip_double_slash:n {#1} }
⟨latexrelease⟩\cs_new:Npn \__hook_strip_double_slash:n #1
⟨latexrelease⟩  { \__hook_strip_double_slash:w #1 // \s__hook_mark }
```

这个函数总是在使用 `\__hook_if_file_hook:wTF` 测试参数是否为文件钩子后调用的,所以我们可以假设它有三部分(要么是 `file/.../before`,要么是 `file/.../after`),因此我们使用 `#1/#2/#3 //` 而不是仅仅 `#1 //`，以防止如果文件名为空，则会丢失一个斜杠。

```
⟨latexrelease⟩\cs_new:Npn \__hook_strip_double_slash:w #1/#2/#3 // #4 \s__hook_mark
⟨latexrelease⟩  {
⟨latexrelease⟩    \tl_if_empty:nTF {#4}
⟨latexrelease⟩      { #1/#2/#3 }
⟨latexrelease⟩      { \__hook_strip_double_slash:w #1/#2/#3 / #4 \s__hook_mark }
⟨latexrelease⟩  }
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_file_hook_normalize:n*, *__hook_strip_double_slash:n*, 和 *__hook_strip_double_slash:w* 定义结束。)

\c__hook_generic_cmd/./before_tl
\c__hook_generic_cmd/./after_tl
\c__hook_generic_env/./before_tl
\c__hook_generic_env/./after_tl
\c__hook_generic_file/./before_tl
\c__hook_generic_file/./after_tl
\c__hook_generic_package/./before_tl
\c__hook_generic_package/./after_tl
\c__hook_generic_class/./before_tl
\c__hook_generic_class/./after_tl
\c__hook_generic_include/./before_tl
\c__hook_generic_include/./after_tl
\c__hook_generic_env/./begin_tl
\c__hook_generic_env/./end_tl
\c__hook_generic_include/./end_tl

定义可能的通用钩子的标记列表。我们不提供任何用户界面，因为这应该是静态的。

cmd 用于命令的通用钩子。

env 用于 `\begin` 和 `\end` 中的通用钩子。

file, package, class, include 在加载文件时使用的通用钩子。

```
⟨latexrelease⟩\IncludeInRelease{2021/11/15}{\c__hook_generics_prop}
⟨latexrelease⟩                  {Standardise~generic~hook~names}
\clist_map_inline:nn { cmd , env , file , package , class , include }
  {
    \tl_const:cn { c__hook_generic_#1/./before_tl } { + }
    \tl_const:cn { c__hook_generic_#1/./after_tl  } { - }
  }
\tl_const:cn { c__hook_generic_env/./begin_tl } { + }
\tl_const:cn { c__hook_generic_env/./end_tl   } { + }
```

```
\tl_const:cn { c__hook_generic_include/./end_tl } { - }
\tl_const:cn { c__hook_generic_include/./excluded_tl } { + }
```

废弃的通用钩子：

```
\clist_map_inline:nn { file , package , class , include }
  {
    \tl_const:cn { c__hook_deprecated_#1/./before_tl } { }
    \tl_const:cn { c__hook_deprecated_#1/./after_tl  } { }
  }
\tl_const:cn { c__hook_deprecated_include/./end_tl } { }
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\c__hook_generics_prop}
⟨latexrelease⟩                  {Standardise~generic~hook~names}
⟨latexrelease⟩\prop_const_from_keyval:Nn \c__hook_generics_prop
⟨latexrelease⟩     {cmd=,env=,file=,package=,class=,include=}
⟨latexrelease⟩\EndIncludeInRelease
```

(*\c__hook_generic_cmd/./before_tl 以及其它的定义结束。*)

\c__hook_generics_reversed_ii_prop
\c__hook_generics_reversed_iii_prop
\c__hook_generics_file_prop

以下通用钩子应该使用反向排序（ii 和 iii 的名称保留用于淘汰周期）：

```
⟨latexrelease⟩\IncludeInRelease{2021/11/15}{\c__hook_generics_reversed_ii_prop}
⟨latexrelease⟩                  {Standardise~generic~hook~names}
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\c__hook_generics_reversed_ii_prop}
⟨latexrelease⟩                  {Standardise~generic~hook~names}
⟨latexrelease⟩\prop_const_from_keyval:Nn \c__hook_generics_reversed_ii_prop {after=,end=}
⟨latexrelease⟩\prop_const_from_keyval:Nn \c__hook_generics_reversed_iii_prop {after=}
⟨latexrelease⟩\prop_const_from_keyval:Nn \c__hook_generics_file_prop {before=,after=}
⟨latexrelease⟩\EndIncludeInRelease
```

(*\c__hook_generics_reversed_ii_prop*，*\c__hook_generics_reversed_iii_prop*，和 *\c__hook_generics_file_prop* 定义结束。)

\c__hook_parameter_cmd/./before_tl
\c__hook_parameter_cmd/./after_tl

标记列表定义了特定类型的通用钩子的参数数量。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\c__hook_parameter_cmd/./before_tl}
⟨latexrelease⟩                  {Hooks~with~args}
```

cmd 钩子被声明为具有 9 个参数，因为它们具有可变数量的参数（取决于它们所附加的命令），所以我们在这里使用了最大值。

```
\tl_const:cn { c__hook_parameter_cmd/./before_tl } { #1#2#3#4#5#6#7#8#9 }
\tl_const:cn { c__hook_parameter_cmd/./after_tl }  { #1#2#3#4#5#6#7#8#9 }
```

```
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\c__hook_parameter_cmd/./before_tl}
⟨latexrelease⟩                    {Hooks~with~args}
⟨latexrelease⟩\EndIncludeInRelease
```

(\c__hook_parameter_cmd/./before_tl 和 \c__hook_parameter_cmd/./after_tl 定义结束。)

\hook_gremove_code:nn
__hook_gremove_code:nn

使用 \hook_gremove_code:nn{⟨hook⟩}{⟨label⟩} 可以移除存储在 ⟨label⟩ 下的 ⟨hook⟩ 的任何代码。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\hook_gremove_code:nn}
⟨latexrelease⟩                    {Hooks~with~args}
\cs_new_protected:Npn \hook_gremove_code:nn #1 #2
  { \__hook_normalize_hook_args:Nnn \__hook_gremove_code:nn {#1} {#2} }
\cs_new_protected:Npn \__hook_gremove_code:nn #1 #2
  {
```

首先检查钩子代码池是否存在。这里不使用 __hook_if_usable:nTF，因为应该可以在钩子定义之前移除代码（见第 2.1.7 节）。

```
    \__hook_if_structure_exist:nTF {#1}
      {
```

然后移除代码块并运行 __hook_update_hook_code:n,以便在我们在 \begin{document} 之后对执行标记列表进行反映更改。

如果要移除所有代码，则清除代码池 \g__hook_⟨hook⟩_code_prop，顶层代码 __hook_toplevel␣⟨hook⟩，以及下一个执行代码 __hook_next␣⟨hook⟩。

```
        \str_if_eq:nnTF {#2} {*}
          {
            \prop_gclear:c { g__hook_#1_code_prop }
            \__hook_toplevel_gset:nn {#1} { }
            \__hook_next_gset:nn {#1} { }
          }
          {
```

如果标签是 top-level，则清除标记列表，因为所有代码都在相同的标签下。

```
            \str_if_eq:nnTF {#2} { top-level }
              { \__hook_toplevel_gset:nn {#1} { } }
              {
                \prop_gpop:cnNF { g__hook_#1_code_prop } {#2} \l__hook_return_tl
                  { \msg_warning:nnnn { hooks } { cannot-remove } {#1} {#2} }
              }
          }
```

最后，如果钩子存在，更新代码。

```
      \__hook_if_usable:nT {#1}
        { \__hook_update_hook_code:n {#1} }
    }
```

如果这个钩子的代码池不存在，显示警告：

```
    {
      \__hook_if_deprecated_generic:nTF {#1}
        {
          \__hook_deprecated_generic_warn:n {#1}
          \__hook_do_deprecated_generic:Nn \__hook_gremove_code:nn {#1} {#2}
        }
        { \msg_warning:nnnn { hooks } { cannot-remove } {#1} {#2} }
    }
}
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\hook_gremove_code:nn}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_new_protected:Npn \__hook_gremove_code:nn #1 #2
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_if_structure_exist:nTF {#1}
⟨latexrelease⟩      {
⟨latexrelease⟩        \str_if_eq:nnTF {#2} {*}
⟨latexrelease⟩          {
⟨latexrelease⟩            \prop_gclear:c { g__hook_#1_code_prop }
⟨latexrelease⟩            \__hook_tl_gclear:c { __hook_toplevel~#1 }
⟨latexrelease⟩            \__hook_tl_gclear:c { __hook_next~#1 }
⟨latexrelease⟩          }
⟨latexrelease⟩          {
⟨latexrelease⟩            \str_if_eq:nnTF {#2} { top-level }
⟨latexrelease⟩              { \__hook_tl_gclear:c { __hook_toplevel~#1 } }
⟨latexrelease⟩              {
⟨latexrelease⟩                \prop_gpop:cnNF { g__hook_#1_code_prop } {#2} \l__hook_return_tl
⟨latexrelease⟩                  { \msg_warning:nnnn { hooks } { cannot-remove } {#1} {#2} }
⟨latexrelease⟩              }
⟨latexrelease⟩          }
⟨latexrelease⟩        \__hook_if_usable:nT {#1}
⟨latexrelease⟩          { \__hook_update_hook_code:n {#1} }
⟨latexrelease⟩      }
⟨latexrelease⟩      {
⟨latexrelease⟩        \__hook_if_deprecated_generic:nTF {#1}
⟨latexrelease⟩          {
```

```
⟨latexrelease⟩            \__hook_deprecated_generic_warn:n {#1}
⟨latexrelease⟩            \__hook_do_deprecated_generic:Nn \__hook_gremove_code:nn {#1} {#2}
⟨latexrelease⟩          }
⟨latexrelease⟩          { \msg_warning:nnnn { hooks } { cannot-remove } {#1} {#2} }
⟨latexrelease⟩      }
⟨latexrelease⟩  }
⟨latexrelease⟩\EndIncludeInRelease
```

(*\hook_gremove_code:nn* 和 *__hook_gremove_code:nn* 定义结束。这个函数被记录在第17页。)

__hook_cs_gput_right:nnn
__hook_cs_gput_right_fast:nnn
__hook_cs_gput_right_slow:nnn
__hook_code_gset_auxi:nnnn
__hook_code_gset_auxi:eeen

此宏用于将代码附加到 toplevel 和 next 标记列表中，并根据其参数数量正确处理它们，以及根据被添加的代码是否应将参数标记理解为参数，还是将其加倍以存储为参数标记。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_cs_gput_right:nnn}
⟨latexrelease⟩                    {Hooks~with~args}
```

检查当前钩子是否已声明且不带参数。在这种情况下，我们进行简化处理，并使用简单且更快速的方法，不需要进行哈希加倍。

```
\cs_new_protected:Npn \__hook_cs_gput_right:nnn #1 #2
  {
    \if:w T
        \__hook_if_declared:nF {#2} { F }
        \tl_if_empty:cF { c__hook_#2_parameter_tl } { F }
          T
      \exp_after:wN \__hook_cs_gput_right_fast:nnn
    \else:
      \exp_after:wN \__hook_cs_gput_right_slow:nnn
    \fi:
        {#1} {#2}
  }
\cs_new_protected:Npn \__hook_cs_gput_right_fast:nnn #1 #2 #3
  { \cs_gset:cpx { __hook#1~#2 } { \exp_not:v { __hook#1~#2 } \exp_not:n {#3} } }
\cs_new_protected:Npn \__hook_cs_gput_right_slow:nnn #1 #2 #3
  {
```

辅助命令 __hook_code_gset_auxi:eeen 最终只是在末尾执行赋值操作。它的第一个参数是宏的参数文本，在这里的选择取决于 \c__hook_⟨*hook*⟩_parameter_tl 是否存在，钩子是否已声明，以及它是否是通用钩子。

```
    \cs_if_exist:cF { __hook#1~#2 }
      { \__hook_code_gset_aux:nnn {#1} {#2} { } }
    \__hook_code_gset_auxi:eeen
      {
        \__hook_if_declared:nTF {#2}
```

```
          { \tl_use:c { c__hook_#2_parameter_tl } }
          {
            \__hook_if_generic:nTF {#2}
              { \__hook_generic_parameter:n {#2} }
              { \c__hook_nine_parameters_tl }
          }
      }
```

这里我们取宏中的现有代码，用它需要的参数进行展开，然后加倍哈希，以便可以重用代码。

PhO: 也许可以改进。通过快速检查 `\cs_replacement_spec`，可以优化添加到空 cs 的情况。

```
      {
        \exp_args:NNo \exp_args:No \__hook_double_hashes:n
          {
            \cs:w __hook#1~#2 \exp_last_unbraced:Ne \cs_end:
              { \__hook_braced_cs_parameter:n { __hook#1~#2 } }
          }
      }
```

现在是新代码：如果我们正在替换参数，则哈希保持不变，否则加倍。

```
      {
        \__hook_if_replacing_args:TF
          { \exp_not:n }
          { \__hook_double_hashes:n }
            {#3}
      }
```

最后，我们将定义带有以上所有内容的控制序列名称。

```
      { __hook#1~#2 }
  }
```

正如承诺的那样，这是执行定义的辅助函数。

```
\cs_new_protected:Npn \__hook_code_gset_auxi:nnnn #1 #2 #3 #4
  { \cs_gset:cpn {#4} #1 { #2 #3 } }
\cs_generate_variant:Nn \__hook_code_gset_auxi:nnnn { eeen }

⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_cs_gput_right:nnn}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_undefine:N \__hook_cs_gput_right:nnn
⟨latexrelease⟩\cs_undefine:N \__hook_cs_gput_right_fast:nnn
⟨latexrelease⟩\cs_undefine:N \__hook_cs_gput_right_slow:nnn
⟨latexrelease⟩\cs_undefine:N \__hook_code_gset_auxi:nnnn
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_cs_gput_right:nnn 以及其它的定义结束。*)

__hook_code_gset:nn
__hook_code_gset:ne
__hook_toplevel_gset:nn
__hook_next_gset:nn
__hook_code_gset_aux:nnn

这些宏定义了 __hook⟨*type*⟩␣⟨*hook*⟩（其中 ⟨*type*⟩ 是 _next、_toplevel 或为空），其中包含给定的代码以及存储在 \c__hook_⟨*hook*⟩_parameter_tl 中的参数（如果该参数不存在，则为空）。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_code_gset:nn}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \__hook_code_gset:nn
  { \__hook_code_gset_aux:nnn { } }
\cs_new_protected:Npn \__hook_toplevel_gset:nn
  { \__hook_code_gset_aux:nnn { _toplevel } }
\cs_new_protected:Npn \__hook_next_gset:nn
  { \__hook_code_gset_aux:nnn { _next } }
\cs_new_protected:Npn \__hook_code_gset_aux:nnn #1 #2 #3
  {
    \cs_gset:cpn { __hook#1~#2 \exp_last_unbraced:Ne }
      { \__hook_parameter:n {#2} }
      {#3}
  }
\cs_generate_variant:Nn \__hook_code_gset:nn { ne }

⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_code_gset:nn}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_undefine:N \__hook_code_gset:nn
⟨latexrelease⟩\cs_undefine:N \__hook_toplevel_gset:nn
⟨latexrelease⟩\cs_undefine:N \__hook_next_gset:nn
⟨latexrelease⟩\cs_undefine:N \__hook_code_gset_aux:nnn
⟨latexrelease⟩\EndIncludeInRelease
```

（__hook_code_gset:nn 以及其它的定义结束。）

__hook_normalise_cs_args:nn

此宏将 __hook⟨*type*⟩␣⟨*hook*⟩ 的参数标准化，以便在钩子声明后采用正确数量的参数。在此时，我们知道 \c__hook_⟨*hook*⟩_parameter_tl 存在，因此使用它来计算参数数量，并将其用作新（重新）定义的宏的 ⟨*parameter text*⟩。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_normalise_cs_args:nn}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \__hook_normalise_cs_args:nn #1 #2
  {
    \cs_if_exist:cT { __hook#1~#2 }
      {
        \__hook_code_gset_auxi:eeen
          { \tl_use:c { c__hook_#2_parameter_tl } }
          {
```

```
            \exp_args:NNo \exp_args:No \__hook_double_hashes:n
              {
                \cs:w __hook#1~#2 \exp_last_unbraced:Ne \cs_end:
                  { \__hook_braced_cs_parameter:n { __hook#1~#2 } }
              }
          }
          { }
          { __hook#1~#2 }
      }
  }
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_normalise_cs_args:nn}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_undefine:N \__hook_normalise_cs_args:nn
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_normalise_cs_args:nn* 定义结束。)

__hook_normalise_code_pool:n
__hook_set_normalise_fn:nn

这个有点巧妙。它接收一个钩子，在其代码池 (\g__hook_⟨*hook*⟩_code_prop) 上进行迭代，重新定义每个代码标签，以仅使用有效的参数。这在以下情况下使用：例如，添加的代码引用了参数 #1 和 #2，但是钩子只有 #1。在这种情况下，每个对 #2 的引用都被改为 ##2。这样做是因为否则，每当钩子发生某些更改（添加代码、设置规则等）时，TEX 将抛出低级错误，这可能没有充分的理由会变得非常重复乏味。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_normalise_code_pool:n}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \__hook_normalise_code_pool:n #1
  {
```

首先，使用钩子名称调用 __hook_set_normalise_fn:nn 来设置所有内容，然后我们将遍历钩子的代码池并应用上面的标准化。完成后，将临时属性列表复制回钩子的属性列表。

```
    \__hook_set_normalise_fn:nn {#1} { Offending~label:~'##1' }
    \prop_clear:N \l__hook_work_prop
    \prop_map_function:cN { g__hook_#1_code_prop } \__hook_normalise_fn:nn
    \prop_gset_eq:cN { g__hook_#1_code_prop } \l__hook_work_prop
  }
```

此函数的唯一目的是定义 __hook_normalise_fn:nn，然后对添加到钩子的代码进行更正。

```
\cs_new_protected:Npn \__hook_set_normalise_fn:nn #1 #2
  {
```

首先，我们定义两个辅助的记号列表。\l__hook_tmpb_tl 包含：

```
{\c__hook_hashes_tl 1}
{\c__hook_hashes_tl 2}
...
{\c__hook_hashes_tl 9}
```

```
    \cs_set:Npn \__hook_tmp:w ##1##2##3##4##5##6##7##8##9 { }
    \tl_set:Ne \l__hook_tmpb_tl
      { \__hook_braced_cs_parameter:n { __hook_tmp:w } }
    \group_begin:
      \__hook_tl_set:cn { c__hook_hash_tl } { \exp_not:N \c__hook_hashes_tl }
      \use:e
        {
    \group_end:
    \tl_set:Nn \exp_not:N \l__hook_tmpb_tl { \l__hook_tmpb_tl }
        }
```

\l__hook_tmpa_tl 包含：

```
{\c__hook_hash_tl 1}
{\c__hook_hash_tl 2}
...
{\c__hook_hash_tl <n>}
```

其中 $\langle n \rangle$ 是钩子声明的参数数量。

```
    \exp_last_unbraced:NNf
    \cs_set:Npn \__hook_tmp:w { \__hook_parameter:n {#1} } { }
    \tl_set:Ne \l__hook_tmpa_tl { \__hook_braced_cs_parameter:n { __hook_tmp:w } }
```

现在这个函数做的是有趣的部分。它被设计用于 \prop_map_function:NN，接收 ##1 中的标签名称和 ##2 中存储的代码。

```
    \cs_gset_protected:Npx \__hook_normalise_fn:nn ##1 ##2
      {
```

这里我们将定义两个辅助宏：第一个在检测到无效参数引用时引发错误。它利用 TEX 的低级错误“Illegal parameter number”，但是定义了一个奇怪命名的控制序列，以便错误可以出现漂亮的格式。例如，如果标签“badpkg”添加了一些代码，在钩子“foo”中引用了参数 #3，而钩子只有两个参数，那么错误将是：

```
! Illegal parameter number in definition of hook 'foo'.
(hooks)              Offending label: 'badpkg'.
<to be read again>
                   3
```

在此定义的点上，如果代码恰好引用了无效的参数，错误就会被引发。如果能够检测到此定义没有引发错误，下一步将是不必要的。我们将所有这些放在一个组中，以便这个奇怪的定义不会泄漏，并将 \tex_escapechar:D 设置为 −1，这样在出现错误时，这个 hack 将显示得更加漂亮。

```
      \group_begin:
        \int_set:Nn \tex_escapechar:D { -1 }
        \cs_set:cpn
            {
              hook~'#1'. ^^J
              (hooks) \prg_replicate:nn { 13 } { ~ }
              #2 % more message text
            }
            \exp_not:v { c__hook_#1_parameter_tl }
          {##2}
      \group_end:
```

下一个宏，名称要低调得多，始终接受九个参数，它只是将代码 ##2 在标签 ##1 下传输到临时属性列表中。前 ⟨n⟩ 个参数来自 \l__hook_tmpa_tl，而另外的 9 − ⟨n⟩ 个参数来自 \l__hook_tmpb_tl（后者包含的 # 个数是前者的两倍）。然后，__hook_double_hashes:n 用于将非参数哈希值加倍，并将 \c__hook_hash_tl 和 \c__hook_hashes_tl 展开为实际的参数标记。

```
      \cs_set:Npn \exp_not:N \__hook_tmp:w
          \exp_not:V \c__hook_nine_parameters_tl
        {
          \prop_put:Nne \exp_not:N \l__hook_work_prop
            {##1} { \exp_not:N \__hook_double_hashes:n {##2} }
        }
```

下一个宏，名称要低调得多，始终接受九个参数，它只是将代码 ##2 在标签 ##1 下传输到临时属性列表中。前 ⟨n⟩ 个参数来自 \l__hook_tmpa_tl，而另外的 9 − ⟨n⟩ 个参数来自 \l__hook_tmpb_tl（后者包含的 # 个数是前者的两倍）。然后，__hook_double_hashes:n 用于将非参数哈希值加倍，并将 \c__hook_hash_tl 和 \c__hook_hashes_tl 展开为实际的参数标记。

```
      \exp_not:N \__hook_tmp:w
        \exp_not:V \l__hook_tmpa_tl
        \exp_args:No \exp_not:o
          { \exp_after:wN \__hook_tmp:w \l__hook_tmpb_tl }
    }
  }
\cs_new_eq:NN \__hook_normalise_fn:nn ?
⟨latexrelease⟩\EndIncludeInRelease
```

```
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_normalise_code_pool:n}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_undefine:N \__hook_normalise_code_pool:n
⟨latexrelease⟩\EndIncludeInRelease
```

通过查看其替换文本来检查控制序列的展开是否为空。

__hook_cs_if_empty_p:c
__hook_cs_if_empty:cTF

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_cs_if_empty:c}
⟨latexrelease⟩                  {Hooks~with~args}
\prg_new_conditional:Npnn \__hook_cs_if_empty:c #1 { p, T, F, TF }
  {
    \if:w \scan_stop: \__hook_replacement_spec:c {#1} \scan_stop:
      \prg_return_true:
    \else:
      \prg_return_false:
    \fi:
  }
\cs_new:Npn \__hook_replacement_spec:c #1
  {
    \exp_args:Nc \token_if_macro:NT {#1}
      { \cs_replacement_spec:c {#1} }
  }
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_cs_if_empty:c}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_undefine:N \__hook_cs_if_empty:c
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_normalise_code_pool:n*, *__hook_set_normalise_fn:nn*, 和 *__hook_cs_if_empty:cTF* 定义结束。)

__hook_braced_cs_parameter:n
__hook_braced_hidden_loop:w
__hook_cs_parameter_count:N
__hook_cs_parameter_count:w
__hook_cs_end:w

查看控制序列的 ⟨*parameter text*⟩，并返回该宏的一系列“隐藏”的大括号参数。只要宏接受从零到九个简单参数的连续运行，这就能正常工作。这些参数被“隐藏”，因为参数标记是在 \c__hook_hash_tl 中返回而不是显式地返回，这样 __hook_double_hashes:n 就不会触及它们。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_braced_cs_parameter:n}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new:Npn \__hook_braced_cs_parameter:n #1
  {
    \exp_last_unbraced:Ne \__hook_braced_hidden_loop:w
      { \exp_args:Nc \__hook_cs_parameter_count:N {#1} } ? \s__hook_mark
  }
\cs_new:Npn \__hook_braced_hidden_loop:w #1
```

```
  {
    \if:w ? #1
      \__hook_use_i_delimit_by_s_mark:nw
    \fi:
    { \exp_not:N \c__hook_hash_tl #1 }
    \__hook_braced_hidden_loop:w
  }
\cs_new:Npn \__hook_cs_parameter_count:N #1
  {
    \exp_last_unbraced:Nf \__hook_cs_parameter_count:w
      { \token_if_macro:NT #1 { \cs_parameter_spec:N #1 } }
      ? \__hook_cs_end:w ? \__hook_cs_end:w ? \__hook_cs_end:w
      ? \__hook_cs_end:w ? \__hook_cs_end:w ? \__hook_cs_end:w
      ? \__hook_cs_end:w ? \__hook_cs_end:w ? \__hook_cs_end:w
      \s__hook_mark
  }
\cs_new:Npn \__hook_cs_parameter_count:w #1#2 #3#4 #5#6 #7#8
  { #2 #4 #6 #8 \__hook_cs_parameter_count:w }
\cs_new:Npn \__hook_cs_end:w #1 \s__hook_mark { }
⟨latexrelease⟩\EndIncludeInRelease
```

这个函数在回滚时不能被取消定义，因为它在该模块的末尾用于将钩子数据结构调整到先前版本。

```
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_braced_cs_parameter:n}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_braced_cs_parameter:n 以及其它的定义结束。*)

__hook_braced_parameter:n
__hook_braced_real_loop:w

在更简单的情况下使用此函数，不需要对井号进行特殊处理。这仅在 __hook_initialize_hook_code:n 内部使用，因此它假定 \c__hook_⟨*hook*⟩_parameter_tl 已定义，但在其他情况下也应该有效。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_braced_parameter:n}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new:Npn \__hook_braced_parameter:n #1
  {
    \if_case:w
      \int_eval:n
        { \exp_args:Nv \str_count:n { c__hook_#1_parameter_tl } / 3 }
      \exp_stop_f:
    \or: {##1}
    \or: {##1} {##2}
    \or: {##1} {##2} {##3}
```

```
    \or: {##1} {##2} {##3} {##4}
    \or: {##1} {##2} {##3} {##4} {##5}
    \or: {##1} {##2} {##3} {##4} {##5} {##6}
    \or: {##1} {##2} {##3} {##4} {##5} {##6} {##7}
    \or: {##1} {##2} {##3} {##4} {##5} {##6} {##7} {##8}
    \or: {##1} {##2} {##3} {##4} {##5} {##6} {##7} {##8} {##9}
    \else:
      \msg_expandable_error:nnn { latex2e } { should-not-happen }
        { Invalid~parameter~spec. }
    \fi:
  }
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_braced_parameter:n}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_undefine:N \__hook_braced_parameter:n
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_braced_parameter:n* 和 *__hook_braced_real_loop:w* 定义结束。)

`\__hook_parameter:n` 这只是一个快捷方式，用于对钩子的 ⟨*parameter text*⟩ 进行 `e`- 或 `f`-展开。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_parameter:n}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new:Npn \__hook_parameter:n #1
  {
    \cs:w c__hook_
    \tl_if_exist:cTF { c__hook_#1_parameter_tl }
      { #1_parameter } { empty }
    _tl \cs_end:
  }
\cs_new:Npn \__hook_generic_parameter:n #1
  { \__hook_generic_parameter:w #1 / / / \s__hook_mark }
\cs_new:Npn \__hook_generic_parameter:w #1 / #2 / #3 / #4 \s__hook_mark
  {
    \cs_if_exist_use:cF { c__hook_parameter_#1/./#3_tl }
      { \c__hook_empty_tl }
  }
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_parameter:n}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_undefine:N \__hook_parameter:n
⟨latexrelease⟩\cs_undefine:N \__hook_generic_parameter:n
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_parameter:n* 定义结束。)

4.7 为钩子代码设置规则

\g__hook_??_code_prop
__hook~??
\g__hook_??_reversed_tl
\c__hook_??_parameter_tl

最初，这些变量只是使用空的“label”名称（不是两个问号）。这有点不太幸运，因为此时 l3doc 在尝试排版文档时会抱怨命令名中间的 __。然而，使用“正常”的名称如 default 的缺点是它与真实的钩子名称并无真正的区别。我现在选择了??，虽然需要一些巧妙的技巧才能将其放入 csname，但由于这被大量使用，代码应该是快速的，所以在代码的后续部分并不使用 c 扩展。

__hook␣?? 没有被使用，但必须定义它来欺骗代码，让其认为?? 实际上是一个钩子。

```
\prop_new:c { g__hook_??_code_prop }
\prop_new:c { __hook~?? }
```

默认规则总是以正常顺序给出（从不以相反顺序）。如果这样的规则应用到了一个反转的钩子上，它会表现得好像规则被反转了（例如，after 变成了 before），因为这些规则首先被应用，然后顺序被反转。

```
\tl_new:c { g__hook_??_reversed_tl }
```

“默认”钩子的参数文本是空的。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\c__hook_??_parameter_tl}
⟨latexrelease⟩                  {Hooks~with~args}
\tl_const:cn { c__hook_??_parameter_tl } { }
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\c__hook_??_parameter_tl}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_undefine:c { c__hook_??_parameter_tl }
⟨latexrelease⟩\EndIncludeInRelease
```

(*\g__hook_??_code_prop* 以及其它的定义结束。)

\hook_gset_rule:nnnn
__hook_gset_rule:nnnn

通过 \hook_gset_rule:nnnn{⟨*hook*⟩}{⟨*label1*⟩}{⟨*relation*⟩}{⟨*label2*⟩} 为给定的⟨*hook*⟩定义了两个代码标签之间的关系。特殊的钩子?? 代表“任何”钩子，它设置了一个默认规则（如果两个钩子之间没有其他关系，则使用该规则）。

```
\cs_new_protected:Npn \hook_gset_rule:nnnn #1#2#3#4
  {
    \__hook_normalize_hook_rule_args:Nnnnn \__hook_gset_rule:nnnn
      {#1} {#2} {#3} {#4}
  }

⟨latexrelease⟩\IncludeInRelease{2022/06/01}{\__hook_gset_rule:nnnn}
⟨latexrelease⟩                  {Refuse~setting~rule~for~one-time~hooks}
```

```
\cs_new_protected:Npn \__hook_gset_rule:nnnn #1#2#3#4
  {
    \__hook_if_deprecated_generic:nT {#1}
      {
        \__hook_deprecated_generic_warn:n {#1}
        \__hook_do_deprecated_generic:Nn \__hook_gset_rule:nnnn {#1}
          {#2} {#3} {#4}
        \__hook_use_none_delimit_by_s_mark:w
      }
    \__hook_if_execute_immediately:nT {#1}
      {
        \msg_error:nnnnnn { hooks } { rule-too-late }
          {#1} {#2} {#3} {#4}
        \__hook_use_none_delimit_by_s_mark:w
      }
```

首先确保钩子的基本数据结构存在：

```
    \__hook_init_structure:n {#1}
```

然后清除两个标签之间的任何先前关系：

```
    \__hook_rule_gclear:nnn {#1} {#2} {#4}
```

接着调用处理给定规则的函数。如果规则无效，抛出错误。

```
    \cs_if_exist_use:cTF { __hook_rule_#3_gset:nnn }
      {
          {#1} {#2} {#4}
        \__hook_update_hook_code:n {#1}
      }
      {
        \msg_error:nnnnnn { hooks } { unknown-rule }
          {#1} {#2} {#3} {#4}
      }
    \s__hook_mark
  }

⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_gset_rule:nnnn}
⟨latexrelease⟩                  {Refuse~setting~rule~for~one-time~hooks}
⟨latexrelease⟩\cs_new_protected:Npn \__hook_gset_rule:nnnn #1#2#3#4
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_if_deprecated_generic:nT {#1}
⟨latexrelease⟩      {
⟨latexrelease⟩        \__hook_deprecated_generic_warn:n {#1}
⟨latexrelease⟩        \__hook_do_deprecated_generic:Nn \__hook_gset_rule:nnnn {#1}
```

```
⟨latexrelease⟩          {#2} {#3} {#4}
⟨latexrelease⟩        \exp_after:wN \use_none:nnnnnnnnn \use_none:n
⟨latexrelease⟩      }
⟨latexrelease⟩    \__hook_init_structure:n {#1}
⟨latexrelease⟩    \__hook_rule_gclear:nnn {#1} {#2} {#4}
⟨latexrelease⟩    \cs_if_exist_use:cTF { __hook_rule_#3_gset:nnn }
⟨latexrelease⟩      {
⟨latexrelease⟩          {#1} {#2} {#4}
⟨latexrelease⟩        \__hook_update_hook_code:n {#1}
⟨latexrelease⟩      }
⟨latexrelease⟩      {
⟨latexrelease⟩        \msg_error:nnnnnn { hooks } { unknown-rule }
⟨latexrelease⟩          {#1} {#2} {#3} {#4}
⟨latexrelease⟩      }
⟨latexrelease⟩  }
⟨latexrelease⟩\EndIncludeInRelease
```

(*\hook_gset_rule:nnnn* 和 *__hook_gset_rule:nnnn* 定义结束。这个函数被记录在第18页。)

__hook_rule_before_gset:nnn
__hook_rule_after_gset:nnn
__hook_rule_<_gset:nnn
__hook_rule_>_gset:nnn

然后我们添加新规则。我们需要在这里对规则进行规范化，以便后续更快地进行处理。对于一对标签 l_A 和 l_B，规则 $l_A > l_B$ 与 $l_B < l_A$ 是相同的，只是呈现方式不同。但通过将规则的形式规范化为单一表示，比如 $l_B < l_A$，可大大减少后续寻找规则所需的时间。

在这里，我们使用\(pdf)strcmp 来按字典顺序排列标签 l_A 和 l_B，以固定顺序进行规范化。然后每次这两个标签一起使用时，都会强制执行这个顺序。

这里我们使用 __hook_label_pair:nn {⟨*hook*⟩} {⟨l_A⟩} {⟨l_B⟩} 来构建一个带有固定顺序的字符串 $l_B|l_A$，并使用 __hook_label_ordered:nnTF 来根据是否排序应用正确的规则到这对标签。

```
\cs_new_protected:Npn \__hook_rule_before_gset:nnn #1#2#3
  {
    \__hook_tl_gset:cx { g__hook_#1_rule_ \__hook_label_pair:nn {#2} {#3} _tl }
      { \__hook_label_ordered:nnTF {#2} {#3} { < } { > } }
  }
\cs_new_eq:cN { __hook_rule_<_gset:nnn } \__hook_rule_before_gset:nnn

\cs_new_protected:Npn \__hook_rule_after_gset:nnn #1#2#3
  {
    \__hook_tl_gset:cx { g__hook_#1_rule_ \__hook_label_pair:nn {#3} {#2} _tl }
      { \__hook_label_ordered:nnTF {#3} {#2} { < } { > } }
  }
\cs_new_eq:cN { __hook_rule_>_gset:nnn } \__hook_rule_after_gset:nnn
```

(*__hook_rule_before_gset:nnn* 以及其它的定义结束。)

__hook_rule_voids_gset:nnn

这个规则如果标签 #2 在钩子 #1 中，就会从标签 #3 中移除（实际上是清除）代码。

```
\cs_new_protected:Npn \__hook_rule_voids_gset:nnn #1#2#3
  {
    \__hook_tl_gset:cx { g__hook_#1_rule_ \__hook_label_pair:nn {#2} {#3} _tl }
      { \__hook_label_ordered:nnTF {#2} {#3} { -> } { <- } }
  }
```

(*__hook_rule_voids_gset:nnn* 定义结束。)

__hook_rule_incompatible-error_gset:nnn
__hook_rule_incompatible-warning_gset:nnn

如果标签 #2 和 #3 在钩子 #1 中同时出现，这些关系将产生错误/警告。

```
\cs_new_protected:cpn { __hook_rule_incompatible-error_gset:nnn } #1#2#3
  { \__hook_tl_gset:cn { g__hook_#1_rule_ \__hook_label_pair:nn {#2} {#3} _tl }
                  { xE } }
\cs_new_protected:cpn { __hook_rule_incompatible-warning_gset:nnn } #1#2#3
  { \__hook_tl_gset:cn { g__hook_#1_rule_ \__hook_label_pair:nn {#2} {#3} _tl }
                  { xW } }
```

(*__hook_rule_incompatible-error_gset:nnn* 和 *__hook_rule_incompatible-warning_gset:nnn* 定义结束。)

__hook_rule_unrelated_gset:nnn
__hook_rule_gclear:nnn

撤销一个设置。__hook_rule_unrelated_gset:nnn 不需要执行任何操作，因为我们在设置任何规则之前使用了 __hook_rule_gclear:nnn。

```
\cs_new_protected:Npn \__hook_rule_unrelated_gset:nnn #1#2#3 { }
\cs_new_protected:Npn \__hook_rule_gclear:nnn #1#2#3
  { \cs_undefine:c { g__hook_#1_rule_ \__hook_label_pair:nn {#2} {#3} _tl } }
```

(*__hook_rule_unrelated_gset:nnn* 和 *__hook_rule_gclear:nnn* 定义结束。)

__hook_label_pair:nn

确保字典顺序更大的标签排在前面。

```
\cs_new:Npn \__hook_label_pair:nn #1#2
  {
    \if_case:w \__hook_str_compare:nn {#1} {#2} \exp_stop_f:
          #1 | #1 %  0
    \or:   #1 | #2 % +1
    \else: #2 | #1 % -1
    \fi:
  }
```

(*__hook_label_pair:nn* 定义结束。)

__hook_label_ordered_p:nn
__hook_label_ordered:nn*TF*

检查标签 #1 和 #2 是否按正确顺序排列（由 __hook_label_pair:nn 返回），如果是则返回 true，否则返回 false。

```
\prg_new_conditional:Npnn \__hook_label_ordered:nn #1#2 { TF }
  {
    \if_int_compare:w \__hook_str_compare:nn {#1} {#2} > 0 \exp_stop_f:
      \prg_return_true:
    \else:
      \prg_return_false:
    \fi:
  }
```

(*__hook_label_ordered:nnTF* 定义结束。)

__hook_if_label_case:nnnnn

为了避免在 __hook_initialize_single:NNn 中两次进行字符串比较（一次是用 \str_if_eq:nn，另一次是用 __hook_label_ordered:nn），我们使用一个三分支的宏来比较 #1 和 #2，如果它们相等，则展开为 \use_i:nnn，如果 #1 字典顺序更大，则展开为 \use_ii:nn，否则展开为 \use_iii:nn。

```
\cs_new:Npn \__hook_if_label_case:nnnnn #1#2
  {
    \cs:w use_
      \if_case:w \__hook_str_compare:nn {#1} {#2}
         i \or: ii \else: iii \fi: :nnn
    \cs_end:
  }
```

(*__hook_if_label_case:nnnnn* 定义结束。)

__hook_update_hook_code:n

在 \begin{document} 之前，这个命令不起作用；在文档正文中，它重新初始化了修改后的数据，

```
\cs_new_eq:NN \__hook_update_hook_code:n \use_none:n
```

(*__hook_update_hook_code:n* 定义结束。)

__hook_initialize_all:

初始化所有已知的钩子（在 \begin{document} 处），即，更新快速执行令牌列表，以正确顺序保存必要的代码。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_initialize_all:}
⟨latexrelease⟩                    {Hooks~with~args}
\cs_new_protected:Npn \__hook_initialize_all:
  {
```

首先，我们将 `\__hook_update_hook_code:n` 这个到目前为止什么都没做的命令改为现在初始化一个钩子。这样，以后对钩子的任何更新都会运行该代码，并更新执行令牌列表。

```
\cs_gset_eq:NN \__hook_update_hook_code:n \__hook_initialize_hook_code:n
```

现在我们循环遍历所有已定义的钩子并更新每一个。在这里，我们必须确定钩子是否有参数，这样辅助命令才知道如何处理井号（hash marks）。我们查看 `\c__hook_`⟨*hook*⟩`_parameter_tl`，如果有任何参数，就相应地设置 `replacing_args`。

```
\__hook_debug:n { \prop_gclear:N \g__hook_used_prop }
\seq_map_inline:Nn \g__hook_all_seq
  {
    \tl_if_empty:cTF { c__hook_##1_parameter_tl }
      { \__hook_replacing_args_false: }
      { \__hook_replacing_args_true: }
    \__hook_update_hook_code:n {##1}
    \__hook_replacing_args_reset:
  }
```

如果我们正在调试，我们逐个显示钩子的结果，对于所有具有数据的钩子。

```
\__hook_debug:n
  {
    \iow_term:x { ^^J All~initialized~(non-empty)~hooks: }
    \prop_map_inline:Nn \g__hook_used_prop
      {
        \iow_term:x
          { ^^J ~ ##1 ~ -> ~ \cs_replacement_spec:c { __hook~##1 } ~ }
      }
  }
```

在所有钩子被初始化之后，我们将“使用”更改为仅调用钩子代码，而不是初始化它（就像在导言部分做的那样）。

```
\__hook_post_initialization_defs:
}

⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_initialize_all:}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_gset_protected:Npn \__hook_initialize_all:
⟨latexrelease⟩  {
⟨latexrelease⟩    \cs_gset_eq:NN \__hook_update_hook_code:n \__hook_initialize_hook_code:n
⟨latexrelease⟩    \__hook_debug:n { \prop_gclear:N \g__hook_used_prop }
⟨latexrelease⟩    \seq_map_inline:Nn \g__hook_all_seq
⟨latexrelease⟩      { \__hook_update_hook_code:n {##1} }
```

```
⟨latexrelease⟩    \__hook_debug:n
⟨latexrelease⟩      {
⟨latexrelease⟩        \iow_term:x{^^JAll~ initialized~ (non-empty)~ hooks:}
⟨latexrelease⟩        \prop_map_inline:Nn \g__hook_used_prop
⟨latexrelease⟩          {
⟨latexrelease⟩            \iow_term:x
⟨latexrelease⟩              { ^^J ~ ##1 ~ -> ~ \cs_replacement_spec:c { __hook~##1 } ~ }
⟨latexrelease⟩          }
⟨latexrelease⟩      }
⟨latexrelease⟩    \cs_gset_eq:NN \hook_use:n \__hook_use_initialized:n
⟨latexrelease⟩    \cs_gset_eq:NN \__hook_preamble_hook:n \use_none:n
⟨latexrelease⟩  }
⟨@@=⟩
⟨latexrelease⟩\cs_gset_eq:NN \@expl@@@initialize@all@@
⟨latexrelease⟩                \__hook_initialize_all:
⟨@@=hook⟩
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_initialize_all:* 定义结束。)

__hook_initialize_hook_code:n

初始化或重新初始化快速执行的钩子代码。在导言部分，这是有选择地进行的，以防某个钩子被使用，而在 \begin{document} 处，这是对所有钩子进行的，然后仅在钩子代码更改后进行。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_initialize_hook_code:n}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \__hook_initialize_hook_code:n #1
  {
    \__hook_debug:n
      { \iow_term:x { ^^J Update~code~for~hook~'#1' \on@line :^^J } }
```

这部分负责排序和更新。首先我们检查旧版钩子宏是否存在，如果存在，我们将其添加到标签 legacy 下的钩子中。这可能会使钩子变得非空，因此我们必须在随后的测试之前执行此操作。

```
    \__hook_include_legacy_code_chunk:n {#1}
```

如果当前钩子没有任何代码块，那么甚至启动排序程序都没有意义，所以我们做了一个快速测试，在这种情况下，只需更新 __hook␣⟨*hook*⟩ 以包含 top-level 和 next 代码块即可。如果存在代码块，我们调用 __hook_initialize_single:NNn 并将其传递给它准备好的控制序列名称，因为这些名称在内部需要多次使用。这样做可以节省一些处理时间，如果提前进行这些操作。

```
    \__hook_if_usable:nT {#1}
      {
```

```
      \prop_if_empty:cTF { g__hook_#1_code_prop }
        {
          \__hook_code_gset:ne {#1}
            {
```

钩子可能带有参数，因此我们在 `_next` 和 `_toplevel` 宏之后添加了一系列大括号参数，以便将传递给钩子的参数转发给它们。

```
              \exp_not:c { __hook_toplevel~#1 } \__hook_braced_parameter:n {#1}
              \exp_not:c { __hook_next~#1 } \__hook_braced_parameter:n {#1}
            }
        }
        {
```

默认情况下，算法对代码块进行排序，然后将结果保存在一个 token list 中以进行快速执行；这是通过逐个添加代码块来完成的，使用 `\tl_gput_right:NV`。当我们对反向钩子的代码进行排序时，我们所要做的就是按相反的顺序将代码块添加到 token list 中。因此，在准备阶段，我们需要做的就是更改后续使用的两个定义。

```
          \__hook_if_reversed:nTF {#1}
            { \cs_set_eq:NN \__hook_tl_gput:Nn    \__hook_tl_gput_left:Nn
              \cs_set_eq:NN \__hook_clist_gput:NV \clist_gput_left:NV  }
            { \cs_set_eq:NN \__hook_tl_gput:Nn    \__hook_tl_gput_right:Nn
              \cs_set_eq:NN \__hook_clist_gput:NV \clist_gput_right:NV }
```

在排序过程中，某些关系（即 `voids`）需要在代码属性列表上进行破坏性操作，以删除不应出现在排序后的钩子 token list 中的代码，因此我们制作了代码属性列表的副本，可以在不更改主要列表的情况下安全地进行操作。

```
          \prop_set_eq:Nc \l__hook_work_prop { g__hook_#1_code_prop }
          \__hook_initialize_single:ccn
            { __hook~#1 } { g__hook_#1_labels_clist } {#1}
```

对于调试显示，我们希望跟踪实际添加了代码的那些钩子，因此我们在 plist 中记录下来。我们使用 plist 来确保仅记录每个钩子名称一次，即，我们只对键进行记录，值是任意的。

```
          \__hook_debug:n
            { \exp_args:NNx \prop_gput:Nnn \g__hook_used_prop {#1} { } }
        }
    }
  }
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_initialize_hook_code:n}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_gset_protected:Npn \__hook_initialize_hook_code:n #1
```

```
⟨latexrelease⟩ {
⟨latexrelease⟩   \__hook_debug:n
⟨latexrelease⟩     { \iow_term:x { ^^J Update~code~for~hook~'#1' \on@line :^^J } }
⟨latexrelease⟩   \__hook_include_legacy_code_chunk:n {#1}
⟨latexrelease⟩   \__hook_if_usable:nT {#1}
⟨latexrelease⟩     {
⟨latexrelease⟩       \prop_if_empty:cTF { g__hook_#1_code_prop }
⟨latexrelease⟩         {
⟨latexrelease⟩           \__hook_tl_gset:co { __hook~#1 }
⟨latexrelease⟩             {
⟨latexrelease⟩               \cs:w __hook_toplevel~#1 \exp_after:wN \cs_end:
⟨latexrelease⟩               \cs:w __hook_next~#1 \cs_end:
⟨latexrelease⟩             }
⟨latexrelease⟩         }
⟨latexrelease⟩         {
⟨latexrelease⟩           \__hook_if_reversed:nTF {#1}
⟨latexrelease⟩             { \cs_set_eq:NN \__hook_tl_gput:Nn    \__hook_tl_gput_left:Nn
⟨latexrelease⟩               \cs_set_eq:NN \__hook_clist_gput:NV \clist_gput_left:NV  }
⟨latexrelease⟩             { \cs_set_eq:NN \__hook_tl_gput:Nn    \__hook_tl_gput_right:Nn
⟨latexrelease⟩               \cs_set_eq:NN \__hook_clist_gput:NV \clist_gput_right:NV }
⟨latexrelease⟩           \prop_set_eq:Nc \l__hook_work_prop { g__hook_#1_code_prop }
⟨latexrelease⟩           \__hook_initialize_single:ccn
⟨latexrelease⟩             { __hook~#1 } { g__hook_#1_labels_clist } {#1}
⟨latexrelease⟩           \__hook_debug:n
⟨latexrelease⟩             { \exp_args:NNx \prop_gput:Nnn \g__hook_used_prop {#1} { } }
⟨latexrelease⟩         }
⟨latexrelease⟩     }
⟨latexrelease⟩ }
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_initialize_hook_code:n* 定义结束。)

__hook_tl_csname:n
__hook_seq_csname:n

更快的方法是传递一个单一的记号，并在需要时展开它，而不是传递一堆字符记号。

FMi: 自己的备注：验证

```
\cs_new:Npn \__hook_tl_csname:n #1 { l__hook_label_#1_tl }
\cs_new:Npn \__hook_seq_csname:n #1 { l__hook_label_#1_seq }
```

(*__hook_tl_csname:n* 和 *__hook_seq_csname:n* 定义结束。)

\l__hook_labels_seq
\l__hook_labels_int
\l__hook_front_tl
\l__hook_rear_tl
\l__hook_label_0_tl

对于排序，我基本上实现了 Knuth 在《计算机程序设计艺术》第 1 卷 263–266 页中给出的拓扑排序算法。对于这个算法，我们需要一些局部变量：

- 用于标记代码块的当前钩子中使用的标签列表：

```
\seq_new:N \l__hook_labels_seq
```

- 当前钩子中使用的标签数量。在 Knuth 的算法中，这被称为 N：

```
\int_new:N \l__hook_labels_int
```

- 要构建的排序代码列表由两个指针管理，一个指向队列的前端，一个指向后端。我们使用记号列表指针来模拟这一点。Knuth 将它们称为 F 和 R：

```
\tl_new:N \l__hook_front_tl
\tl_new:N \l__hook_rear_tl
```

- 队列起始处的数据保存在此记号列表中，它对应于 Don 称之为 `QLINK[0]` 的内容，但由于我们不是在内存中操作单个字，因此稍有不同的处理方式：

```
\tl_new:c { \__hook_tl_csname:n { 0 } }
```

(*\l__hook_labels_seq 以及其它的定义结束。*)

__hook_initialize_single:NNn
__hook_initialize_single:ccn

`\__hook_initialize_single:NNn` 实现了钩子代码块的排序，并将结果保存在令牌列表中以便快速执行(`#4`)。参数包括 ⟨*hook-code-plist*⟩、⟨*hook-code-tl*⟩、⟨*hook-top-level-code-tl*⟩、⟨*hook-next-code-tl*⟩、⟨*hook-ordered-labels-clist*⟩ 和 ⟨*hook-name*⟩（后者仅用于调试—⟨*hook-rule-plist*⟩通过 ⟨*hook-name*⟩ 访问）。

与 Don 的算法相比，额外的复杂性在于我们不使用简单的正整数，而是任意的字母数字标签。与往常一样，Don 的数据结构设计得可以省略很多测试，我尽可能地模仿了这一点。结果是一个我目前不测试的限制：标签不能等于数字 0！

FMi: 需要检查一下，以防万一... 也许

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_initialize_single:NNn}
⟨latexrelease⟩                    {Hooks~with~args}
\cs_new_protected:Npn \__hook_initialize_single:NNn #1#2#3
  {
```

步骤 T1：初始化数据结构 ...

```
    \seq_clear:N \l__hook_labels_seq
    \int_zero:N  \l__hook_labels_int
```

存储钩子的名称：

```
    \tl_set:Nn \l__hook_cur_hook_tl {#3}
```

我们循环遍历持有代码的属性列表，并记录所有在其中列出的标签。我们只关心那些标签的规则。在此过程中，我们计数（这给了我们 Knuth 算法中的 N）。为了确保名为 `front`、`rear`、`labels` 或 `return` 的标签不会与我们的代码发生冲突，我们在变量前加上前缀 `label_`。

```
\prop_map_inline:Nn \l__hook_work_prop
  {
    \int_incr:N \l__hook_labels_int
    \seq_put_right:Nn \l__hook_labels_seq {##1}
    \__hook_tl_set:cn { \__hook_tl_csname:n {##1} } { 0 }
    \seq_clear_new:c { \__hook_seq_csname:n {##1} }
  }
```

步骤 T2 和 T3：在这里，我们将相关的规则排序到数据结构中 …

这个循环构成了 `\l__hook_work_prop` 中标签的垂直和水平方向上的方阵。然而，由于规则 $l_A\langle rel\rangle l_B$ 与 $l_B\langle rel\rangle^{-1} l_A$ 是相同的，我们可以在矩阵的对角线处提前结束循环（即，当两个标签相等时），节省了大量时间。规则的设置方式（参见上面的 `\__hook_rule_before_gset:nnn` 的实现）确保我们在矩阵的被忽略侧没有规则，并且所有规则都被考虑到。规则在 `\__hook_apply_label_pair:nnn` 中被应用，该命令以正确排序的标签对作为参数。

```
\prop_map_inline:Nn \l__hook_work_prop
  {
    \prop_map_inline:Nn \l__hook_work_prop
      {
        \__hook_if_label_case:nnnnn {##1} {####1}
          { \prop_map_break: }
          { \__hook_apply_label_pair:nnn {##1} {####1} }
          { \__hook_apply_label_pair:nnn {####1} {##1} }
              {#3}
      }
  }
```

现在休息一下，看看已经设置好的数据结构：

```
\__hook_debug:n { \__hook_debug_label_data:N \l__hook_work_prop }
```

步骤 T4：

```
\tl_set:Nn \l__hook_rear_tl { 0 }
\tl_set:cn { \__hook_tl_csname:n { 0 } } { 0 }
\seq_map_inline:Nn \l__hook_labels_seq
  {
    \int_compare:nNnT { \cs:w \__hook_tl_csname:n {##1} \cs_end: } = 0
        {
          \tl_set:cn { \__hook_tl_csname:n { \l__hook_rear_tl } }{##1}
```

```
              \tl_set:Nn \l__hook_rear_tl {##1}
            }
      }
    \tl_set_eq:Nc \l__hook_front_tl { \__hook_tl_csname:n { 0 } }
    \__hook_tl_gclear:N #1
    \clist_gclear:N #2
```

整个循环在步骤 T5–T7 中组合起来：

```
    \bool_while_do:nn { ! \str_if_eq_p:Vn \l__hook_front_tl { 0 } }
      {
```

这部分是步骤 T5：

```
        \int_decr:N \l__hook_labels_int
        \prop_get:NVN \l__hook_work_prop \l__hook_front_tl \l__hook_return_tl
        \exp_args:NNV \__hook_tl_gput:Nn #1 \l__hook_return_tl
        \__hook_clist_gput:NV #2 \l__hook_front_tl
        \__hook_debug:n{ \iow_term:x{Handled~ code~ for~ \l__hook_front_tl} }
```

这是步骤 T6，不过我们不使用指针 P 来遍历后继项，而是使用映射函数的 `##1`。

```
        \seq_map_inline:cn { \__hook_seq_csname:n { \l__hook_front_tl } }
          {
            \tl_set:cx { \__hook_tl_csname:n {##1} }
                       { \int_eval:n
                           { \cs:w \__hook_tl_csname:n {##1} \cs_end: - 1 }
                       }
            \int_compare:nNnT
                { \cs:w \__hook_tl_csname:n {##1} \cs_end: } = 0
                {
                  \tl_set:cn { \__hook_tl_csname:n { \l__hook_rear_tl } } {##1}
                  \tl_set:Nn \l__hook_rear_tl            {##1}
                }
          }
```

这是步骤 T7：

```
        \tl_set_eq:Nc \l__hook_front_tl
                      { \__hook_tl_csname:n { \l__hook_front_tl } }
```

这是步骤 T8：如果我们还没有移动所有标签的代码（即，如果 `\l__hook_labels_int` 仍大于零），我们就有一个循环，我们的偏序关系无法被展开。

```
      }
    \int_compare:nNnF \l__hook_labels_int = 0
      {
        \iow_term:x{====================}
        \iow_term:x{Error:~ label~ rules~ are~ incompatible:}
```

在错误情况下，这不是真正需要的信息，但现在先这样吧 ...

FMi: 在某个雨天改善输出

```
      \__hook_debug_label_data:N \l__hook_work_prop
      \iow_term:x{====================}
    }
```

在我们将所有钩子代码添加到 #1 后，我们通过为 top-level (#2) 和单次执行 (#3) 添加额外的代码来完成它。这些通常应该是空的。top-level 代码是用 __hook_tl_gput:Nn 添加的，因为对于反转钩子，它可能会发生变化（那么 top-level 就是添加的第一个代码块）。next 代码始终添加在最后（在右侧）。钩子可能带有参数，所以我们在 _next 和 _toplevel 宏之后添加了一串大括号参数，以便将传递给钩子的参数转发给它们。

```
  \exp_args:NNe \__hook_tl_gput:Nn #1
    { \exp_not:c { __hook_toplevel~#3 } \__hook_braced_parameter:n {#3} }
  \__hook_tl_gput_right:Ne #1
    { \exp_not:c { __hook_next~#3 } \__hook_braced_parameter:n {#3} }
  \use:e
    {
      \cs_gset:cpn { __hook~#3 } \use:c { c__hook_#3_parameter_tl }
        { \exp_not:V #1 }
    }
}
\cs_generate_variant:Nn \__hook_initialize_single:NNn { cc }
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_initialize_single:NNn}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_new_protected:Npn \__hook_initialize_single:NNn #1#2#3
⟨latexrelease⟩  {
⟨latexrelease⟩    \seq_clear:N \l__hook_labels_seq
⟨latexrelease⟩    \int_zero:N  \l__hook_labels_int
⟨latexrelease⟩    \tl_set:Nn \l__hook_cur_hook_tl {#3}
⟨latexrelease⟩    \prop_map_inline:Nn \l__hook_work_prop
⟨latexrelease⟩      {
⟨latexrelease⟩        \int_incr:N \l__hook_labels_int
⟨latexrelease⟩        \seq_put_right:Nn \l__hook_labels_seq {##1}
⟨latexrelease⟩        \__hook_tl_set:cn { \__hook_tl_csname:n {##1} } { 0 }
⟨latexrelease⟩        \seq_clear_new:c { \__hook_seq_csname:n {##1} }
⟨latexrelease⟩      }
⟨latexrelease⟩    \prop_map_inline:Nn \l__hook_work_prop
```

```
⟨latexrelease⟩      {
⟨latexrelease⟩        \prop_map_inline:Nn \l__hook_work_prop
⟨latexrelease⟩          {
⟨latexrelease⟩            \__hook_if_label_case:nnnnn {##1} {####1}
⟨latexrelease⟩              { \prop_map_break: }
⟨latexrelease⟩              { \__hook_apply_label_pair:nnn {##1} {####1} }
⟨latexrelease⟩              { \__hook_apply_label_pair:nnn {####1} {##1} }
⟨latexrelease⟩                  {#3}
⟨latexrelease⟩          }
⟨latexrelease⟩      }
⟨latexrelease⟩    \__hook_debug:n { \__hook_debug_label_data:N \l__hook_work_prop }
⟨latexrelease⟩    \tl_set:Nn \l__hook_rear_tl { 0 }
⟨latexrelease⟩    \tl_set:cn { \__hook_tl_csname:n { 0 } } { 0 }
⟨latexrelease⟩    \seq_map_inline:Nn \l__hook_labels_seq
⟨latexrelease⟩      {
⟨latexrelease⟩        \int_compare:nNnT { \cs:w \__hook_tl_csname:n {##1} \cs_end: } = 0
⟨latexrelease⟩            {
⟨latexrelease⟩              \tl_set:cn { \__hook_tl_csname:n { \l__hook_rear_tl } }{##1}
⟨latexrelease⟩              \tl_set:Nn \l__hook_rear_tl {##1}
⟨latexrelease⟩            }
⟨latexrelease⟩      }
⟨latexrelease⟩    \tl_set_eq:Nc \l__hook_front_tl { \__hook_tl_csname:n { 0 } }
⟨latexrelease⟩    \__hook_tl_gclear:N #1
⟨latexrelease⟩    \clist_gclear:N #2
⟨latexrelease⟩    \bool_while_do:nn { ! \str_if_eq_p:Vn \l__hook_front_tl { 0 } }
⟨latexrelease⟩      {
⟨latexrelease⟩        \int_decr:N \l__hook_labels_int
⟨latexrelease⟩        \prop_get:NVN \l__hook_work_prop \l__hook_front_tl \l__hook_return_tl
⟨latexrelease⟩        \exp_args:NNV \__hook_tl_gput:Nn #1 \l__hook_return_tl
⟨latexrelease⟩        \__hook_clist_gput:NV #2 \l__hook_front_tl
⟨latexrelease⟩        \__hook_debug:n{ \iow_term:x{Handled~ code~ for~ \l__hook_front_tl} }
⟨latexrelease⟩        \seq_map_inline:cn { \__hook_seq_csname:n { \l__hook_front_tl } }
⟨latexrelease⟩          {
⟨latexrelease⟩            \tl_set:cx { \__hook_tl_csname:n {##1} }
⟨latexrelease⟩                      { \int_eval:n
⟨latexrelease⟩                          { \cs:w \__hook_tl_csname:n {##1} \cs_end: - 1 }
⟨latexrelease⟩                      }
⟨latexrelease⟩            \int_compare:nNnT
⟨latexrelease⟩                { \cs:w \__hook_tl_csname:n {##1} \cs_end: } = 0
⟨latexrelease⟩                {
⟨latexrelease⟩                  \tl_set:cn { \__hook_tl_csname:n { \l__hook_rear_tl } } {##1}
⟨latexrelease⟩                  \tl_set:Nn \l__hook_rear_tl            {##1}
```

```
⟨latexrelease⟩                }
⟨latexrelease⟩              }
⟨latexrelease⟩            \tl_set_eq:Nc \l__hook_front_tl
⟨latexrelease⟩                          { \__hook_tl_csname:n { \l__hook_front_tl } }
⟨latexrelease⟩          }
⟨latexrelease⟩        \int_compare:nNnF \l__hook_labels_int = 0
⟨latexrelease⟩          {
⟨latexrelease⟩            \iow_term:x{====================}
⟨latexrelease⟩            \iow_term:x{Error:~ label~ rules~ are~ incompatible:}
⟨latexrelease⟩            \__hook_debug_label_data:N \l__hook_work_prop
⟨latexrelease⟩            \iow_term:x{====================}
⟨latexrelease⟩          }
⟨latexrelease⟩        \exp_args:NNo \__hook_tl_gput:Nn #1 { \cs:w __hook_toplevel~#3 \cs_end: }
⟨latexrelease⟩        \__hook_tl_gput_right:No #1 { \cs:w __hook_next~#3 \cs_end: }
⟨latexrelease⟩    }
⟨latexrelease⟩\cs_generate_variant:Nn \__hook_tl_gput_right:Nn { No }
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_initialize_single:NNn* 定义结束。)

__hook_tl_gput:Nn
__hook_clist_gput:NV

这些要么添加在右侧（正常钩子），要么添加在左侧（反转钩子）。这是在 __hook_initialize_hook_code:n 中设置的，在其他地方它们的行为是未定义的。

```
\cs_new:Npn \__hook_tl_gput:Nn    { \ERROR }
\cs_new:Npn \__hook_clist_gput:NV { \ERROR }
```

(*__hook_tl_gput:Nn* 和 *__hook_clist_gput:NV* 定义结束。)

__hook_apply_label_pair:nnn
__hook_label_if_exist_apply:nnnF

这是在上面描述的循环中执行的步骤 T2 和 T3 的有效载荷。这个宏假设 #1 和 #2 是有序的，这意味着与对应对 #1 和 #2 的任何规则是 \g__hook_⟨*hook*⟩_rule_#1|#2_tl 相关，而不是 \g__hook_⟨*hook*⟩_rule_#2|#1_tl。这也节省了大量时间，因为我们只需要检查标签的顺序一次。

这里的参数是 ⟨*label1*⟩、⟨*label2*⟩、⟨*hook*⟩ 和 ⟨*hook-code-plist*⟩。我们将要应用下一个规则并将其输入到数据结构中。__hook_apply_label_pair:nnn 将只为 ⟨*hook*⟩ 调用 __hook_label_if_exist_apply:nnnF，如果找不到规则，还将尝试使用表示默认钩子规则的 ⟨*hook*⟩ 名称 ??。

__hook_label_if_exist_apply:nnnF 将检查给定钩子的规则是否存在，如果存在，则调用 __hook_apply_rule:nnn。

```
\cs_new_protected:Npn \__hook_apply_label_pair:nnn #1#2#3
  {
```

额外的复杂性：由于我们使用默认规则和本地钩子特定规则，我们首先必须检查是否存在本地规则，如果存在，则使用它。否则检查是否存在默认规则，然后使用它。

```
    \__hook_label_if_exist_apply:nnnF {#1} {#2} {#3}
      {
```

如果没有钩子特定的规则，我们检查是否存在默认规则，并在存在时使用它。

```
        \__hook_label_if_exist_apply:nnnF {#1} {#2} { ?? } { }
      }
  }
\cs_new_protected:Npn \__hook_label_if_exist_apply:nnnF #1#2#3
  {
    \if_cs_exist:w g__hook_ #3 _rule_ #1 | #2 _tl \cs_end:
```

具体要做什么取决于我们遇到的规则类型。如果是一个 `before` 规则，它将由算法处理，但其他类型需要以不同的方式进行管理。所有这些都在 `\__hook_apply_rule:nnnN` 中完成。

```
      \__hook_apply_rule:nnn {#1} {#2} {#3}
      \exp_after:wN \use_none:n
    \else:
      \use:nn
    \fi:
  }
```

(*__hook_apply_label_pair:nnn 和 __hook_label_if_exist_apply:nnnF 定义结束。*)

`\__hook_apply_rule:nnn` 这是在循环遍历矩阵时执行的步骤 T2 和 T3 的代码。这是步骤 T3 的一部分。我们将要应用下一个规则并将其输入到数据结构中。参数是 ⟨*label1*⟩、⟨*label2*⟩、⟨*hook-name*⟩ 和 ⟨*hook-code-plist*⟩。

```
\cs_new_protected:Npn \__hook_apply_rule:nnn #1#2#3
  {
    \cs:w __hook_apply_
      \cs:w g__hook_#3_reversed_tl \cs_end: rule_
        \cs:w g__hook_ #3 _rule_ #1 | #2 _tl \cs_end: :nnn \cs_end:
      {#1} {#2} {#3}
  }
```

(*__hook_apply_rule:nnn 定义结束。*)

`\__hook_apply_rule_<:nnn` `\__hook_apply_rule_>:nnn` 最常见的情况是 < 和 >，所以我们首先处理它们。它们在 TAOCP 中是关系 $\prec$ 和 $\succ$，并且它们决定了排序。

```
\cs_new_protected:cpn { __hook_apply_rule_<:nnn } #1#2#3
  {
    \__hook_debug:n { \__hook_msg_pair_found:nnn {#1} {#2} {#3} }
```

```
    \tl_set:cx { \__hook_tl_csname:n {#2} }
      { \int_eval:n{ \cs:w \__hook_tl_csname:n {#2} \cs_end: + 1 } }
    \seq_put_right:cn{ \__hook_seq_csname:n {#1} }{#2}
  }
\cs_new_protected:cpn { __hook_apply_rule_>:nnn } #1#2#3
  {
    \__hook_debug:n { \__hook_msg_pair_found:nnn {#1} {#2} {#3} }
    \tl_set:cx { \__hook_tl_csname:n {#1} }
      { \int_eval:n{ \cs:w \__hook_tl_csname:n {#1} \cs_end: + 1 } }
    \seq_put_right:cn{ \__hook_seq_csname:n {#2} }{#1}
  }
```

(*__hook_apply_rule_<:nnn* 和 *__hook_apply_rule_>:nnn* 定义结束。)

__hook_apply_rule_xE:nnn
__hook_apply_rule_xW:nnn

这些关系使得钩子中的两个标签不兼容。xE 如果在同一个钩子中发现这些标签，就会引发错误；xW 则会发出警告。

```
\cs_new_protected:cpn { __hook_apply_rule_xE:nnn } #1#2#3
  {
    \__hook_debug:n { \__hook_msg_pair_found:nnn {#1} {#2} {#3} }
    \msg_error:nnnnnn { hooks } { labels-incompatible }
      {#1} {#2} {#3} { 1 }
    \use:c { __hook_apply_rule_->:nnn } {#1} {#2} {#3}
    \use:c { __hook_apply_rule_<-:nnn } {#1} {#2} {#3}
  }
\cs_new_protected:cpn { __hook_apply_rule_xW:nnn } #1#2#3
  {
    \__hook_debug:n { \__hook_msg_pair_found:nnn {#1} {#2} {#3} }
    \msg_warning:nnnnnn { hooks } { labels-incompatible }
      {#1} {#2} {#3} { 0 }
  }
```

(*__hook_apply_rule_xE:nnn* 和 *__hook_apply_rule_xW:nnn* 定义结束。)

__hook_apply_rule_->:nnn
__hook_apply_rule_<-:nnn

如果我们看到 ->，我们必须丢弃标签 #3 的代码并继续。我们可以更好地做一点，丢弃该标签的所有内容，因为我们放置这样的空代码并不重要。但是，这会使算法变得更加复杂，而收益很少。[9] 因此，我们仍然不必要地尝试对其进行排序，具体取决于规则，这可能会导致一个本来可以解决的循环。如果这变成了一个真正的问题，我们可以改进代码。

[9]这也有一个优点，即排序的结果不会改变，否则（如果我们不小心）可能会导致（对于不相关的代码块）结果发生变化。

在这里，我们从 \l__hook_cur_hook_tl 中删除代码，而不是从 #3，因为后者可能是 ??，而默认钩子不存储任何代码。而是从 \l__hook_cur_hook_tl 中删除它，可以使默认规则 -> 和 <- 正常工作。

```
\cs_new_protected:cpn { __hook_apply_rule_->:nnn } #1#2#3
  {
    \__hook_debug:n
      {
        \__hook_msg_pair_found:nnn {#1} {#2} {#3}
        \iow_term:x{--->~ Drop~ '#2'~ code~ from~
          \iow_char:N \\ g__hook_ \l__hook_cur_hook_tl _code_prop ~
          because~ of~ '#1' }
      }
    \prop_put:Nnn \l__hook_work_prop {#2} { }
  }
\cs_new_protected:cpn { __hook_apply_rule_<-:nnn } #1#2#3
  {
    \__hook_debug:n
      {
        \__hook_msg_pair_found:nnn {#1} {#2} {#3}
        \iow_term:x{--->~ Drop~ '#1'~ code~ from~
          \iow_char:N \\ g__hook_ \l__hook_cur_hook_tl _code_prop ~
          because~ of~ '#2' }
      }
    \prop_put:Nnn \l__hook_work_prop {#1} { }
  }
```

(*__hook_apply_rule_->:nnn* 和 *__hook_apply_rule_<-:nnn* 定义结束。)

__hook_apply_-rule_<:nnn
__hook_apply_-rule_>:nnn
__hook_apply_-rule_<-:nnn
__hook_apply_-rule_->:nnn
__hook_apply_-rule_xW:nnn
__hook_apply_-rule_xE:nnn

反转规则。

```
\cs_new_eq:cc { __hook_apply_-rule_<:nnn } { __hook_apply_rule_>:nnn }
\cs_new_eq:cc { __hook_apply_-rule_>:nnn } { __hook_apply_rule_<:nnn }
\cs_new_eq:cc { __hook_apply_-rule_<-:nnn } { __hook_apply_rule_<-:nnn }
\cs_new_eq:cc { __hook_apply_-rule_->:nnn } { __hook_apply_rule_->:nnn }
\cs_new_eq:cc { __hook_apply_-rule_xE:nnn } { __hook_apply_rule_xE:nnn }
\cs_new_eq:cc { __hook_apply_-rule_xW:nnn } { __hook_apply_rule_xW:nnn }
```

(*__hook_apply_-rule_<:nnn* 以及其它的定义结束。)

__hook_msg_pair_found:nnn

一个宏，用来避免移动这么多标记。

```
\cs_new_protected:Npn \__hook_msg_pair_found:nnn #1#2#3
  {
    \iow_term:x{~ \str_if_eq:nnTF {#3} {??} {default} {~normal} ~
        rule~ \__hook_label_pair:nn {#1} {#2}:~
```

```
        \use:c { g__hook_#3_rule_ \__hook_label_pair:nn {#1} {#2} _tl } ~
        found}
  }
```

(*__hook_msg_pair_found:nnn* 定义结束。)

__hook_debug_label_data:N

```
\cs_new_protected:Npn \__hook_debug_label_data:N #1 {
  \iow_term:x{Code~ labels~ for~ sorting:}
  \iow_term:x{~ \seq_use:Nnnn\l__hook_labels_seq {~and~}{,~}{~and~} }
  \iow_term:x{^^J Data~ structure~ for~ label~ rules:}
  \prop_map_inline:Nn #1
      {
        \iow_term:x{~ ##1~ =~ \tl_use:c{ \__hook_tl_csname:n {##1} }~ ->~
          \seq_use:cnnn{ \__hook_seq_csname:n {##1} }{~->~}{~->~}{~->~}
        }
      }
  \iow_term:x{}
}
```

(*__hook_debug_label_data:N* 定义结束。)

\hook_show:n
\hook_log:n
__hook_log_line:x
__hook_log_line_indent:x
__hook_log:nN

这个宏根据其参数提供的钩子信息将内容写入到 .log 文件和终端（如果使用了 \show_hook:n）。在内部，两者共享相同的结构，除了在最后，\hook_show:n 会触发 TeX 的提示。

```
\cs_new_protected:Npn \hook_log:n #1
  {
    \cs_set_eq:NN \__hook_log_cmd:x \iow_log:x
    \__hook_normalize_hook_args:Nn \__hook_log:nN {#1} \tl_log:x
  }
\cs_new_protected:Npn \hook_show:n #1
  {
    \cs_set_eq:NN \__hook_log_cmd:x \iow_term:x
    \__hook_normalize_hook_args:Nn \__hook_log:nN {#1} \tl_show:x
  }
\cs_new_protected:Npn \__hook_log_line:x #1
  { \__hook_log_cmd:x { >~#1 } }
\cs_new_protected:Npn \__hook_log_line_indent:x #1
  { \__hook_log_cmd:x { >~\@spaces #1 } }

⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_log:nN}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \__hook_log:nN #1 #2
```

```
{
  \__hook_if_deprecated_generic:nT {#1}
    {
      \__hook_deprecated_generic_warn:n {#1}
      \__hook_do_deprecated_generic:Nn \__hook_log:nN {#1} #2
      \exp_after:wN \use_none:nnnnnnnnn \use_none:nnnnn
    }
  \__hook_preamble_hook:n {#1}
  \__hook_log_cmd:x
    {
      ^^J ->~The~
      \__hook_if_generic:nT {#1} { generic~ }
      hook~'#1'
      \__hook_if_disabled:nF {#1}
        {
          \exp_args:Nf \__hook_print_args:nn {#1}
            {
              \int_eval:n
                { \str_count:e { \__hook_parameter:n {#1} } / 3 }
            }
        }
      :
    }

  \__hook_if_usable:nF {#1}
    { \__hook_log_line:x { The~hook~is~not~declared. } }
  \__hook_if_disabled:nT {#1}
    { \__hook_log_line:x { The~hook~is~disabled. } }
  \hook_if_empty:nTF {#1}
    { #2 { The~hook~is~empty } }
    {
      \__hook_log_line:x { Code~chunks: }
      \prop_if_empty:cTF { g__hook_#1_code_prop }
        { \__hook_log_line_indent:x { --- } }
        {
          \prop_map_inline:cn { g__hook_#1_code_prop }
            {
              \exp_after:wN \cs_set:Npn \exp_after:wN \__hook_tmp:w
                \c__hook_nine_parameters_tl {##2}
              \__hook_log_line_indent:x
                { ##1~->~\cs_replacement_spec:N \__hook_tmp:w }
            }
        }
```

如果在 top-level 令牌列表中有代码，则打印它：

```
      \__hook_log_line:x
        {
          Document-level~(top-level)~code
          \__hook_if_usable:nT {#1}
            { ~(executed~\__hook_if_reversed:nTF {#1} {first} {last} ) } :
        }
      \__hook_log_line_indent:x
        {
          \__hook_cs_if_empty:cTF { __hook_toplevel~#1 }
            { --- }
            { -> ~ \cs_replacement_spec:c { __hook_toplevel~#1 } }
        }

      \__hook_log_line:x { Extra~code~for~next~invocation: }
      \__hook_log_line_indent:x
        {
          \__hook_cs_if_empty:cTF { __hook_next~#1 }
            { --- }
```

如果令牌列表不为空，我们想要显示它，但不包括第一个标记（用于清除自身的代码），所以我们调用一个辅助命令来去掉它们。

```
            {
              -> ~ \exp_last_unbraced:Nf \__hook_log_next_code:w
                { \cs_replacement_spec:c { __hook_next~#1 } }
            }
        }
```

循环遍历钩子中的规则，并对找到的每个规则进行打印。如果没有规则，则打印 ---。这里的布尔变量 \l__hook_tmpa_bool 表示钩子是否没有规则。

```
      \__hook_log_line:x { Rules: }
      \bool_set_true:N \l__hook_tmpa_bool
      \__hook_list_rules:nn {#1}
        {
          \bool_set_false:N \l__hook_tmpa_bool
          \__hook_log_line_indent:x
            {
              ##2~ with~
              \str_if_eq:nnT {##3} {??} { default~ }
              relation~ ##1
            }
        }
      \bool_if:NT \l__hook_tmpa_bool
```

```
          { \__hook_log_line_indent:x { --- } }
```

当钩子被声明（即，排序算法应用到该钩子上）且不为空时

```
        \bool_lazy_and:nnTF
          { \__hook_if_usable_p:n {#1} }
          { ! \hook_if_empty_p:n {#1} }
          {
            \__hook_log_line:x
              {
                Execution~order
                \bool_if:NTF \l__hook_tmpa_bool
                  { \__hook_if_reversed:nT {#1} { ~(after~reversal) } }
                  { ~(after~
                    \__hook_if_reversed:nT {#1} { reversal~and~ }
                    applying~rules)
                  } :
              }
            #2 % \tl_show:n
              {
                \@spaces
                \clist_if_empty:cTF { g__hook_#1_labels_clist }
                  { --- }
                  { \clist_use:cn { g__hook_#1_labels_clist } { ,~ } }
              }
          }
          {
            \__hook_log_line:x { Execution~order: }
            #2
              {
                \@spaces Not~set~because~the~hook~ \__hook_if_usable:nTF {#1}
                  { code~pool~is~empty }
                  { is~\__hook_if_disabled:nTF {#1} {disabled} {undeclared} }
              }
          }
      }
  }
⟨latexrelease⟩\EndIncludeInRelease
%
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_log:nN}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_new_protected:Npn \__hook_log:nN #1 #2
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_if_deprecated_generic:nT {#1}
```

```
⟨latexrelease⟩      {
⟨latexrelease⟩        \__hook_deprecated_generic_warn:n {#1}
⟨latexrelease⟩        \__hook_do_deprecated_generic:Nn \__hook_log:nN {#1} #2
⟨latexrelease⟩        \exp_after:wN \use_none:nnnnnnnnn \use_none:nnnnn
⟨latexrelease⟩      }
⟨latexrelease⟩    \__hook_preamble_hook:n {#1}
⟨latexrelease⟩    \__hook_log_cmd:x
⟨latexrelease⟩      { ^^J ->~The~ \__hook_if_generic:nT {#1} { generic~ } hook~'#1': }
⟨latexrelease⟩    \__hook_if_usable:nF {#1}
⟨latexrelease⟩      { \__hook_log_line:x { The~hook~is~not~declared. } }
⟨latexrelease⟩    \__hook_if_disabled:nT {#1}
⟨latexrelease⟩      { \__hook_log_line:x { The~hook~is~disabled. } }
⟨latexrelease⟩    \hook_if_empty:nTF {#1}
⟨latexrelease⟩      { #2 { The~hook~is~empty } }
⟨latexrelease⟩      {
⟨latexrelease⟩        \__hook_log_line:x { Code~chunks: }
⟨latexrelease⟩        \prop_if_empty:cTF { g__hook_#1_code_prop }
⟨latexrelease⟩          { \__hook_log_line_indent:x { --- } }
⟨latexrelease⟩          {
⟨latexrelease⟩            \prop_map_inline:cn { g__hook_#1_code_prop }
⟨latexrelease⟩              { \__hook_log_line_indent:x { ##1~->~\tl_to_str:n {##2} } }
⟨latexrelease⟩          }
⟨latexrelease⟩        \__hook_log_line:x
⟨latexrelease⟩          {
⟨latexrelease⟩            Document-level~(top-level)~code
⟨latexrelease⟩            \__hook_if_usable:nT {#1}
⟨latexrelease⟩              { ~(executed~\__hook_if_reversed:nTF {#1} {first} {last} ) } :
⟨latexrelease⟩          }
⟨latexrelease⟩        \__hook_log_line_indent:x
⟨latexrelease⟩          {
⟨latexrelease⟩            \tl_if_empty:cTF { __hook_toplevel~#1 }
⟨latexrelease⟩              { --- }
⟨latexrelease⟩              { -> ~ \exp_args:Nv \tl_to_str:n { __hook_toplevel~#1 } }
⟨latexrelease⟩          }
⟨latexrelease⟩        \__hook_log_line:x { Extra~code~for~next~invocation: }
⟨latexrelease⟩        \__hook_log_line_indent:x
⟨latexrelease⟩          {
⟨latexrelease⟩            \tl_if_empty:cTF { __hook_next~#1 }
⟨latexrelease⟩              { --- }
⟨latexrelease⟩              { ->~ \exp_args:Nv \__hook_log_next_code:n { __hook_next~#1 } }
⟨latexrelease⟩          }
⟨latexrelease⟩        \__hook_log_line:x { Rules: }
```

```
⟨latexrelease⟩        \bool_set_true:N \l__hook_tmpa_bool
⟨latexrelease⟩        \__hook_list_rules:nn {#1}
⟨latexrelease⟩          {
⟨latexrelease⟩            \bool_set_false:N \l__hook_tmpa_bool
⟨latexrelease⟩            \__hook_log_line_indent:x
⟨latexrelease⟩              {
⟨latexrelease⟩                ##2~ with~
⟨latexrelease⟩                \str_if_eq:nnT {##3} {??} { default~ }
⟨latexrelease⟩                relation~ ##1
⟨latexrelease⟩              }
⟨latexrelease⟩          }
⟨latexrelease⟩        \bool_if:NT \l__hook_tmpa_bool
⟨latexrelease⟩          { \__hook_log_line_indent:x { --- } }
⟨latexrelease⟩        \bool_lazy_and:nnTF
⟨latexrelease⟩            { \__hook_if_usable_p:n {#1} }
⟨latexrelease⟩            { ! \hook_if_empty_p:n {#1} }
⟨latexrelease⟩          {
⟨latexrelease⟩            \__hook_log_line:x
⟨latexrelease⟩              {
⟨latexrelease⟩                Execution~order
⟨latexrelease⟩                \bool_if:NTF \l__hook_tmpa_bool
⟨latexrelease⟩                  { \__hook_if_reversed:nT {#1} { ~(after~reversal) } }
⟨latexrelease⟩                  { ~(after~
⟨latexrelease⟩                    \__hook_if_reversed:nT {#1} { reversal~and~ }
⟨latexrelease⟩                    applying~rules)
⟨latexrelease⟩                  } :
⟨latexrelease⟩              }
⟨latexrelease⟩            #2 % \tl_show:n
⟨latexrelease⟩              {
⟨latexrelease⟩                \@spaces
⟨latexrelease⟩                \clist_if_empty:cTF { g__hook_#1_labels_clist }
⟨latexrelease⟩                  { --- }
⟨latexrelease⟩                  { \clist_use:cn { g__hook_#1_labels_clist } { ,~ } }
⟨latexrelease⟩              }
⟨latexrelease⟩          }
⟨latexrelease⟩          {
⟨latexrelease⟩            \__hook_log_line:x { Execution~order: }
⟨latexrelease⟩            #2
⟨latexrelease⟩              {
⟨latexrelease⟩                \@spaces Not~set~because~the~hook~ \__hook_if_usable:nTF {#1}
⟨latexrelease⟩                  { code~pool~is~empty }
⟨latexrelease⟩                  { is~\__hook_if_disabled:nTF {#1} {disabled} {undeclared} }
```

```
⟨latexrelease⟩                }
⟨latexrelease⟩            }
⟨latexrelease⟩        }
⟨latexrelease⟩    }
⟨latexrelease⟩\EndIncludeInRelease
```

To display the code for next invocation only (i.e., from `\AddToHookNext` we have to remove the string `\__hook_clear_next:n{`⟨*hook*⟩`}`, so the simplest is to use a macro delimited by a `}`$_1$2.

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_log_next_code:n}
⟨latexrelease⟩                    {Hooks~with~args}
\exp_last_unbraced:NNNNo
\cs_new:Npn \__hook_log_next_code:w #1 \c_right_brace_str { }
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_log_next_code:n}
⟨latexrelease⟩                    {Hooks~with~args}
⟨latexrelease⟩\cs_gset:Npn \__hook_log_next_code:n #1
⟨latexrelease⟩  { \exp_args:No \tl_to_str:n { \use_none:nn #1 } }
⟨latexrelease⟩\EndIncludeInRelease
```

__hook_log_next_code:n

美化打印钩子的参数数量。

```
\cs_new:Npn \__hook_print_args:nn #1 #2
  {
    \int_compare:nNnT {#2} > { 0 }
      {
        \__hook_if_declared:nT {#1} { \use_none:nnn }
        \__hook_if_cmd_hook:nT {#1}
          { \use_i:nnn { ~ (unknown ~ } }
        \use:n { ~ (#2 ~ }
        argument \int_compare:nNnT {#2} > { 1 } { s } )
      }
  }
```

__hook_print_args:n

(\hook_show:n 以及其它的定义结束。这些函数被记录在第18页。)

__hook_list_rules:nn
__hook_list_one_rule:nnn
__hook_list_if_rule_exists:nnnF

这个宏接受一个 ⟨*hook*⟩ 和一个 ⟨*inline function*⟩，并循环遍历 ⟨*hook*⟩ 中每一对 ⟨*labels*⟩，如果这对 ⟨*labels*⟩ 之间有关系，那么使用 `#1` = ⟨*relation*⟩，`#2` = ⟨*label*$_1$⟩|⟨*label*$_2$⟩，以及 `#3` = ⟨*hook*⟩（后者可能是对 `\__hook_list_rules:nn` 的参数 `#1`，或者如果它是默认规则，则是 `??`）来执行 ⟨*inline function*⟩。

```
\cs_new_protected:Npn \__hook_list_rules:nn #1 #2
  {
    \cs_set_protected:Npn \__hook_tmp:w ##1 ##2 ##3 {#2}
```

```
    \prop_map_inline:cn { g__hook_#1_code_prop }
      {
        \prop_map_inline:cn { g__hook_#1_code_prop }
          {
            \__hook_if_label_case:nnnnn {##1} {####1}
              { \prop_map_break: }
              { \__hook_list_one_rule:nnn {##1} {####1} }
              { \__hook_list_one_rule:nnn {####1} {##1} }
                  {#1}
          }
      }
  }
```

这两个与 `\__hook_apply_label_pair:nnn` 和 `\__hook_label_if_exist_apply:nnnF` 非常相似，但不是应用规则，而是将其传递给 ⟨*inline function*⟩。

```
\cs_new_protected:Npn \__hook_list_one_rule:nnn #1#2#3
  {
    \__hook_list_if_rule_exists:nnnF {#1} {#2} {#3}
      { \__hook_list_if_rule_exists:nnnF {#1} {#2} { ?? } { } }
  }
\cs_new_protected:Npn \__hook_list_if_rule_exists:nnnF #1#2#3
  {
    \if_cs_exist:w g__hook_ #3 _rule_ #1 | #2 _tl \cs_end:
      \exp_args:Nv \__hook_tmp:w
        { g__hook_ #3 _rule_ #1 | #2 _tl } { #1 | #2 } {#3}
      \exp_after:wN \use_none:nn
    \fi:
    \use:n
  }
```

(*__hook_list_rules:nn*, *__hook_list_one_rule:nnn*, 和 *__hook_list_if_rule_exists:nnnF* 定义结束。)

`\__hook_debug_print_rules:n` 用于调试的快捷方式，类似于 `\prop_show:N` 打印输出。

```
\cs_new_protected:Npn \__hook_debug_print_rules:n #1
  {
    \iow_term:n { The~hook~#1~contains~the~rules: }
    \cs_set_protected:Npn \__hook_tmp:w ##1
      {
        \__hook_list_rules:nn {#1}
          {
            \iow_term:x
              {
                > ##1 {####2} ##1 => ##1 {####1}
```

```
                \str_if_eq:nnT {####3} {??} { ~(default) }
              }
          }
      }
    \exp_args:No \__hook_tmp:w { \use:nn { ~ } { ~ } }
  }
```

(*__hook_debug_print_rules:n* 定义结束。)

4.8 指定下一次调用的代码

\hook_gput_next_code:nn

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\hook_gput_next_code:nn}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \hook_gput_next_code:nn #1 #2
  {
    \__hook_replacing_args_false:
    \__hook_normalize_hook_args:Nn \__hook_gput_next_code:nn {#1} {#2}
    \__hook_replacing_args_reset:
  }
\cs_new_protected:Npn \hook_gput_next_code_with_args:nn #1 #2
  {
    \__hook_replacing_args_true:
    \__hook_normalize_hook_args:Nn \__hook_gput_next_code:nn {#1} {#2}
    \__hook_replacing_args_reset:
  }
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\hook_gput_next_code:nn}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_gset_protected:Npn \hook_gput_next_code:nn #1
⟨latexrelease⟩  { \__hook_normalize_hook_args:Nn \__hook_gput_next_code:nn {#1} }
⟨latexrelease⟩\cs_gset_protected:Npn \hook_gput_next_code_with_args:nn #1 #2 { }
⟨latexrelease⟩\EndIncludeInRelease
```

(*\hook_gput_next_code:nn* 定义结束。这个函数被记录在第17页。)

__hook_gput_next_code:nn

```
\cs_new_protected:Npn \__hook_gput_next_code:nn #1 #2
  {
    \__hook_if_disabled:nTF {#1}
      { \msg_error:nnn { hooks } { hook-disabled } {#1} }
      {
        \__hook_if_structure_exist:nTF {#1}
```

```
          { \__hook_gput_next_do:nn }
          { \__hook_try_declaring_generic_next_hook:nn }
            {#1} {#2}
      }
  }
```

(*__hook_gput_next_code:nn* 定义结束。)

__hook_gput_next_do:nn 通过 __hook_chk_args_allowed:nn 进行合法性检查。然后检查是否“下一个代码”令牌列表为空：如果是，我们需要添加 \tl_gclear:c 来清除它，以便代码只持续一次使用。令牌列表被提前清除，以防嵌套使用导致遗失。在仅展开的环境中使用钩子时，使用 \tl_gclear:c 而不是 \tl_gclear:N，这样在 \tl_gclear:N 之前令牌列表不会展开：否则会导致无限循环。此外，如果主代码令牌列表为空，则需要更新钩子代码以添加下一个执行的令牌列表。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_gput_next_do:nn}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \__hook_gput_next_do:nn #1
  {
    \__hook_init_structure:n {#1}
    \__hook_chk_args_allowed:nn {#1} { AddToHookNext }
    \__hook_cs_if_empty:cT { __hook~#1 }
      { \__hook_update_hook_code:n {#1} }
    \__hook_cs_if_empty:cT { __hook_next~#1 }
      { \__hook_next_gset:nn {#1} { \__hook_clear_next:n {#1} } }
    \__hook_cs_gput_right:nnn { _next } {#1}
  }
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_gput_next_do:nn}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_gset_protected:Npn \__hook_gput_next_do:nn #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \exp_args:Nc \__hook_gput_next_do:Nnn
⟨latexrelease⟩      { __hook_next~#1 } {#1}
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_gset_protected:Npn \__hook_gput_next_do:Nnn #1 #2
⟨latexrelease⟩  {
⟨latexrelease⟩    \tl_if_empty:cT { __hook~#2 }
⟨latexrelease⟩      { \__hook_update_hook_code:n {#2} }
⟨latexrelease⟩    \tl_if_empty:NT #1
⟨latexrelease⟩      { \__hook_tl_gset:Nn #1 { \__hook_clear_next:n {#2} } }
⟨latexrelease⟩    \__hook_tl_gput_right:Nn #1
⟨latexrelease⟩  }
```

```
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_gput_next_do:nn* 定义结束。)

\hook_gclear_next_code:n 丢弃为下一次调用钩子设置的任何内容。

```
\cs_new_protected:Npn \hook_gclear_next_code:n #1
  { \__hook_normalize_hook_args:Nn \__hook_clear_next:n {#1} }
```

(*\hook_gclear_next_code:n* 定义结束。这个函数被记录在第17页。)

__hook_clear_next:n

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_clear_next:n}
⟨latexrelease⟩                    {Hooks~with~args}
\cs_new_protected:Npn \__hook_clear_next:n #1
  { \__hook_next_gset:nn {#1} { } }
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_clear_next:n}
⟨latexrelease⟩                    {Hooks~with~args}
⟨latexrelease⟩\cs_gset_protected:Npn \__hook_clear_next:n #1
⟨latexrelease⟩  { \cs_gset_eq:cN { __hook_next~#1 } \c_empty_tl }
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_clear_next:n* 定义结束。)

4.9 使用钩子

\hook_use:n
__hook_use_initialized:n
__hook_preamble_hook:n

此处定义的 \hook_use:n 用于导言部分，在那里钩子默认不被初始化。__hook_use_initialized:n 也被定义，它是用于文档内的非保护版本。它们的定义是相同的，除了 __hook_preamble_hook:n（在可展开版本中不会有影响，但这将是一个不必要的额外展开）。

__hook_use_initialized:n 在导言部分保持可展开的定义。__hook_preamble_hook:n 在导言部分初始化钩子，并在 \begin{document} 处被重新定义为 \use_none:n。

这两个版本在内部执行相同的操作：它们检查给定的钩子是否存在，如果存在则尽快使用它。

在 \begin{document} 处，所有钩子都被初始化，并且它们的任何更改都会引起更新，因此 \hook_use:n 可以被制作成可展开的版本。最好不要保护它，这样如果它不包含任何代码，就可以展开为空。对于通用钩子而言也很重要，我们不会生成 \relax 作为检查控制序列名的副作用。与 TEX 低级的 \csname ...\endcsname 构造不同，\tl_if_exist:c 小心避免这种情况。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\hook_use:n}
```

```
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \hook_use:n #1
  {
    \__hook_preamble_hook:n {#1}
    \__hook_use_initialized:n {#1}
  }
\cs_new:Npn \__hook_use_initialized:n #1
  {
    \if_cs_exist:w __hook~#1 \cs_end:
      \cs:w __hook~#1 \use_i:nn
    \fi:
    \use_none:n
    \cs_end:
  }
\cs_new_protected:Npn \__hook_preamble_hook:n #1
  {
    \if_cs_exist:w __hook~#1 \cs_end:
      \__hook_initialize_hook_code:n {#1}
    \fi:
  }
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2021/11/15}{\hook_use:n}
⟨latexrelease⟩                  {Standardise~generic~hook~names}
⟨latexrelease⟩\cs_new_protected:Npn \hook_use:n #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \tl_if_exist:cT { __hook~#1 }
⟨latexrelease⟩      {
⟨latexrelease⟩        \__hook_preamble_hook:n {#1}
⟨latexrelease⟩        \cs:w __hook~#1 \cs_end:
⟨latexrelease⟩      }
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_new:Npn \__hook_use_initialized:n #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \if_cs_exist:w __hook~#1 \cs_end:
⟨latexrelease⟩      \cs:w __hook~#1 \exp_after:wN \cs_end:
⟨latexrelease⟩    \fi:
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_new_protected:Npn \__hook_preamble_hook:n #1
⟨latexrelease⟩  { \__hook_initialize_hook_code:n {#1} }
⟨latexrelease⟩\cs_new:Npn \hook_use:nnw #1 { }
⟨latexrelease⟩\EndIncludeInRelease
```

```
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\hook_use:n}
⟨latexrelease⟩                  {Standardise~generic~hook~names}
⟨latexrelease⟩\cs_new_protected:Npn \hook_use:n #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \tl_if_exist:cTF { __hook~#1 }
⟨latexrelease⟩      {
⟨latexrelease⟩        \__hook_preamble_hook:n {#1}
⟨latexrelease⟩        \cs:w __hook~#1 \cs_end:
⟨latexrelease⟩      }
⟨latexrelease⟩      { \__hook_use:wn #1 / \s__hook_mark {#1} }
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_new:Npn \__hook_use_initialized:n #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \if_cs_exist:w __hook~#1 \cs_end:
⟨latexrelease⟩    \else:
⟨latexrelease⟩      \__hook_use_undefined:w
⟨latexrelease⟩    \fi:
⟨latexrelease⟩    \cs:w __hook~#1 \__hook_use_end:
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_new:Npn \__hook_use_undefined:w #1 #2 __hook~#3 \__hook_use_end:
⟨latexrelease⟩  {
⟨latexrelease⟩    #1 % fi
⟨latexrelease⟩    \__hook_use:wn #3 / \s__hook_mark {#3}
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_new_protected:Npn \__hook_preamble_hook:n #1
⟨latexrelease⟩  { \__hook_initialize_hook_code:n {#1} }
⟨latexrelease⟩\cs_new_eq:NN \__hook_use_end: \cs_end:
⟨latexrelease⟩\cs_new:Npn \hook_use:nnw #1 { }
⟨latexrelease⟩\EndIncludeInRelease
```

(*\hook_use:n*, *__hook_use_initialized:n*, 和 *__hook_preamble_hook:n* 定义结束。这个函数被记录在第16页。
)

\hook_use:nnw
__hook_use_initialized:nnw

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\hook_use:nnw}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \hook_use:nnw #1
  {
    \__hook_preamble_hook:n {#1}
    \__hook_use_initialized:nnw {#1}
  }
\cs_new:Npn \__hook_use_initialized:nnw #1 #2
  {
```

```
    \cs:w
      \if_cs_exist:w __hook~#1 \cs_end:
        __hook~#1
      \else:
        use_none: \prg_replicate:nn {#2} { n }
      \fi:
    \cs_end:
  }
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\hook_use:nnw}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_gset:Npn \hook_use:nnw #1 #2
⟨latexrelease⟩  { \use:c { use_none: \prg_replicate:nn {#2} { n } } }
⟨latexrelease⟩\EndIncludeInRelease
```

(*\hook_use:nnw* 和 *__hook_use_initialized:nnw* 定义结束。这个函数被记录在第16页。)

__hook_post_initialization_defs:

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_post_initialization_defs:}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \__hook_post_initialization_defs:
  {
    \cs_gset_eq:NN \hook_use:n \__hook_use_initialized:n
    \cs_gset_eq:NN \hook_use:nnw \__hook_use_initialized:nnw
    \cs_gset_eq:NN \__hook_preamble_hook:n \use_none:n
    \cs_gset_eq:NN \__hook_post_initialization_defs: \prg_do_nothing:
  }
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_post_initialization_defs:}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_undefine:N \__hook_post_initialization_defs:
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_post_initialization_defs:* 定义结束。)

__hook_use:wn
__hook_try_file_hook:n
__hook_if_usable_use:n

__hook_use:wn 进行了一个快速检查，以测试当前钩子是否是文件钩子：这些需要特殊处理。如果不是，钩子不存在。如果是，那么调用 __hook_try_file_hook:n，并使用 __hook_if_file_hook:wTF 检查当前钩子是否是文件特定钩子。如果不是，那么它就是一个通用的 file/ 钩子，如果存在则使用它。

如果它是一个文件特定的钩子，它将通过与声明期间相同的规范化流程，然后如果定义了就使用它。__hook_if_usable_use:n 检查钩子是否存在，如果存在，则调用 __hook_preamble_hook:n，然后使用该钩子。

```
⟨latexrelease⟩\IncludeInRelease{2021/11/15}{\__hook_use:wn}
⟨latexrelease⟩                  {Standardise~generic~hook~names}
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_use:wn}
⟨latexrelease⟩                  {Standardise~generic~hook~names}
⟨latexrelease⟩\cs_new:Npn \__hook_use:wn #1 / #2 \s__hook_mark #3
⟨latexrelease⟩  {
⟨latexrelease⟩    \str_if_eq:nnTF {#1} { file }
⟨latexrelease⟩      { \__hook_try_file_hook:n {#3} }
⟨latexrelease⟩      { } % Hook doesn't exist
⟨latexrelease⟩  }

⟨latexrelease⟩\cs_new_protected:Npn \__hook_try_file_hook:n #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_if_file_hook:wTF #1 / \s__hook_mark
⟨latexrelease⟩      {
⟨latexrelease⟩        \exp_args:Ne \__hook_if_usable_use:n
⟨latexrelease⟩          { \exp_args:Ne \__hook_file_hook_normalize:n {#1} }
⟨latexrelease⟩      }
⟨latexrelease⟩      { \__hook_if_usable_use:n {#1} } % file/ generic hook (e.g. file/before)
⟨latexrelease⟩  }

⟨latexrelease⟩\cs_new_protected:Npn \__hook_if_usable_use:n #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \tl_if_exist:cT { __hook~#1 }
⟨latexrelease⟩      {
⟨latexrelease⟩        \__hook_preamble_hook:n {#1}
⟨latexrelease⟩        \cs:w __hook~#1 \cs_end:
⟨latexrelease⟩      }
⟨latexrelease⟩  }
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_use:wn*, *__hook_try_file_hook:n*, 和 *__hook_if_usable_use:n* 定义结束。)

\hook_use_once:n
\hook_use_once:nnw

对于只能且应该仅使用一次的钩子，我们有一个特殊的使用命令，可以进一步禁止向其添加更多代码。这样做的效果是，任何进一步添加到钩子的代码会立即执行，而不是存储在钩子中。

代码需要一些技巧来防止钩子名称被空格修剪，因为 `\hook_use:n` 和 `\hook_use_once:n` 被记录为不修剪空格。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\hook_use_once:nnw}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \hook_use_once:n #1
  {
```

```
    \__hook_if_execute_immediately:nF {#1}
      { \__hook_normalize_hook_args:Nn \__hook_use_once:nn { \use:n {#1} } { 0 } }
  }
\cs_new_protected:Npn \hook_use_once:nnw #1 #2
  {
    \__hook_if_execute_immediately:nF {#1}
      { \__hook_normalize_hook_args:Nn \__hook_use_once:nn { \use:n {#1} } {#2} }
  }
⟨latexrelease⟩\EndIncludeInRelease
```

(*\hook_use_once:n* 和 *\hook_use_once:nnw* 定义结束。这些函数被记录在第16页。)

```
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\hook_use_once:nnw}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_gset_protected:Npn \hook_use_once:n #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_if_execute_immediately:nF {#1}
⟨latexrelease⟩      { \__hook_normalize_hook_args:Nn \__hook_use_once:n { \use:n {#1} } }
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_gset:Npn \hook_use_once:nnw #1 #2
⟨latexrelease⟩  { \use:c { use_none: \prg_replicate:nn {#2} { n } } }
⟨latexrelease⟩\EndIncludeInRelease
```

__hook_use_once:nn

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_use_once:nn}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \__hook_use_once:nn #1 #2
  {
    \__hook_preamble_hook:n {#1}
    \__hook_use_once_set:n {#1}
```

当一个钩子带有参数时，对 __hook_use_initialized:n 的调用应该是最后发生的事情，否则抓取的参数会出错。因此，在钩子之后进行清理，我们需要有些技巧，将清理代码偷偷放在钩子的末尾，同时还要与下一次执行代码放在一起。

```
    \__hook_replacing_args_false:
    \__hook_cs_gput_right:nnn { _next } {#1} { \__hook_use_once_clear:n {#1} }
    \__hook_replacing_args_reset:
    \__hook_if_usable:nTF {#1}
      { \__hook_use_initialized:n {#1} }
      {
        \int_compare:nNnT {#2} > { 0 }
          { \use:c { use_none: \prg_replicate:nn {#2} { n } } }
      }
```

```
  }
⟨latexrelease⟩\EndIncludeInRelease
%
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_use_once:nn}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_gset_protected:Npn \__hook_use_once:n #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_preamble_hook:n {#1}
⟨latexrelease⟩    \__hook_use_once_set:n {#1}
⟨latexrelease⟩    \__hook_use_initialized:n {#1}
⟨latexrelease⟩    \__hook_use_once_clear:n {#1}
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_undefine:N \__hook_use_once:nn
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_use_once:nn* 定义结束。)

__hook_use_once_set:n
__hook_use_once_clear:n

在实际钩子代码执行之前使用 __hook_use_once_set:n,这样钩子内部使用 \AddToHook 会导致代码立即执行。将 \g__hook_⟨*hook*⟩_reversed_tl 设置为 I 阻止进一步向钩子添加代码。然后 __hook_use_once_clear:n 清除钩子, 以便进一步调用 \hook_use:n 或 \hook_use_once:n 时会展开为空。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_use_once_clear:n}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new_protected:Npn \__hook_use_once_set:n #1
  { \__hook_tl_gset:cn { g__hook_#1_reversed_tl } { I } }
\cs_new_protected:Npn \__hook_use_once_clear:n #1
  {
    \__hook_code_gset:nn {#1} { }
    \__hook_next_gset:nn {#1} { }
    \__hook_toplevel_gset:nn {#1} { }
    \prop_gclear_new:c { g__hook_#1_code_prop }
  }
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_use_once_clear:n}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_new_protected:Npn \__hook_use_once_clear:n #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_tl_gclear:c { __hook~#1 }
⟨latexrelease⟩    \__hook_tl_gclear:c { __hook_next~#1 }
⟨latexrelease⟩    \__hook_tl_gclear:c { __hook_toplevel~#1 }
⟨latexrelease⟩    \prop_gclear_new:c { g__hook_#1_code_prop }
```

```
⟨latexrelease⟩ }
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_use_once_set:n* 和 *__hook_use_once_clear:n* 定义结束。)

__hook_if_execute_immediately_p:n
__hook_if_execute_immediately:nTF

要检查要添加的代码是否应立即执行（即，如果钩子是一次性钩子），我们检查 \g__hook_⟨*hook*⟩_reversed_tl 是否为 I。\if:w 的技巧在于允许 reversed 令牌列表为空。

```
\prg_new_conditional:Npnn \__hook_if_execute_immediately:n #1 { T, F, TF }
  {
    \exp_after:wN \__hook_use_none_delimit_by_s_mark:w
    \if:w I
        \if_cs_exist:w g__hook_#1_reversed_tl \cs_end:
          \cs:w g__hook_#1_reversed_tl \exp_after:wN \cs_end:
        \fi:
        X
      \s__hook_mark \prg_return_true:
    \else:
      \s__hook_mark \prg_return_false:
    \fi:
  }
```

(*__hook_if_execute_immediately:nTF* 定义结束。)

4.10 查询钩子

更简单的数据类型，比如记号列表，有三种可能的状态：它们可以存在且为空，存在且非空，也可能不存在，在这种情况下就不适用空（尽管在这种情况下 \tl_if_empty:N 返回假）。

钩子则稍微复杂一些：它们有几个其他状态，如 4.4.2 中讨论的那样。一个钩子可能存在，也可能不存在，无论哪种情况，它可能为空，也可能非空（即使一个不存在的钩子可能也是非空的）或者可能被禁用。

当一个钩子没有添加任何代码时，无论是添加到其永久代码池还是其“下一个”记号列表，它被称为空。钩子不需要声明为已添加代码到其代码池（可能发生的情况是包 A 定义了一个钩子 foo，但在加载包 B 之后，包 B 向该钩子添加了一些代码。在这种情况下，重要的是记住包 B 添加的代码，直到加载包 A 为止）。

其他所有状态只能通过内部测试进行查询，因为对于包代码来说，不同的状态是无关紧要的。

\hook_if_empty_p:n
\hook_if_empty:nTF

测试一个钩子是否为空（即，该钩子未添加任何代码）。一个 ⟨*hook*⟩ 为空意味着它的三个部分 \g__hook_⟨*hook*⟩_code_prop，__hook_toplevel␣⟨*hook*⟩ 和 __hook_next␣⟨*hook*⟩ 都为空。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\hook_if_empty:n}
⟨latexrelease⟩                  {Hooks~with~args}
\prg_new_conditional:Npnn \hook_if_empty:n #1 { p , T , F , TF }
  {
    \if:w
        T
        \prop_if_exist:cT { g__hook_#1_code_prop }
          { \prop_if_empty:cF { g__hook_#1_code_prop } { F } }
        \__hook_cs_if_empty:cF { __hook_toplevel~#1 } { F }
        \__hook_cs_if_empty:cF { __hook_next~#1 } { F }
        T
      \prg_return_true:
    \else:
      \prg_return_false:
    \fi:
  }
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\hook_if_empty:n}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\prg_new_conditional:Npnn \hook_if_empty:n #1 { p , T , F , TF }
⟨latexrelease⟩  {
⟨latexrelease⟩    \__hook_if_structure_exist:nTF {#1}
⟨latexrelease⟩      {
⟨latexrelease⟩        \bool_lazy_and:nnTF
⟨latexrelease⟩            { \prop_if_empty_p:c { g__hook_#1_code_prop } }
⟨latexrelease⟩            {
⟨latexrelease⟩              \bool_lazy_and_p:nn
⟨latexrelease⟩                { \tl_if_empty_p:c { __hook_toplevel~#1 } }
⟨latexrelease⟩                { \tl_if_empty_p:c { __hook_next~#1 } }
⟨latexrelease⟩            }
⟨latexrelease⟩          { \prg_return_true: }
⟨latexrelease⟩          { \prg_return_false: }
⟨latexrelease⟩      }
⟨latexrelease⟩      { \prg_return_true: }
⟨latexrelease⟩  }
⟨latexrelease⟩\EndIncludeInRelease
```

（*\hook_if_empty:nTF* 定义结束。这个函数被记录在第18页。）

__hook_if_usable_p:n
__hook_if_usable:nTF

如果存储钩子排序代码的记号列表 __hook␣⟨*hook*⟩ 存在，那么该钩子是可用的。不能在这里使用属性列表 \g__hook_⟨*hook*⟩_code_prop，因为通常需要向钩子添加代码，而不知道是否已经声明了这样的钩子，甚至是否将来会有声明（例如，如果定义它的包尚未加载）。

```
\prg_new_conditional:Npnn \__hook_if_usable:n #1 { p , T , F , TF }
  {
    \cs_if_exist:cTF { __hook~#1 }
      { \prg_return_true: }
      { \prg_return_false: }
  }
```

（*__hook_if_usable:nTF* 定义结束。）

__hook_if_structure_exist_p:n
__hook_if_structure_exist:nTF

内部检查钩子是否已经使用 __hook_init_structure:n 设置了其基本的内部结构。这意味着钩子已经以某种方式被使用过（向其添加了代码块或规则），但仍然没有使用 \hook_new:n 声明。

```
\prg_new_conditional:Npnn \__hook_if_structure_exist:n #1 { p , T , F , TF }
  {
    \prop_if_exist:cTF { g__hook_#1_code_prop }
      { \prg_return_true: }
      { \prg_return_false: }
  }
```

（*__hook_if_structure_exist:nTF* 定义结束。）

__hook_if_declared_p:n
__hook_if_declared:nTF

内部测试，检查钩子是否已经使用 \hook_new:n 或其变体正式声明。

```
\prg_new_conditional:Npnn \__hook_if_declared:n #1 { p, T, F, TF }
  {
    \tl_if_exist:cTF { g__hook_#1_declared_tl }
      { \prg_return_true: }
      { \prg_return_false: }
  }
```

（*__hook_if_declared:nTF* 定义结束。）

__hook_if_reversed_p:n
__hook_if_reversed:nTF

一个内部条件，用于检查钩子是否是反转的。

```
\prg_new_conditional:Npnn \__hook_if_reversed:n #1 { p , T , F , TF }
  {
    \exp_after:wN \__hook_use_none_delimit_by_s_mark:w
    \if:w - \cs:w g__hook_#1_reversed_tl \cs_end:
      \s__hook_mark \prg_return_true:
    \else:
```

```
      \s__hook_mark \prg_return_false:
    \fi:
  }
```

(*__hook_if_reversed:nTF* 定义结束。)

__hook_if_generic_p:n
__hook_if_generic:nTF
__hook_if_deprecated_generic_p:n
__hook_if_deprecated_generic:nTF

一个内部条件，用于检查名称是否属于通用钩子。已弃用版本需要检查 #3 是否为空，以避免在例如 file/before 上返回 true。

```
\prg_new_conditional:Npnn \__hook_if_generic:n #1 { T, TF }
  { \__hook_if_generic:w #1 / / / \s__hook_mark }
\cs_new:Npn \__hook_if_generic:w #1 / #2 / #3 / #4 \s__hook_mark
  {
    \cs_if_exist:cTF { c__hook_generic_#1/./#3_tl }
      { \prg_return_true: }
      { \prg_return_false: }
  }
\prg_new_conditional:Npnn \__hook_if_deprecated_generic:n #1 { T, TF }
  { \__hook_if_deprecated_generic:w #1 / / / \s__hook_mark }
\cs_new:Npn \__hook_if_deprecated_generic:w #1 / #2 / #3 / #4 \s__hook_mark
  {
    \cs_if_exist:cTF { c__hook_deprecated_#1/./#2_tl }
      {
        \tl_if_empty:nTF {#3}
          { \prg_return_false: }
          { \prg_return_true: }
      }
      { \prg_return_false: }
  }
```

(*__hook_if_generic:nTF* 和 *__hook_if_deprecated_generic:nTF* 定义结束。)

__hook_if_cmd_hook_p:n
__hook_if_cmd_hook:nTF
__hook_if_cmd_hook_p:w
__hook_if_cmd_hook:wTF

一个内部条件，用于检查给定的钩子是否是有效的通用 cmd 钩子。

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\__hook_if_cmd_hook:n}
⟨latexrelease⟩                    {Hooks~with~args}
\prg_new_conditional:Npnn \__hook_if_cmd_hook:n #1 { T }
  { \__hook_if_cmd_hook:w #1 / / / \s__hook_mark }
\cs_new:Npn \__hook_if_cmd_hook:w #1 / #2 / #3 / #4 \s__hook_mark
  {
    \if:w Y
          \str_if_eq:nnF {#1} { cmd } { N }
          \tl_if_exist:cF { c__hook_generic_#1/./#3_tl } { N }
          Y
      \prg_return_true:
```

```
    \else:
      \prg_return_false:
    \fi:
  }
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\__hook_if_cmd_hook:n}
⟨latexrelease⟩                    {Hooks~with~args}
⟨latexrelease⟩\cs_undefine:N \__hook_if_cmd_hook:nT
⟨latexrelease⟩\EndIncludeInRelease
```

(*__hook_if_cmd_hook:nTF* 和 *__hook_if_cmd_hook:wTF* 定义结束。)

__hook_if_generic_reversed_p:n
__hook_if_generic_reversed:n*TF*

一个内部条件，用于检查名称是否属于通用反转钩子。

```
\prg_new_conditional:Npnn \__hook_if_generic_reversed:n #1 { T }
  { \__hook_if_generic_reversed:w #1 / / / \scan_stop: }
\cs_new:Npn \__hook_if_generic_reversed:w #1 / #2 / #3 / #4 \scan_stop:
  {
    \if_charcode:w - \cs:w c__hook_generic_#1/./#3_tl \cs_end:
      \prg_return_true:
    \else:
      \prg_return_false:
    \fi:
  }
```

(*__hook_if_generic_reversed:nTF* 定义结束。)

__hook_if_replacing_args:TF
__hook_misused_if_replacing_args:nn
__hook_replacing_args_true:
__hook_replacing_args_false:
__hook_replacing_args_reset:
\g__hook_replacing_stack_seq

一个内部条件，用于检查添加到钩子的代码是否包含参数。

```
\seq_new:N \g__hook_replacing_stack_seq
\cs_new:Npn \__hook_misused_if_replacing_args:nn #1 #2
  {
    \msg_expandable_error:nnn { latex2e } { should-not-happen }
      { Misused~\__hook_if_replacing_args:. }
  }
\cs_new:Npn \__hook_if_replacing_args:TF
  { \__hook_misused_if_replacing_args:nn }
\cs_new_protected:Npn \__hook_replacing_args_true:
  {
    \seq_gpush:No \g__hook_replacing_stack_seq
      { \__hook_if_replacing_args:TF }
    \cs_set:Npn \__hook_if_replacing_args:TF { \use_i:nn }
  }
\cs_new_protected:Npn \__hook_replacing_args_false:
  {
```

```
    \seq_gpush:No \g__hook_replacing_stack_seq
      { \__hook_if_replacing_args:TF }
    \cs_set:Npn \__hook_if_replacing_args:TF { \use_ii:nn }
  }
\cs_new_protected:Npn \__hook_replacing_args_reset:
  {
    \seq_gpop:NN \g__hook_replacing_stack_seq \l__hook_return_tl
    \cs_gset_eq:NN \__hook_if_replacing_args:TF \l__hook_return_tl
  }
```

(*__hook_if_replacing_args:TF* 以及其它的定义结束。)

4.11 消息

钩子错误是 LaTeX 内核错误:

```
\prop_gput:Nnn \g_msg_module_type_prop { hooks } { LaTeX }
```

内核错误也是如此（这应该最终移动到其他地方)。

```
\prop_gput:Nnn \g_msg_module_type_prop { latex2e } { LaTeX }
\prop_gput:Nnn \g_msg_module_name_prop { latex2e } { kernel }

\msg_new:nnnn { hooks } { labels-incompatible }
  {
    Labels~'#1'~and~'#2'~are~incompatible
    \str_if_eq:nnF {#3} {??} { ~in~hook~'#3' } .~
    \int_compare:nNnTF {#4} = { 1 }
      { The~ code~ for~ both~ labels~ will~ be~ dropped. }
      { You~ may~ see~ errors~ later. }
  }
  { LaTeX~found~two~incompatible~labels~in~the~same~hook.~
    This~indicates~an~incompatibility~between~packages.  }

\msg_new:nnnn { hooks } { exists }
    { Hook~'#1'~ has~ already~ been~ declared. }
    { There~ already~ exists~ a~ hook~ declaration~ with~ this~
      name.\\
      Please~ use~ a~ different~ name~ for~ your~ hook.}

⟨latexrelease⟩\IncludeInRelease{2023/06/01}{too-many-args}
⟨latexrelease⟩                    {Hooks~with~args}

\msg_new:nnnn { hooks } { too-many-args }
  { Too~many~arguments~for~hook~'#1'. }
  {
```

```
    You~tried~to~declare~a~hook~with~#2~arguments,~but~a~
    hook~can~only~have~up~to~nine.~LaTeX~will~define~this~
    hook~with~nine~arguments.
  }

\msg_new:nnnn { hooks } { without-args }
  { Hook~'#1'~has~no~arguments. }
  {
    You~tried~to~use~\iow_char:N\\#2WithArguments~
    on~a~hook~that~takes~no~arguments.\\
    Check~the~usage~of~the~hook~or~use~\iow_char:N\\#2~instead.\\
    \\
    LaTeX~will~use~\iow_char:N\\#2.
  }

\msg_new:nnnn { hooks } { one-time-args }
  { You~can't~have~arguments~in~used~one-time~hook~'#1'. }
  {
    You~tried~to~use~\iow_char:N\\#2WithArguments~
    on~a~one-time~hook~that~has~already~been~used.~
    You~have~to~add~the~code~before~the~hook~is~used,~
    or~add~the~code~without~arguments~using~\iow_char:N\\#2~instead.\\
    \\
    LaTeX~will~use~\iow_char:N\\#2.
  }

⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{too-many-args}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\EndIncludeInRelease

\msg_new:nnnn { hooks } { hook-disabled }
  { Cannot~add~code~to~disabled~hook~'#1'. }
  {
    The~hook~'#1'~you~tried~to~add~code~to~was~previously~disabled~
    with~\iow_char:N\\hook_disable_generic:n~or~\iow_char:N\\DisableGenericHook,~so~
    it~cannot~have~code~added~to~it.
  }

\msg_new:nnn { hooks } { empty-label }
  {
    Empty~code~label~\msg_line_context:.~
    Using~'\__hook_currname_or_default:'~instead.
  }
```

```
\msg_new:nnn { hooks } { no-default-label }
  {
    Missing~(empty)~default~label~\msg_line_context:. \\
    This~command~was~ignored.
  }

\msg_new:nnnn { hooks } { unknown-rule }
  {
    Unknown~ relationship~ '#3'~
    between~ labels~ '#2'~ and~ '#4'~
    \str_if_eq:nnF {#1} {??} { ~in~hook~'#1' }. ~
    Perhaps~ a~ misspelling?
  }
  {
    The~ relation~ used~ not~ known~ to~ the~ system.~ Allowed~ values~ are~
    'before'~ or~ '<',~
    'after'~ or~ '>',~
    'incompatible-warning',~
    'incompatible-error',~
    'voids'~ or~
    'unrelated'.
  }

\msg_new:nnnn { hooks } { rule-too-late }
  {
    Sorting~rule~for~'#1'~hook~applied~too~late.\\
    Try~setting~this~rule~earlier.
  }
  {
    You~tried~to~set~the~ordering~of~hook~'#1'~using\\
    \ \ \iow_char:N\\DeclareHookRule{#1}{#2}{#3}{#4}\\
    but~hook~'#1'~was~already~used~as~a~one-time~hook,~
    thus~sorting~is\\
    no~longer~possible.~Declare~the~rule~
    before~the~hook~is~used.
  }

\msg_new:nnnn { hooks } { misused-top-level }
  {
    Illegal~use~of~\iow_char:N \\AddToHook{#1}[top-level]{...}.\\
    'top-level'~is~reserved~for~the~user's~document.
  }
  {
```

```
    The~'top-level'~label~is~meant~for~user~code~only,~and~should~only~
    be~used~(sparingly)~in~the~main~document.~Use~the~default~label~
    '\__hook_currname_or_default:'~for~this~\@cls@pkg,~or~another~
    suitable~label.
  }
\msg_new:nnn { hooks } { set-top-level }
  {
    You~cannot~change~the~default~label~#1~'top-level'.~Illegal \\
    \use:nn { ~ } { ~ } \iow_char:N \\#2{#3} \\
    \msg_line_context:.
  }
\msg_new:nnn { hooks } { extra-pop-label }
  {
    Extra~\iow_char:N \\PopDefaultHookLabel. \\
    This~command~will~be~ignored.
  }
\msg_new:nnn { hooks } { missing-pop-label }
  {
    Missing~\iow_char:N \\PopDefaultHookLabel. \\
    The~label~'#1'~was~pushed~but~never~popped.~Something~is~wrong.
  }
\msg_new:nnn { latex2e } { should-not-happen }
  {
    This~should~not~happen.~#1 \\
    Please~report~at~https://github.com/latex3/latex2e.
  }
\msg_new:nnn { hooks } { activate-disabled }
  {
    Cannot~ activate~ hook~ '#1'~ because~ it~ is~ disabled!
  }
\msg_new:nnn { hooks } { cannot-remove }
  {
    Cannot~remove~chunk~'#2'~from~hook~'#1'~because~
    \__hook_if_structure_exist:nTF {#1}
      { it~does~not~exist~in~that~hook. }
      { the~hook~does~not~exist. }
  }
\msg_new:nnn { hooks } { generic-deprecated }
```

```
{
  Generic~hook~'#1/#2/#3'~is~deprecated. \\
  Use~hook~'#1/#3/#2'~instead.
}
```

4.12 LaTeX 2ε 包接口命令

\NewHook
\NewReversedHook
\NewMirroredHookPair

声明新钩子 ...

```
\NewDocumentCommand \NewHook             { m }
  { \hook_new:n {#1} }
\NewDocumentCommand \NewReversedHook     { m }
  { \hook_new_reversed:n {#1} }
\NewDocumentCommand \NewMirroredHookPair { mm }
  { \hook_new_pair:nn {#1}{#2} }
```

(*\NewHook*, *\NewReversedHook*, 和 *\NewMirroredHookPair* 定义结束。这些函数被记录在第3页。)

\NewHookWithArguments
\NewReversedHookWithArguments
\NewMirroredHookPairWithArguments

声明带参数的新钩子...

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\NewHookWithArguments}
⟨latexrelease⟩                  {Hooks~with~args}
\NewDocumentCommand \NewHookWithArguments             { mm }
  { \hook_new_with_args:nn {#1} {#2} }
\NewDocumentCommand \NewReversedHookWithArguments     { mm }
  { \hook_new_reversed_with_args:nn {#1} {#2} }
\NewDocumentCommand \NewMirroredHookPairWithArguments { mmm }
  { \hook_new_pair_with_args:nnn {#1} {#2} {#3} }
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\NewHookWithArguments}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_new_protected:Npn \NewHookWithArguments #1 #2 { }
⟨latexrelease⟩\cs_new_protected:Npn \NewReversedHookWithArguments #1 #2 { }
⟨latexrelease⟩\cs_new_protected:Npn \NewMirroredHookPairWithArguments #1 #2 #3 { }
⟨latexrelease⟩\EndIncludeInRelease
```

(*\NewHookWithArguments*, *\NewReversedHookWithArguments*, 和 *\NewMirroredHookPairWithArguments* 定义结束。这些函数被记录在第4页。)

```
⟨latexrelease⟩\IncludeInRelease{2021/06/01}{\ActivateGenericHook}
⟨latexrelease⟩                  {Providing~hooks}
```

\ActivateGenericHook

提供新的钩子 ...

```
\NewDocumentCommand \ActivateGenericHook { m }
  { \hook_activate_generic:n {#1} }
```

(*\ActivateGenericHook* 定义结束。这个函数被记录在第4页。)

\DisableGenericHook 禁用通用钩子。

```
\NewDocumentCommand \DisableGenericHook { m }
  { \hook_disable_generic:n {#1} }
```

(*\DisableGenericHook* 定义结束。这个函数被记录在第4页。)

```
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\ActivateGenericHook}
⟨latexrelease⟩                  {Providing~hooks}
⟨latexrelease⟩\def \ActivateGenericHook #1 { }
⟨latexrelease⟩\def \DisableGenericHook #1 { }
⟨latexrelease⟩\EndIncludeInRelease
```

\AddToHook
\AddToHookWithArguments

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\AddToHookWithArguments}
⟨latexrelease⟩                  {Hooks~with~args}
\NewDocumentCommand \AddToHook { m o +m }
  { \hook_gput_code:nnn {#1} {#2} {#3} }
\NewDocumentCommand \AddToHookWithArguments { m o +m }
  { \hook_gput_code_with_args:nnn {#1} {#2} {#3} }
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\AddToHookWithArguments}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_new_protected:Npn \AddToHookWithArguments #1 #2 #3 { }
⟨latexrelease⟩\EndIncludeInRelease
```

(*\AddToHook* 和 *\AddToHookWithArguments* 定义结束。这些函数被记录在第6页。)

\AddToHookNext
\AddToHookNextWithArguments

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\AddToHookNextWithArguments}
⟨latexrelease⟩                  {Hooks~with~args}
\NewDocumentCommand \AddToHookNext { m +m }
  { \hook_gput_next_code:nn {#1} {#2} }
\NewDocumentCommand \AddToHookNextWithArguments { m +m }
  { \hook_gput_next_code_with_args:nn {#1} {#2} }
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\AddToHookNextWithArguments}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_new_protected:Npn \AddToHookNextWithArguments #1 #2 { }
⟨latexrelease⟩\EndIncludeInRelease
```

(*\AddToHookNext* 和 *\AddToHookNextWithArguments* 定义结束。这些函数被记录在第8页。)

\ClearHookNext

```
\NewDocumentCommand \ClearHookNext { m }
  { \hook_gclear_next_code:n {#1} }
```

(*\ClearHookNext* 定义结束。这个函数被记录在第8页。)

\RemoveFromHook

```
\NewDocumentCommand \RemoveFromHook { m o }
  { \hook_gremove_code:nn {#1} {#2} }
```

(*\RemoveFromHook* 定义结束。这个函数被记录在第7页。)

\SetDefaultHookLabel
\PushDefaultHookLabel
\PopDefaultHookLabel

现在定义一个包装器，用传入的参数替换堆栈顶部，并相应更新 \g__hook_hook_curr_name_tl。

```
\NewDocumentCommand \SetDefaultHookLabel { m }
  { \__hook_set_default_hook_label:n {#1} }
```

标签只会在 \@onefilewithoptions（\usepackage 和 \documentclass）中自动更新，但是一些包（例如 TikZ）定义了类似包的接口，比如 \usetikzlibrary，它们是 \input 的包装器，因此它们继承了当前生效的默认标签（通常是 top-level，但如果在另一个包中加载，则可能会更改）。为了为这些文件中的钩子提供类似包的行为，我们提供对默认标签栈的高级访问。

```
\NewDocumentCommand \PushDefaultHookLabel { m }
  { \__hook_curr_name_push:n {#1} }
\NewDocumentCommand \PopDefaultHookLabel { }
  { \__hook_curr_name_pop: }
```

当前标签栈保存除当前文件之外的所有文件的标签(更或多,类似于 \@currnamestack)，而当前标签记号列表 \g__hook_hook_curr_name_tl 保存当前文件的标签。但是 \@pushfilename 发生在 \@currname 设置之前，所以我们需要向前查找以获取标签的 \@currname。expl3 还需要 \@pushfilename 中的当前文件，因此这里我们滥用了 \@expl@push@filename@aux@@，来执行 __hook_curr_name_push:n。

```
\cs_gset_protected:Npn \@expl@push@filename@aux@@ #1#2#3
  {
    \__hook_curr_name_push:n {#3}
    \str_gset:Nx \g_file_curr_name_str {#3}
    #1 #2 {#3}
  }
```

(*\SetDefaultHookLabel*，*\PushDefaultHookLabel*，和 *\PopDefaultHookLabel* 定义结束。这些函数被记录在第11页。)

\UseHook
\UseOneTimeHook
\UseHookWithArguments
\UseOneTimeHookWithArguments

避免使用 xparse 的开销和保护，这里我们不需要（因为如果空，钩子应该毫无痕迹地消失）！

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}{\UseHookWithArguments}
⟨latexrelease⟩                  {Hooks~with~args}
\cs_new:Npn \UseHook         { \hook_use:n }
\cs_new:Npn \UseOneTimeHook { \hook_use_once:n }
\cs_new:Npn \UseHookWithArguments        { \hook_use:nnw }
\cs_new:Npn \UseOneTimeHookWithArguments { \hook_use_once:nnw }
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2020/10/01}{\UseHookWithArguments}
⟨latexrelease⟩                  {Hooks~with~args}
⟨latexrelease⟩\cs_new:Npn \UseHookWithArguments #1 #2 { }
⟨latexrelease⟩\cs_new:Npn \UseOneTimeHookWithArguments #1 #2 { }
⟨latexrelease⟩\EndIncludeInRelease
```

(*\UseHook* 以及其它的定义结束。这些函数被记录在第5页。)

\ShowHook
\LogHook

```
\cs_new_protected:Npn \ShowHook { \hook_show:n }
\cs_new_protected:Npn \LogHook { \hook_log:n }
```

(*\ShowHook* 和 *\LogHook* 定义结束。这些函数被记录在第14页。)

\DebugHooksOn
\DebugHooksOff

```
\cs_new_protected:Npn \DebugHooksOn  { \hook_debug_on:  }
\cs_new_protected:Npn \DebugHooksOff { \hook_debug_off: }
```

(*\DebugHooksOn* 和 *\DebugHooksOff* 定义结束。这些函数被记录在第15页。)

\DeclareHookRule

```
\NewDocumentCommand \DeclareHookRule { m m m m }
                    { \hook_gset_rule:nnnn {#1}{#2}{#3}{#4} }
```

(*\DeclareHookRule* 定义结束。这个函数被记录在第12页。)

\DeclareDefaultHookRule

这个声明仅在 \begin{document} 之前受支持。

```
\NewDocumentCommand \DeclareDefaultHookRule { m m m }
                    { \hook_gset_rule:nnnn {??}{#1}{#2}{#3} }
\@onlypreamble\DeclareDefaultHookRule
```

(*\DeclareDefaultHookRule* 定义结束。这个函数被记录在第12页。)

\ClearHookRule

一个特殊的设置规则，用于移除现有的关系。基本上是 @@_rule_gclear:nnn 加上修复调试用的属性列表。

FMi: 也许需要一个 *L3* 接口，或者也许应该删除？

```
\NewDocumentCommand \ClearHookRule { m m m }
{ \hook_gset_rule:nnnn {#1}{#2}{unrelated}{#3} }
```

(*\ClearHookRule* 定义结束。这个函数被记录在第12页。)

\IfHookEmptyTF 这里我们避免了 xparse 的开销，因为 \IfHookEmptyTF 被用在 \end 中（也就是说，每个 LaTeX 环境中）。作为进一步的优化，使用 \let 而不是 \def，以避免一次展开步骤。

```
\cs_new_eq:NN \IfHookEmptyTF \hook_if_empty:nTF
```

(*\IfHookEmptyTF* 定义结束。这个函数被记录在第13页。)

\IfHookExistsTF 标记为待移除，并且不再在文档部分记录！

PhO: \IfHookExistsTF 在 *jlreq.cls*、*pxatbegshi.sty*、*pxeverysel.sty*、*pxeveryshi.sty* 中使用，因此公共名称可能是一段时间内内部条件的别名。无论如何，这些包对 \IfHookExistsTF 的使用实际上并不正确，可以进行更改。

```
\cs_new_eq:NN \IfHookExistsTF \__hook_if_usable:nTF
```

(*\IfHookExistsTF* 定义结束。)

4.13 需要清理的弃用部分

\hook_disable:n 弃用。

\hook_provide:n
\hook_provide_reversed:n
\hook_provide_pair:nn
__hook_activate_generic_reversed:n
__hook_activate_generic_pair:nn

```
\cs_new_protected:Npn \hook_disable:n
  {
    \__hook_deprecated_warn:nn
      { hook_disable:n }
      { hook_disable_generic:n }
    \hook_disable_generic:n
  }
\cs_new_protected:Npn \hook_provide:n
  {
    \__hook_deprecated_warn:nn
      { hook_provide:n }
      { hook_activate_generic:n }
    \hook_activate_generic:n
  }
\cs_new_protected:Npn \hook_provide_reversed:n
  {
    \__hook_deprecated_warn:nn
```

```
      { hook_provide_reversed:n }
      { hook_activate_generic:n }
    \__hook_activate_generic_reversed:n
  }
\cs_new_protected:Npn \hook_provide_pair:nn
  {
    \__hook_deprecated_warn:nn
      { hook_provide_pair:nn }
      { hook_activate_generic:n }
    \__hook_activate_generic_pair:nn
  }
\cs_new_protected:Npn \__hook_activate_generic_reversed:n #1
  { \__hook_normalize_hook_args:Nn \__hook_activate_generic:nn {#1} { - } }
\cs_new_protected:Npn \__hook_activate_generic_pair:nn #1#2
  { \hook_activate_generic:n {#1} \__hook_activate_generic_reversed:n {#2} }
```

(*\hook_disable:n* 以及其它的定义结束。)

\DisableHook
\ProvideHook
\ProvideReversedHook
\ProvideMirroredHookPair

弃用。

```
\cs_new_protected:Npn \DisableHook
  {
    \__hook_deprecated_warn:nn
      { DisableHook }
      { DisableGenericHook }
    \hook_disable_generic:n
  }
\cs_new_protected:Npn \ProvideHook
  {
    \__hook_deprecated_warn:nn
      { ProvideHook }
      { ActivateGenericHook }
    \hook_activate_generic:n
  }
\cs_new_protected:Npn \ProvideReversedHook
  {
    \__hook_deprecated_warn:nn
      { ProvideReversedHook }
      { ActivateGenericHook }
    \__hook_activate_generic_reversed:n
  }
\cs_new_protected:Npn \ProvideMirroredHookPair
  {
    \__hook_deprecated_warn:nn
```

```
      { ProvideMirroredHookPair }
      { ActivateGenericHook }
    \__hook_activate_generic_pair:nn
  }
```

(*\DisableHook 以及其它的定义结束。*)

__hook_deprecated_warn:nn 警告有关弃用的信息，告知应该使用的替代方法。

```
\cs_new_protected:Npn \__hook_deprecated_warn:nn #1 #2
  { \msg_warning:nnnn { hooks } { deprecated } {#1} {#2} }
\msg_new:nnn { hooks } { deprecated }
  {
    Command~\iow_char:N\\#1~is~deprecated~and~will~be~removed~in~a~
    future~release. \\ \\
    Use~\iow_char:N\\#2~instead.
  }
```

(*__hook_deprecated_warn:nn 定义结束。*)

4.14 其他地方需要的内部命令

在这里，我们设置了一些可怕的（但是一致的）LaTeX 2ε 名称，以允许在此模块之外使用内部命令。我们必须取消 @@，因为我们想要用双“at”符号代替双下划线。

```
⟨@@=⟩
```

\@expl@@@initialize@all@@

\@expl@@@hook@curr@name@pop@@

```
\cs_new_eq:NN \@expl@@@initialize@all@@
              \__hook_initialize_all:

\cs_new_eq:NN \@expl@@@hook@curr@name@pop@@
              \__hook_curr_name_pop:
```

(*\@expl@@@initialize@all@@ 和 \@expl@@@hook@curr@name@pop@@ 定义结束。*)

这里回滚并不会取消定义接口命令，因为它们可能在没有回滚功能的包中使用。所以我们只是让它们什么都不做，这取决于代码的使用情况可能会起作用也可能不会。

```
%
⟨latexrelease⟩\IncludeInRelease{0000/00/00}{lthooks}
⟨latexrelease⟩                  {The~hook~management}%
⟨latexrelease⟩
⟨latexrelease⟩\def \NewHook#1{}
⟨latexrelease⟩\def \NewReversedHook#1{}
```

```
⟨latexrelease⟩\def \NewMirroredHookPair#1#2{}
⟨latexrelease⟩
⟨latexrelease⟩\def \DisableGenericHook #1{}
⟨latexrelease⟩
⟨latexrelease⟩\long\def\AddToHookNext#1#2{}
⟨latexrelease⟩
⟨latexrelease⟩\def\AddToHook#1{\@gobble@AddToHook@args}
⟨latexrelease⟩\providecommand\@gobble@AddToHook@args[2][]{}
⟨latexrelease⟩
⟨latexrelease⟩\def\RemoveFromHook#1{\@gobble@RemoveFromHook@arg}
⟨latexrelease⟩\providecommand\@gobble@RemoveFromHook@arg[1][]{}
⟨latexrelease⟩
⟨latexrelease⟩\def \UseHook        #1{}
⟨latexrelease⟩\def \UseOneTimeHook #1{}
⟨latexrelease⟩\def \ShowHook #1{}
⟨latexrelease⟩\let \DebugHooksOn \@empty
⟨latexrelease⟩\let \DebugHooksOff\@empty
⟨latexrelease⟩
⟨latexrelease⟩\def \DeclareHookRule #1#2#3#4{}
⟨latexrelease⟩\def \DeclareDefaultHookRule #1#2#3{}
⟨latexrelease⟩\def \ClearHookRule #1#2#3{}
```

如果未提供钩子管理，我们将存在性测试设为 false，将空性测试设为 true，希望这在大多数情况下是合理的。如果不是，那么包需要防范在旧内核中运行。

```
⟨latexrelease⟩\long\def \IfHookExistsTF #1#2#3{#3}
⟨latexrelease⟩\long\def \IfHookEmptyTF #1#2#3{#2}
⟨latexrelease⟩
⟨latexrelease⟩\EndModuleRelease

⟨@@=hook⟩

⟨latexrelease⟩\cs:w __hook_rollback_tidying: \cs_end:
⟨latexrelease⟩\bool_lazy_and:nnT
⟨latexrelease⟩    { \int_compare_p:nNn { \sourceLaTeXdate } > { 20230600 } }
⟨latexrelease⟩    { \int_compare_p:nNn { \requestedLaTeXdate } < { 20230601 } }
⟨latexrelease⟩  {
⟨latexrelease⟩    \cs_gset_protected:Npn \__hook_rollback_tidying:
⟨latexrelease⟩      {
⟨latexrelease⟩        \@latex@error { Rollback~code~executed~twice }
⟨latexrelease⟩          {
⟨latexrelease⟩            Something~went~wrong~(unless~this~was~
⟨latexrelease⟩            done~on~purpose~in~a~testing~environment).
⟨latexrelease⟩          }
```

```
⟨latexrelease⟩        \use_none:nnnn
⟨latexrelease⟩      }
⟨latexrelease⟩    \cs_set:Npn \__hook_tmp:w #1 #2
⟨latexrelease⟩      {
⟨latexrelease⟩        \__hook_tl_gset:cx { __hook#1~#2 }
⟨latexrelease⟩          {
⟨latexrelease⟩            \exp_args:No \exp_not:o
⟨latexrelease⟩              {
⟨latexrelease⟩                \cs:w __hook#1~#2 \exp_last_unbraced:Ne \cs_end:
⟨latexrelease⟩                  { \__hook_braced_cs_parameter:n { __hook#1~#2 } }
⟨latexrelease⟩              }
⟨latexrelease⟩          }
⟨latexrelease⟩      }
⟨latexrelease⟩    \seq_map_inline:Nn \g__hook_all_seq
⟨latexrelease⟩      {
⟨latexrelease⟩        \exp_after:wN \cs_gset_nopar:Npn
⟨latexrelease⟩          \cs:w g__hook_#1_code_prop \exp_args:NNo \exp_args:No
⟨latexrelease⟩            \cs_end: { \cs:w g__hook_#1_code_prop \cs_end: }
⟨latexrelease⟩        \__hook_tmp:w { _toplevel } {#1}
⟨latexrelease⟩        \__hook_tmp:w { _next } {#1}
⟨latexrelease⟩      }
⟨latexrelease⟩  }
\ExplSyntaxOff
⟨/2ekernel | latexrelease⟩

⟨@@=⟩
```

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

hook internal commands:

I

N

O

or commands:

P

prg commands:

prop commands:

Q

quark commands:

U

W

修订记录