TEXhigh ——高亮 TEX 源码

雾月*

2025 年 9 月 24 日　v0.4.0

texhigh 是用于高亮 TEX 代码的宏包，基于由 Rust 编写的命令行工具 texhigh，texhigh 宏包是对该命令行工具的进一步封装。

§ 1　主要特性

texhigh 的主要优势是速度快、可自定义程度高。

由于 texhigh 使用 Rust 编写，比起使用 Python 编写的 pygmentize 要快许多。同样是处理 4 万行左右的 `expl3-code.tex` 文件，texhigh 只需 0.2s 左右，而 pygmentize 则需 3.5s，有十几倍的差距。

texhigh 有两类配置。一类是作用于 texhigh 命令行工具的（config，“配置”），它们用于控制如何解析要高亮的代码；另一类是作用于 TEX 输出的（style，“样式”），它们用于控制如何输出解析后的内容。

texhigh 仍然属于静态解析工具，对于 TEX 这类上下文有关的语言，还无法完美地处理所有情形。尽管如此，texhigh 还是有许多可配置的地方。

texhigh 还有实验性质的计算文字布局（layout）的功能，通过它，可以输出 emoji 和颜文字等。

§ 2　主要命令

\texhighverb[p1]、\texhighfile[p2] 这两个命令分别用于高亮 TEX 代码和文件。\texhigh-text[p2] 也可用于高亮代码。\texhighinput[p2] 则用于输出已被 texhigh 处理好的内容。texhigh 用于高亮该环境的内容。

\texhighverb

```
\texhighverb [⟨选项⟩] ⟨token⟩ ⟨tokens⟩ ⟨token⟩
\texhighverb [⟨选项⟩] {⟨balanced tokens⟩}
```

用法和 \verb 类似，用于高亮 ⟨*tokens*⟩。

*longaster@163.com

\texhightext

```
\texhightext [⟨选项⟩] {⟨balanced tokens⟩}
```

把 ⟨*balanced tokens*⟩ 转为字符串，然后高亮它们。\texhighverb[P1] 内部也使用该命令。

我们可以用 \texhightext[P2] 来定义自己的高亮命令：

例 1

```
\NewDocumentCommand {\myverb} { v } {\texhightext[use-ctab=latex3]{#1}}

% 二者一样
\texhighverb[use-ctab=latex3] |\cs_set:Npn \__my_func:n #1 { \tl_to_str:n {#1} }|

\myverb|\cs_set:Npn \__my_func:n #1 { \tl_to_str:n {#1} }|
```

```
\cs_set:Npn \__my_func:n #1 { \tl_to_str:n {#1} }
\cs_set:Npn \__my_func:n #1 { \tl_to_str:n {#1} }
```

\texhighfile

```
\texhighfile [⟨选项⟩] {⟨文件⟩}
```

高亮 ⟨文件⟩。

\texhighinput

```
\texhighinput [⟨选项⟩] {⟨文件⟩}
```

导入由 texhigh 命令行工具输出的文件。

例如，使用 \texhighfile[P2] [output=myout.thv]{somefile.tex} 后，可以用 \texhighinput[P2] {myout.thv} 直接导入。

\texhighdefshortverb

```
\texhighdefshortverb [⟨选项⟩] ⟨符号⟩
```

把 ⟨符号⟩ 局部地设置为“短的”高亮命令。如 | | 那样。texhigh 默认情况下不把任何符号设置为短的高亮命令。

例 2

```
\texhighdefshortverb\| \texhighdefshortverb!

|\def\nrelax{not \relax}| !\def\nrelax{\relax not}!
```

```
\def\nrelax{not \relax} \def\nrelax{\relax not}
```

texhigh

```
\begin{texhigh} [⟨选项⟩]
    ...
\end{texhigh}
```

高亮环境内的内容。

如果要基于它来定义环境，则需使用命令的形式：

```
\NewDocumentEnvironment{myverb}{O{}}
  {\texhigh[gobble=auto,#1]}
  {\endtexhigh}
```

并且不能使用 b 参数。

也可以将 c 参数与 \texhighfile[P2] 配合使用达到类似的效果，见例 3。

例 3

```
\ExplSyntaxOn
\tl_set:Nn \l__my_filename_tl { ./ \jobname.texhigh.verb }
\NewDocumentEnvironment{myverb}{c}
  { \UseHook{end/texhigh/begin} }
  {
    \iow_open:Nn \g_tmpa_iow { \l__my_filename_tl }
    \seq_set_split_keep_spaces:Nnn \l_tmpa_seq { \obeyedline } {#1}
    \seq_map_tokens:Nn \l_tmpa_seq { \iow_now:Nn \g_tmpa_iow }
    \iow_close:N \g_tmpa_iow
```

```
    \texhighfile [gobble=auto] { \l__my_filename_tl }
    \UseHook{end/texhigh/end}
  }
\ExplSyntaxOff

\begin{myverb}
  \def\foo{foo}
  \show\foo
\end{myverb}
```

```
\def\foo{foo}
\show\foo
```

§ 3 选项

3.1 基本配置

本小节的选项都是用于设置 texhigh 的配置的。

high/use-ctab

use-ctab = ⟨*catcode table*⟩ 初始值：document

使用 ⟨*catcode table*⟩ 来解析当前要高亮的代码。字符的类别码（catcode）不同，解析得到的 token 也不同。

例如，使用 \makeatletter 会设置 @ 的类别码为 11，这使得 \@firstofone 被解析为 \@firstofone 这个控制序列，而使用 LaTeX 正文中默认的类别码时，则会解析为控制序列 \@ 以及 f$_{11}$i$_{11}$r$_{11}$s$_{11}$t$_{11}$o$_{11}$f$_{11}$o$_{11}$n$_{11}$e$_{11}$。解析得到的 token 不同，高亮结果也会有所差异。

use-ctab 就是用于设置解析时使用的类别码（更确切地说，是类别码表）。可用的值有 document，latex，latexcode，latex3，latex3code，cjk，cjkl3，cjkcode，cjkl3code 等，还可以使用自定义的类别码表，见第 3.4 节。

document 和 latex 就是 LaTeX 正文中默认使用的类别码表，latexcode 在 latex 的基础上还设置了 @ 的类别码为 11。latex3 在 latex 的基础上使用 \ExplSyntaxOn 开启后的类别码。latex3code 在 latex3 的基础上还设置了 @ 的类别码为 11。cjk 设置所有的 CJK 表意字符的类别码为 11。cjkl3 在 cjk 的基础上还设置了 @ 的类别码为 11。

high/ctab-file

ctab-file = ⟨*ctabset files*⟩

解析 ⟨*ctabset files*⟩ 中的类别码表。这使得 use-ctab 可以使用这些类别码表。文件结构见第 3.4 节。

high/config-file
high/config-file+

config-file = ⟨*config files*⟩
config-file+ = ⟨*config files*⟩

使用 ⟨*config files*⟩ 设置的配置。若有重复的键，则只使用最后的那个。

文件类型的为 TOML，具体可用的配置见第 3.5 节。

high/gobble

gobble = ⟨auto|正整数⟩ 初始值：0

每行最开始要忽略的字符数。如果为 auto，则忽略的字符数为第一行开始的空格数。

high/tabs-len

tabs-len = ⟨正整数⟩ 初始值：2

要把水平制表符替换为多少空格。

high/lines

lines = ⟨行号⟩ 初始值：0
lines = ⟨开始, 结束⟩

要高亮的代码行。为 0 或 0,0 时，高亮所有行，否则只高亮对应行（行号从 1 开始），若为一个范围，则包括开始行，但不包括结束行。

gobble 为 auto 时，只会检测保留下来的行。

high/filename

`filename = ⟨文件⟩` 初始值：`\jobname.texhigh.verb`

要高亮的文件。设置它时，会把 cache-dir P4 附加到它前面。

使用 texhigh 环境时，会把代码保存到这个文件中。

high/cache-dir

`cache-dir = ⟨路径⟩`

设置缓存路径。如果是文件夹，则必须以 / 结尾，texhigh 不会自动附加 /。且必须手动创建该文件夹，texhigh 不会创建它们，如果没有创建该文件夹，则会出错。

high/output

`output = ⟨文件⟩`

把 texhigh 的处理结果保存至 ⟨文件⟩ 中。设置它时，会把 cache-dir P4 附加到它前面。

high/no-ctabs

`no-ctabs`

清除已经设置的类别码表和 use-ctab P3 的值。

high/no-configs

`no-configs`

清除已经设置的配置。

high/enhanced

`enhanced = ⟨true|false⟩` 初始值：`true`

是否启用增强模式，增强模式有更多功能。默认启用。

high/banner

`banner = ⟨true|false⟩` 初始值：`false`

是否输出 texhigh 的版权信息。一般无需修改。

high/text-base64

`text-base64 = ⟨true|false⟩` 初始值：`true`

是否把 \texhightext P2 高亮的代码以 base64 编码后再传入 texhigh。默认启用。一般无需修改。

high/kpse

`kpse = ⟨true|false⟩` 初始值：`false`

是否使用 kpathsea 查找 config-file P3 使用的文件。启用它后，texhigh 会显著地变慢。建议使用 kpsewhich 找到对应的文件后，将其复制到工作目录，或使用绝对路径，而不是使用该选项。

3.2 样式选项

\THSaveStyle

`\THSaveStyle {⟨style⟩} {⟨code⟩}`

设置高亮风格。

\THUseSavedStyle

`\THUseSavedStyle {⟨style⟩}`

使用由 \THSaveStyle P4 保存的风格。

high/style

`style = ⟨styles⟩`

使用由 \THSaveStyle P4 保存的风格。

high/line-number
high/linenos

`line-number|linenos = ⟨true|false⟩` 初始值：`false`

是否开启行号。

high/first-line-number
high/first-linenos

`first-line-number|first-linenos = ⟨起始行号⟩`

texhigh 每次高亮时不会重新设置行号，需要手动设置起始行号。

high/line-number-space
high/line-number-space+
high/line-number-space-

line-number-space = ⟨长度⟩ 初始值：3mm
line-number-space+ = ⟨长度⟩
line-number-space- = ⟨长度⟩

设置（或增加，或减少）行号与代码之间的距离。

high/line-number-format

line-number-format = ⟨*code*⟩ 初始值：\scriptsize\sffamily #1

设置行号的格式。#1 代表行号。

一般情况下，它不必使用 \arabic 等命令，而由 the-line-number 修改。

high/the-line-number

the-line-number = ⟨*code*⟩ 初始值：\arabic{TeXHighLine}

修改 TeXHighLine 计数器的显示方式。TeXHighLine 用于保存当前行号。

注意：texhigh 遇到新行时不会递增行号，只会在需要显示行号时才递增。

high/line-number-pos

line-number-pos = ⟨left|right|both⟩ 初始值：both

行号的位置。

high/line-kind

line-kind = ⟨enhanced|normal|none⟩ 初始值：normal

每个代码行的开始和结尾也可以自定义。默认情况下（normal），只会插入行号和换行。如果设置为 enhanced，则可以使用代码槽（socket）texhigh/start-line、texhigh/between-line、texhigh/end-line 来添加代码。

texhigh/start-line
texhigh/between-line
texhigh/end-line

这三个代码槽（socket）会在指定位置执行。可通过 \AssignSocketPlug 来使用不同的插件（plug）。关于 socket 和 plug，见 ltsockets-doc.pdf。

\texhigh@start@line

钩子，它后面紧跟当前代码行。可在 texhigh/start-line 代码槽中修改该命令用以检测当前代码行。

例 4

```
\makeatletter
\ExplSyntaxOn
\clist_const:Ne \c__my_comment_range_clist { \tl_to_str:n { comment, } }
\cs_new:Npn \@detect@start@comment #1
  {
    \tl_if_eq:nnTF {#1} { \THrs }
      { \@detect@start@comment@ }
      { \texhigh@default@start@line #1 }
  }
\cs_new:Npn \@detect@start@comment@ #1
  {
    \clist_if_in:NoF \c__my_comment_range_clist
      { \tl_to_str:n {#1} }
      { \texhigh@default@start@line }
    \THrs {#1}
  }
\ExplSyntaxOff
\NewSocketPlug{texhigh/start-line}{detect-start-comment}
  {\let\texhigh@start@line\@detect@start@comment}
\THSaveStyle{line-number-skip-comment}{%
  \SetKeys[texhigh/high]{line-kind=enhanced, linenos, first-linenos=1}%
  \AssignSocketPlug{texhigh/start-line}{detect-start-comment}%
}
\makeatother

\begin{texhigh}[style=line-number-skip-comment, gobble=auto]
  Non-Comment Line
  % A Comment Line
```

```
  % B Comment Line
  Non-Comment Line
    % C Comment Line but spaces
\end{texhigh}
```

```
1 Non-Comment Line
  % A Comment Line
  % B Comment Line
2 Non-Comment Line
3   % C Comment Line but spaces
```

high/left-space
high/left-space+
high/left-space-
high/right-space
high/right-space+
high/right-space-

left-space = ⟨长度⟩
left-space+ = ⟨长度⟩
left-space- = ⟨长度⟩

设置（或增加，或减少）代码距离文字左右两端的距离。

high/font
high/font+

font = ⟨*code*⟩ 初始值：\ttfamily\raggedright

设置代码的字体。texhigh 还设置了 \linespread{1}，可以在此选项内修改间距因子。

high/before-text
high/before-text+
high/extra-code
high/extra-code+

before-text = ⟨*code*⟩
extra-code = ⟨*code*⟩

在设置完所有选项后执行 ⟨*code*⟩。

high/before
high/before+
high/after
high/after+

before = ⟨*code*⟩

钩子的执行顺序为 ⟨*options*⟩ ⟨*font*⟩ ⟨*before-text*⟩ ⟨*extra-code*⟩ ⟨*before*⟩ *highlighted code* ⟨*after*⟩。

high/save-catcode

save-catcode
save-catcode = ⟨*code*⟩ 初始为空

用于保存当前的类别码。使用 \@texhigh@reset@ctab p14 可恢复到此处的类别码。⟨*code*⟩ 可以修改这个宏。默认情况下只会保存 \dospecials 里的字符的类别码。

\texhighdefstyle

\texhighdefstyle {⟨*full path*⟩} {⟨*options*⟩}
\texhighdefstyle [⟨*module*⟩] {⟨*path*⟩} {⟨*options*⟩}
\texhighdefstyle * {⟨*full path*⟩} {⟨*code*⟩}
\texhighdefstyle * [⟨*module*⟩] {⟨*path*⟩} {⟨*code*⟩}

自定义额外的选项。不带星号的，⟨*options*⟩ 为同一 ⟨*module*⟩ 下的选项，带星号的 ⟨*code*⟩ 为代码。⟨*options*⟩ 和 ⟨*code*⟩ 都可使用一个参数，为该选项的值。

⟨*module*⟩ 就是选项的灰色部分（不含末尾的 /）。例如，high/save-catcode p6 的 *module* 就是 high，*path* 就是 save-catcode，*full path* 就是 high/save-catcode。

例如，使用

```
\texhighdefstyle[high]{my option 1}{
  first-line-number=1, line-number,
  line-number-pos=left, line-number-space=5mm,
}
\texhighdefstyle*[high]{my len 1}{\def\mylen{#1}}
```

后，就可以在 \texhighfile p2 等命令或环境中使用 \texhighfile[my option 1, my len ↩
↪ 1=10]{..} 了。

3.3 配置选项

配置选项基本都有对应的 texhigh 的命令行选项和配置文件中的键。它们当中有许多都是 texhigh 命令行选项的封装。

high/break-at

```
break-at = {⟨字符列表⟩}                                    初始值：\ , \^^I
break-at = [⟨salt⟩] {⟨字符列表⟩}
```

设置在哪些字符后插入可断点。默认为空格和水平制表符。

⟨*salt*⟩ 为唯一标识符，由字母、数字等组成。如果不给出 ⟨*salt*⟩ 或两个 ⟨*salt*⟩ 相同，则后面的那个会覆盖之前的。

high/char-replacements

```
char-replacements = { ⟨char⟩, ... }
char-replacements = { ⟨char⟩=⟨repr⟩, ... }
```

高亮时替换 ⟨*char*⟩，包括普通字符和控制序列名称[1]里的字符。

可以用 \THSetCharReplacement 来修改 ⟨*char*⟩ 要被替换为什么。例如，设置

```
char-replacements={\ =\textvisiblespace}
```

可以把空格替换为 ␣（\textvisiblespace）。

如果设置了 ⟨*repr*⟩，相当于隐式使用 \THSetCharReplacement。

\THSetCharReplacement

```
\THSetCharReplacement   {⟨char⟩} {⟨repr⟩}
\THSetCharReplacement * {⟨char⟩} {⟨repr⟩}
```

设置 ⟨*char*⟩ 的替换文本，在显示 ⟨*char*⟩ 是，会替换为 ⟨*repr*⟩。若使用不带星号的版本，则还会使用 \ifincsname 判断是否在控制序列的名里，若是，则不替换。

high/this-cs

```
this-cs = ⟨code⟩
```

设置所有控制序列的样式。

例 5

```
\def\thiscsstyle#1{{\THcolor{blue}#1}}
\texhighverb[this-cs=\thiscsstyle]{\relax}

\makeatletter
\def\thiscsstyle@#1#2{{\THcolor{blue}#1\underline{#2}}}
\def\thiscsstyle#1{\thiscsstyle@#1} % #1 为 {<escape char>}{<cs name>}
\makeatother
\texhighverb[this-cs=\thiscsstyle]{\relax}
```

\relax

\relax

high/escape-inside

```
escape-inside = {⟨命令⟩}
escape-inside = ⟨char1⟩⟨char2⟩
```

把 ⟨命令⟩ 的参数“逃逸”（escape），或让 ⟨$char_1$⟩ 和 ⟨$char_2$⟩ 中间的内容“逃逸”。

所谓“逃逸”，就是指让它们不高亮，而保留其原始形式。

例 6

```
\begin{texhigh}[escape-inside=\E, escape-inside=&&]
  The \E{\textbf{text}} will be &\itshape escaped&.
  Even the \E{\verb|\relax|} works.
\end{texhigh}
```

The **text** will be *escaped*.

[1] 所谓控制序列名称就是除开转义符的其余部分，如 \relax 的名称就是 relax。

```
Even the \relax works.
```

high/math-escape
high/comments-math-escape
high/non-comments-math-escape

math-escape
math-escape = ⟨**any**|in-comments|non-comments⟩
comments-math-escape 不可设置值
non-comments-math-escape 不可设置值

math-escape 把 $ $ 和 \(\) 中间的内容作为数学公式。值为 in-comments 时，它们需出现注释中才有效，为 non-comments，则不是出现在注释中才有效，为 any 则不做限制，这也是默认情形。

例 7

```
\texhighverb[math-escape]{A \textit{cr} = $\rho \times \pi$ = \(\frac{N}{m}\)}.

\begin{texhigh}[gobble=auto, comments-math-escape]
  A \textit{cr} = $\rho \times \pi$ = \(\frac{N}{m}\)
  % A \textit{cr} = $\rho \times \pi$ = \(\frac{N}{m}\)
\end{texhigh}
```

A \textit{cr} = $\rho \times \pi$ = \(\frac{N}{m}\).
A \textit{cr} = $\rho \times \pi$ = $\frac{N}{m}$
% A \textit{cr} = $\rho \times \pi$ = $\frac{N}{m}$

high/texcomments
high/texcl

texcomments|texcl = ⟨**true**|false⟩ 初始值：**false**

使整个注释像正常文本那样不高亮，直接输出。

例 8

```
\begin{texhigh}[gobble=auto,texcomments]
  A \textbf{normal} text.
  % A \textbf{comment} text.
\end{texhigh}
```

A \textbf{normal} text.
A **comment** text.

high/char-category*

char-category* = {⟨*name*⟩} {⟨*regex*⟩} {⟨*style*⟩}

设置字符的类别及其样式。只对 token 为字符的才生效，不会影响控制序列名称里的字符。

⟨*name*⟩ 为名字，⟨*regex*⟩ 为（纯文本）正则表达式（见第 5 节），⟨*style*⟩ 可使用 1 个参数，为当前字符。

当 ⟨*style*⟩ 为 \THPASS[P11] 这个特殊值时，只设置类别而不修改样式，如果你已经定义了样式，而不想修改它，可以使用这个特殊值。

例 9

```
\texhighverb[char-category*={symbol}{[\ \$\&]}{\mbox{\THcolor{red}#1}}]
  |\def\a #1{$#1$} \a {\ b&\&}|
```

\def\a #1{$#1$} \a {\ b&\&}

例 10

```
\THSetCharReplacement{\ }{\textvisiblespace} % texhigh 已设置
\THSetCharReplacement{\$}{\S} % 把 $ 替换为 \S
\texhighverb[char-replacements={\ \$}, % 设置哪些字符要替换
  char-category*={symbol}{[\ \$\&]}{\mbox{\color{red}#1}}, % 修改颜色
] |\def\a #1{$#1$} \a {\ b&\&}|
```

\def␣\a␣#1{§#1§}␣\a␣{␣b&\&}

high/char-category

char-category = ⟨options..⟩

设置字符的类别。⟨*options*⟩ 是（多个）可选值。目前 texhigh 没有提供任何可选值。

例 11

```
\texhighdefstyle[high]{char-category/my symbol}
  % 注意这里要用 ##1，因为 #1 代表 “my symbol”
  {char-category*={symbol}{[\ \$\&]}{\mbox{\THcolor{red}##1}}}

\texhighverb[char-category=my symbol]{\def\a #1{$#1$} \a {\ b&\&}}
```

```
\def\a #1{$#1$} \a {\ b&\&}
```

\THSetCH
\THLetCH

```
\THSetCH {⟨name⟩} {⟨code with 1 arg⟩}
\THLetCH {⟨name1⟩} {⟨name2⟩}
```

设置字符的类别为 ⟨*name*⟩ 的样式。或把 ⟨$name_2$⟩ 样式复制给 ⟨$name_1$⟩。

⟨*code*⟩ 可使用 1 个参数，为当前字符。

char-category* P.8 内部就是使用该命令。\THPASS P.11 在此处无效。

high/cs-category
high/cs-category*

```
cs-category  = {⟨name⟩} {⟨cs name list⟩} {⟨style⟩}
cs-category* = {⟨name⟩} {⟨regex⟩} {⟨style⟩}
```

设置控制序列的类别及其样式。

⟨*name*⟩ 为名字，⟨*cs name list*⟩ 为控制序列名称列表，⟨*regex*⟩ 为（纯文本）正则表达式（见第 5 节），⟨*style*⟩ 可使用 2 个参数，分别为转义符和控制序列名。

当 ⟨*style*⟩ 为 \THPASS P.11 这个特殊值时，只设置类别而不修改样式，如果你已经定义了样式，而不想修改它，可以使用这个特殊值。

\THSetCS
\THLetCS

```
\THSetCS {⟨name⟩} {⟨code with 2 args⟩}
\THLetCS {⟨name1⟩} {⟨name2⟩}
```

设置控制序列的类别为 ⟨*name*⟩ 的样式。或把 ⟨$name_2$⟩ 样式复制给 ⟨$name_1$⟩。

⟨*code*⟩ 可使用 2 个参数，分别为转义符和控制序列名。比如控制序列 \relax，转义符为“\”，控制序列名为“relax”。

cs-category P.9 和 cs-category* P.9 内部就是使用该命令。\THPASS P.11 在此处无效。

high/lexer
high/lexer*

```
lexer* = {⟨lexer cat_start⟩} {⟨lexer cat_end⟩} {⟨ctabs⟩} {⟨action⟩} {⟨value_action⟩}
lexer* = [⟨salt⟩] {⟨lexer cat_start⟩} {⟨lexer cat_end⟩} {⟨ctabs⟩} {⟨action⟩} {⟨value_action⟩}
lexer  = ⟨options..⟩
```

用于设置如何解析要高亮的代码。

传递给 texhigh 的并非是 TEX tokens，而是纯文本。由于 TEX 是上下文有关的，解析得到的 tokens 会根据前文不同而不同。在静态解析时，无法自动应用宏执行过程时触发的各类效果，如类别码发生改变等等都不会被 texhigh 检测到。为此，texhigh 支持通过检测当前所处的上下文来半自动应用宏执行后产生的效果。

⟨*lexer cat*⟩ 为触发 lexer 改变的条件。可以是纯文本、正则表达式、TEX 正则表达式、位置（行、列号）等。正则表达式和 TEX 正则表达式检测的内容为当前位置之前的所有符号。

每种 ⟨*action*⟩ 对应 ⟨*value*⟩ 的形式也不同。可用的 ⟨*action*⟩ 为：

CatCode 修改类别码。⟨*value*⟩ 为类别码表名或 ⟨*char*⟩=⟨*catcode*⟩ 构成的列表。

EndLine 修改行结束时插入的字符。⟨*value*⟩ 为数字。

high/lexer-catcode
high/lexer-catcode*
high/extra-catcode*

```
lexer-catcode* = {⟨lexer cat_start⟩} {⟨lexer cat_end⟩} {⟨value⟩}
lexer-catcode  = ⟨options..⟩
extra-catcode* = {⟨value⟩}
```

lexer* P.9 的 ⟨*action*⟩ 为 CatCode 的特例。

extra-catcode 是 ⟨$lexer\ cat_{start}$⟩ 为 0，⟨$lexer\ cat_{end}$⟩ 为空的特例，相当于整个代码段都使用 ⟨*value*⟩。

high/no-lexer

`no-lexer` 不可设置值

清除当 ⟨*salt*⟩ 为空（或未设置）时的 lexer*[P.9] 所设置的 lexer。

high/enabled-ranges
high/enabled-ranges+
high/remove-enabled-ranges
high/remove-enabled-ranges+

```
enabled-ranges = {⟨range name list⟩}
remove-enabled-ranges = {⟨range name list⟩}
```

texhigh 通过 range 机制检测命令的参数。range 就是一个特殊的代码片段，可为不同的代码片段设置不同的样式。设置 range 并不会一定会启用它，texhigh 只会启用指定的 range，这样可用把需要的 range 先写在配置文件中，然后按需启用。

high/range

`range = {⟨full name⟩} {⟨range actions⟩}`

创建并启用 ⟨*full name*⟩。

range 可以分为普通代码段和逃逸代码段。普通代码段会被高亮，可以为其单独设置样式，也支持检测参数。逃逸代码段不会被高亮，而会直接执行。同一个 ⟨*full name*⟩ 只能使用其中的一种，要么是普通代码段，要么是逃逸代码段。不过，⟨*full name*⟩ 支持一种特殊的语法，当它为 +{⟨*salt*⟩}⟨*real name*⟩ 时，texhigh 认为其名为 ⟨*real name*⟩，同时 enabled-ranges[P.10] 也可以只用 ⟨*real name*⟩，texhigh 会启用所有 ⟨*full name*⟩，只要其 ⟨*real name*⟩ 包含在 enabled-ranges[P.10] 中。

⟨*range actions*⟩ 也为键值选项，其 ⟨*module*⟩ 为 `high/range-config`：

high/range-config/escape

`escape = ⟨true|false⟩` 重设为：**false**

是否为逃逸代码段。

high/range-config/start
high/range-config/start*

```
start  = {⟨range cat⟩}    重设为空
start* = {⟨range cat*⟩}   重设为空
```

此 range 何时开始。⟨*range cat*⟩ 为纯文本或正则表达式或 TEX 正则表达式。检测的内容为当前位置及其后的所有内容。

high/range-config/arguments

`arguments = {⟨arguments⟩}` 重设为空

⟨*arguments*⟩ 为 ltcmd（xparse）的参数，参数数目最多为 9 个。

⟨*argument*⟩ 包括 `m o O d D r R s t v` 以及 `l u g G` 暂不支持 `e E c b`，并且 `! + = >` 也是无效的。此外，还支持一个特殊的符号 `^^J`，它用于捕获当前行剩下的内容，带一个参数，表示是否要求大括号成对存在。

high/range-config/remove-start

`remove-start = ⟨true|false⟩` 重设为：**false**

移除该代码段的 ⟨*start*⟩ 部分。仅为逃逸代码段时有效。

high/range-config/insert-brace

`insert-brace = ⟨true|false⟩` 重设为：**false**

把该代码段的参数用一对 `{ }` 括起来。仅为逃逸代码段时有效。

high/range-config/use-argument

`use-argument = ⟨true|false⟩` 重设为：**false**

移除参数最外层的括号或定界符。仅为逃逸代码段时有效。

high/range-config/insert-ending

`insert-ending = ⟨true|false⟩` 重设为：`false`

若最后那个参数类型为 u、U 或 ^^J 时，把标记参数结尾的符号移动到该代码段的后面。若为逃逸代码段，则还需 use-argument P.10 为 true。

例如，当使用参数 ^^J 时，标记该参数结尾的符号为换行符，设置该选项为真时，会把该换行符移动到该代码段的后面，这样 texhigh 才能正确检测行的结束。

high/range-config/in-comments

`in-comments = ⟨required|never|any⟩` 重设为：`any`

是否仅在注释内才检测该 range。

值为 required / must / true 要求必须在注释内，值为 forbidden / never / prohibited / false 要求不能在注释内，值为 irrelevant / any / dontcare 则不做限制。

high/range-config/start-is-arg

`start-is-arg = ⟨true|false⟩` 重设为：`false`

检测参数时是否包括 ⟨*start*⟩。

high/range-config/args-numbered

`args-numbered = ⟨true|false⟩` 重设为：`false`

texhigh 把每个参数都作为单独的一个 range，名为 argument.⟨*arg type*⟩ 或 argument.⟨*arg pos*⟩，默认为前者，若 args-numbered P.11 为真时，则使用后者。

⟨*arg type*⟩ 为参数类型，如 m、D 等，⟨*arg pos*⟩ 为参数的位置，如 1、2、9 等，从 1 开始，最多为 9。

high/range-config/skip-if-pre
high/range-config/skip-if-post

`skip-if-pre = {⟨regex⟩}` 重设为空
`skip-if-post = {⟨regex⟩}` 重设为空

在寻找完该 range 的参数后，若该 range 之前（或之后）的内容匹配 ⟨*regex*⟩，则跳过该 range，继续寻找可能的 range。

\THSetRange

```
\THSetRange {⟨range name⟩}    {⟨start code⟩}[⟨end code⟩]
\THSetRange {⟨range name⟩}    [⟨range action⟩] {⟨start code⟩}[⟨end code⟩]
\THSetRange {⟨range name⟩} * {⟨start code⟩}[⟨end code⟩]
\THSetRange {⟨range name⟩} * [⟨range action⟩] {⟨start code⟩}[⟨end code⟩]
```

设置 range 及其样式。

若不给出 ⟨*range action*⟩，则只修改样式，否则还会创建并启用该 range。

不带星号的版本会在 ⟨*start code*⟩ 之前加上 \begingroup 以及 \@texhigh@reset@ctab P.14，在 ⟨*end code*⟩ 之后加上 \endgroup。

\THSetRS
\THSetRE
\THSetES
\THSetEE
\THLetRS
\THLetRE
\THLetES
\THLetEE

```
\THSetRS {⟨range name⟩} {⟨start code⟩}
\THSetRE {⟨range name⟩} {⟨end code⟩}
```

设置普通代码段和逃逸代码段的样式。代码段像环境一样，可以分别设置开始和结尾要执行的代码。

\THSetRange P.11 就是同时设置普通代码段和逃逸代码段的样式。

\THCollectRange

```
\THCollectRange {⟨do code⟩} \THrs ⟨text code⟩ \THre
\THCollectRange {⟨do code⟩} \THes ⟨text code⟩ \THee
```

收集普通代码段或逃逸代码段。⟨*do code*⟩ 后面紧跟着 {⟨*text code*⟩}。

\THPASS

特殊的标记。

high/__config
high/__config*

```
__config  = {⟨raw key⟩} {⟨raw value⟩}
__config* = {⟨raw key⟩} {⟨raw value⟩}
```

设置配置。尽量不要使用它们。

3.4 类别码表

类别码表由方括号括起来的表名和 ⟨*chars*⟩=⟨*catcode*⟩ 组成。

预定义的类别码表位于 texhigh-rs 的仓库的 `prelude-ctabset.thcs` 内。

3.5 配置文件

配置文件的格式为 TOML。

texhigh-rs 的仓库里 `prelude-config.toml` 给出了一个完整的配置文件。

3.6 token 的分类

texhigh 把不同的 token 分成几类（*class*），每个类型都可设置不同的类别（*category*），其中可设置样式的类型有：

bp 可断点；主要有两个类别：cs 和 char，cs 的断点在控制序列前插入，char 在字符后插入；

cs 控制序列；除了由 cs-category P.9 等设置之外，还有 latex3.primitive、latex3.function.internal、latex3.function.public、latex3.function.kernel、latex3.variable.internal、latex3.variable.public、latex3.variable.kernel、latex.programming、latex.internal、latex.document、primitive.knuthtex、primitive.etex、primitive.pdftex、primitive.xetex、primitive.luatex、primitive.uptex、primitive.widely、primitive.sometex、primitive.luametatex；

ch 字符；除了由 char-category P.8 等设置之外，对于组开始符和组结束符，有类别：group.⟨*group level*⟩；对于其它类别码不为 11 及 12 的字符，有类别：catcode.⟨*catcode*⟩；

rs 普通代码段的开始；除了由 range P.10 设置的之外，还有 math.inline、comment；

re 普通代码段的结束；同 rs；

st 文本；它仅包含类别码为 10、11、12 的字符；

es 逃逸代码段的开始；

ee 逃逸代码段的结束；

pn 标点。

它们都有对应的 \THSet⟨*class*⟩ 和 \THLet⟨*class*⟩ 命令用于设置样式，其中 ⟨*class*⟩ 需大写，如 \THSetCS P.9。

\THSetBP
\THLetBP
\THSetST
\THLetST
\THSetPN
\THLetPN

```
\THSetBP {⟨name⟩} {⟨code⟩}
\THSetST {⟨name⟩} {⟨code with 1 arg⟩}
\THSetPN {⟨name⟩} {⟨code with 1 arg⟩}
```

设置对应类型的样式。

\THRemoveClass
\THRemoveClasses

```
\THRemoveClass {⟨class⟩} {⟨name list⟩}
\THRemoveClasses {⟨class list⟩}
```

\THRemoveClass P.12 删除类型为 ⟨*class*⟩ 的名为 ⟨*name*⟩ 的样式。\THRemoveClasses P.12 删除类型为 ⟨*class*⟩ 的除“?”之外的所有样式。

\THSetFallback

```
\THSetFallback {⟨class⟩} {⟨name⟩} {⟨fallback names⟩}
```

如果 ⟨*class*⟩ 的没有设置名为 ⟨*name*⟩ 的样式，则依次使用 ⟨*fallback names*⟩ 中的样式，如果它们都未设置，则使用“?”样式。

§ 4 辅助命令

本节的命令一般可在设置样式时使用。

\THcolor
\THColorStatus

```
\THcolor {⟨颜色表达式⟩}
\THcolor [⟨色彩模式⟩] {⟨color spec⟩}
\THColorStatus {⟨status⟩}
```

设置文本和填充的颜色。与 xcolor 的 \color 命令相比，\THcolor P.13 可以通过设置 ⟨*status*⟩ 来启用或取消设置该颜色，如果 ⟨*status*⟩ 为 0，则不设置颜色，否则设置对应颜色。

\texhigh@fallback ⋆
\texhigh@fallback@ ⋆

```
\texhigh@fallback  {⟨class⟩} {⟨name⟩}
\texhigh@fallback@ {⟨class⟩} {⟨name⟩}
```

这两个命令展开为没有发现名为 name 的样式时，要使用的样式。

\texhigh@fallback@ P.13 已经查找了其 fallback 列表中的样式，\texhigh@fallback P.13 则还没有查找 fallback 列表。可以修改它们以使用不同的 fallback 策略。它们必须完全可展。

\texhigh@cat@if@exists ⋆
\texhigh@cat@fallback ⋆

```
\texhigh@cat@if@exists {⟨class⟩} {⟨cat name⟩} {⟨true⟩} {⟨false⟩}
\texhigh@cat@fallback {⟨class⟩} {⟨cat name⟩} {⟨false⟩}
```

\texhigh@cat@if@exists P.13 检测 ⟨*class*⟩ 是否有名为 ⟨*cat name*⟩ 的样式。\texhigh@cat@fallback P.13 查找 ⟨*class*⟩ 的 ⟨*cat name*⟩ 的 fallback 列表，展开为样式存在时的样式名，若这些样式都不存在，则使用 ⟨*false*⟩。

\texhigh@find@dotparent ⋆
\texhigh@dot@split ⋆

```
\texhigh@dot@split {⟨name⟩}
\texhigh@find@dotparent {⟨name⟩}
```

\texhigh@dot@split P.13 把 ⟨*name*⟩ 分成两份，第一份为最后一个句点之前的内容，第二份为最后一个句点之后的内容。这两份都用 \exp_not:n 保护起来。\texhigh@find@dotparent P.13 则只保留第一份。

例 12

```
\makeatletter
\def\printstr#1{\detokenize\expandafter{\expanded{#1}}}
\printstr{\texhigh@dot@split {\a.b.c.\d}}\quad
\printstr{\texhigh@find@dotparent {\a.b.c.\d}}\quad
\makeatother
```

{\a .b.c}{\d } \a .b.c

例 13

```
\makeatletter\ExplSyntaxOn
% 让它依次检测该样式的“父样式”，即最后一个句点之前的内容
% 如对于 a.b.c.d，会依次检测 a.b.c，a.b 和 a，若都不存在则使用 ?
% 我们修改 \texhigh@fallback@ 而非 \texhigh@fallback，这样它会首先检测 fallback 列表
\cs_set:Npn \texhigh@fallback@ #1#2
  { \__my_texhigh_fallback_dot:ne {#1} { \texhigh@find@dotparent {#2} } }
\cs_new:Npn \__my_texhigh_fallback_dot:nn #1 #2
  {
    \tl_if_empty:nTF {#2} { ? }
      {
        \texhigh@cat@if@exists {#1} {#2} {#2}
          { \__my_texhigh_fallback_dot:ne {#1} { \texhigh@find@dotparent {#2} } }
      }
  }
\cs_generate_variant:Nn \__my_texhigh_fallback_dot:nn { ne }
\ExplSyntaxOff\makeatother
```

```
\THRemoveClass{cs}{primitive.knuthtex}
\THSetFallback{cs}{primitive.knuthtex}{unknown-foo}
\THSetCS{primitive}{\mbox{\THcolor{blue}\bfseries #1#2}}
\texhighverb|\def|
```

\def

`\@texhigh@reset@ctab`
`\@texhigh@reset@font`

钩子，分别用于重设类别码和字体。

`\@texhigh@rescan@lines`

```
\@texhigh@rescan@lines {⟨balanced tokens⟩}
\@texhigh@rescan@lines ⟨token⟩ ⟨tokens⟩ ⟨token⟩
```

重新扫描 ⟨*tokens*⟩，可使得 `\verb` 等命令生效。和 `\verb` 一样，⟨*tokens*⟩ 不能作为命令的参数。

`\@texhigh@underline`

```
\@texhigh@underline {⟨text⟩}
```

给不可断行的 ⟨*text*⟩ 加上下划线，该下划线的长度比 ⟨*text*⟩ 的略短。

`\@texhigh@shadetext`

```
\@texhigh@shadetext {⟨tikz options⟩} {⟨text⟩}
```

给 ⟨*text*⟩ 加上阴影或底纹。⟨*tikz options*⟩ 一般包含创建阴影或类似的选项，如 `left color=red, right color=blue` 或 `fill stretch image=image.png` 等。

需自行加载 tikz 宏包，或启用 texhigh 的 `tikz` 宏包选项。

`\texhigh@replicate ⋆`

```
\texhigh@replicate {⟨num⟩} {⟨code⟩}
```

重复 ⟨*code*⟩ ⟨*num*⟩ 次。

`\texhigh@pdfliteral`

```
\texhigh@pdfliteral {⟨pdf literal⟩}
```

⟨*pdf literal*⟩ 会被完全展开。

`\THmB` `\THmC` `\THmD` `\THmH` `\THmN` `\THmP` `\THmR` `\THmS` `\THmT`

特殊的标记：

\THmB	\THmC	\THmD	\THmH	\THmN	\THmP	\THmR	\THmS	\THmT
\	^	$	#	^^J	%	^^M	⟨*space*⟩	⟨*tab*⟩

§ 5 正则表达式

这里的正则表达式指的是纯文本正则表达式（简写为 *regex*），也就是只对纯文本生效，此外还有 TEX 正则表达式（简写为 *regtex*），只对 tokens 生效，见第 6 节。

纯文本正则表达式的语法见 https://docs.rs/regex/latest/regex/#syntax。

§ 6 TEX 正则表达式

TEX 正则表达式是作用于 TEX tokens 的正则表达式，l3regex 就是此类。与普通的正则表达式相比，它多了检测 token 的转义序列，例如 `\c{relax}` 检测 token 是否为 `\relax`，只有 `\relax` 才匹配此转义序列，普通正则表达式就做不到这一点。

texhigh 支持的 *regtex* 的语法和 l3regex 很相似，但暂不支持 `\b` `\B` `\G` `\u` 这几个转义序列，以及 `\c` 转义序列的否定形式（即暂不支持 `[^\c{begin}\c{end}]` 这类用法）。

§ 7　高亮原理

texhigh 高亮代码有两步，首先把原始的代码传递给 texhigh 命令行工具，它解析 TEX 代码，检测 token 的分类以及类别，输出为特殊的 TEX 代码，除了逃逸代码段外，只有有限的几个控制序列：\THls、\THle、\THin、\THcr、\THbp、\THcs、\THch、\THrs、\THre、\THst、\THes、\THee、\THpn，一般情况下不应重定义它们。然后根据这几个控制序列的参数不同得到不同的样式。

§ 8　实验特性

8.1　计算文字布局

使用本节所述的特性需自行加载 fontspec 宏包。

texhigh 有 \kaomoji P.15 命令用于排布文字，文字的显示效果部分取决于系统拥有的字体和所设置的字体。

\kaomoji

```
\kaomoji    {⟨options⟩} {⟨plain text⟩}
\kaomoji * {⟨options⟩} {⟨plain text⟩} {⟨image command⟩}
```

不带星号的命令会由 texhigh 计算布局（包括文字位置、字体等）再使用 TEX 排版出来，由 TEX 输出文字。而带星号的命令由 texhigh 排版 ⟨*plain text*⟩，将其输出为图片，TEX 只负责导入该图片。

⟨*image command*⟩ 是加载图片的命令，如 \includegraphics{⟨*options*⟩} 等。

layout/system-fonts

```
system-fonts = ⟨true|false⟩                                    初始值：true
```

是否使用系统字体。

layout/fonts
layout/fonts+

```
fonts = {⟨font list⟩}
```

设置文字可用的字体。

layout/fontsize
layout/fontsize*

```
fontsize  = {⟨dim⟩}
fontsize* = {⟨fontsize command⟩}
```

设置字体大小。

layout/lineheight

```
lineheight = {⟨dim⟩}
```

设置行高。

layout/force

```
force = ⟨true|false⟩                                           初始值：true
```

使用图片模式时，当图片已经存在时是否重新生成。

layout/cache-dir

```
cache-dir = {⟨directory⟩}
```

同 cache-dir P.4。

§ 9 版本历史

v0.4.0 (2025/09/24)

添加文档。 1

§ 10 代码索引

粗体的数字表示描述对应索引项的页码；意大利体的数字表示使用对应索引项的页码。

K

L

layout 的选项：

S

代码槽（Socket）：

T

TEX and LATEX 2ε的命令：