LaTeX3 教程一：简介

项子越

ziyue.alan.xiang@gmail.com

https://github.com/xziyue/latex3-chinese-video

2020 年 8 月 25 日

- 提供类似现代编程语言的语法与命名
- 简化 LaTeX 的宏展开控制
- 标准化 LaTeX 各个功能的接口

- LaTeX3 并不是为一般用户而设计的：它是用来实现高级用户接口的一套底层方法。

```
\def\FV@GetLine{\@noligs\expandafter\FV@CheckScan\FancyVerbGetLine}
%% DG/SR modification end
\begingroup
\catcode`\^^M=\active%
\gdef\FancyVerbGetLine#1^^M{%
  \@nil%
  \FV@CheckEnd{#1}%
  \ifx\@tempa\FV@EnvironName%            % True if end is found
    \ifx\@tempb\FV@@@CheckEnd\else\FV@BadEndError\fi%
    \let\next\FV@EndScanning%
  \else%
    \def\FV@Line{#1}%
    \def\next{\FV@PreProcessLine\FV@GetLine}%
  \fi%
  \next}%
\endgroup
```

（节选自fancyvrb）

```
\newcommand{\testcmda}{abc}
\newcommand{\testcmdb}{def\testcmda}
\par\testcmdb
\renewcommand{\testcmda}{def}
\par\testcmdb
```

defabc
defdef

```
string a("abc");

string func_1(string *a){
    // 如果原 a 的值发生改变，那么返回值也会改变
    return "def" + *a;
}

string func_2(string a){
    // 即使原 a 的值发生改变，返回值也不会改变
    return "def" + a;
}
```

```
\par\uppercase{abcdefghi}
\newcommand{\testcmda}{def}
\par\uppercase{abc\testcmda ghi}
```

ABCDEFGHI
ABCdefGHI

- 整数
- 浮点数
- 布尔逻辑
- 凭据表（token list）
- 字符串
- 文件 IO

我们键入 tex 文件中的所有内容都可视作凭据表。

凭据表内的三类对象：

- 字符
- 命令
- 凭据表

```
{abc \def {abc\def} ghi}
```

{abc \def {abc\def} ghi}

LaTeX3 命名法提倡将函数的来源以及参数类型、变量的类型编码到其名字内。

- 可以更方便地让用户区别命令与变量
- 可以避免不同宏包之间命令的冲突
- LaTeX 并不拥有真正的类型系统，这样的命名方式可以让用户在编程时自行检查错误
- 只是一套指导意见，并不是强制性的要求

\< 作用域 >_< 介绍 >_< 类型 >　　\< 作用域 >__< 介绍 >_< 类型 >

作用域

- l：局部变量
- g：全局变量
- c：常量

```
\l_tmpa_tl
\g_tmpa_int
\c_left_brace_str
```

类型

- tl: 凭据表
- str：字符串
- int：整型
- fp：浮点数
- seq：队列
- dim：尺度/长度

⋮

`\<` 模块 `>_<` 介绍 `>:<` 参数列表 `>` `\__<` 模块 `>_<` 介绍 `>:<` 参数列表 `>`

常用的参数类型：

- N：接收一个命令，传递命令本身。
- V：与 N 类似，但是传递命令的值。
- n：接收一个凭据表。
- o：与 n 类似，但是对凭据表内的内容进行一次展开。
- x: 与 n 类似, 但是对凭据表内的内容进行递归展开。
- T/F：与 n 类似，用于判断语句中，根据判断结果执行 T/F 代码。
- c：接收一个凭据表，返回以其为名字的命令。
- p：参数列表（#1#2...）

```
\tl_item:Nn
\bool_if:nTF
\tl_put_right:Nx
```

如何使用 LaTeX3?

1. `\usepackage{expl3}`
2. 启用 LaTeX3 语法：`\ExplSyntaxOn`
3. 关闭 LaTeX3 语法：`\ExplSyntaxOff`

最小示例

```
\documentclass{article}
\usepackage{expl3}
\begin{document}
\ExplSyntaxOn
%LaTeX3 代码
\ExplSyntaxOff
\end{document}
```

关于`\ExplSyntaxOn`

- 所有空格及换行都会被忽略
- 下划线（_）和冒号（:）等同于英文字母

```
\ExplSyntaxOn
\cs_set:Npn \my_factorial:n #1 {
    \int_set:Nn \l_tmpa_int {1}
    \seq_clear:N \l_tmpa_seq
    \int_step_inline:nn {#1} {
        \seq_put_right:Nn \l_tmpa_seq {##1}
        \int_set:Nn \l_tmpa_int {\l_tmpa_int * ##1}
    }
    $\seq_use:Nn \l_tmpa_seq {\times} = \int_use:N \l_tmpa_int$
}
\par\my_factorial:n {3}
\par\my_factorial:n {7}
\ExplSyntaxOff
```

$1 \times 2 \times 3 = 6$

$1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 = 5040$

LaTeX3 教程二：变量，函数及基本程序结构

项子越

ziyue.alan.xiang@gmail.com

https://github.com/xziyue/latex3-chinese-video

2021 年 3 月 22 日

- 变量的声明和使用
- 函数的声明和使用
- 循环语句
- 条件语句

声明变量：使用new结尾的函数

- `\bool_new:N`
- `\int_new:N`
- `\seq_new:N`
- `\dim_new:N`
- `\fp_new:N`

- `tl`: 凭据表
- `str`：字符串
- `int`：整型
- `fp`：浮点数
- `seq`：队列
- `dim`：尺度/长度
- `bool`：布尔型

- `N`：接收一个命令，传递命令本身。
- `n`：接收一个凭据表。

设置变量：使用set结尾的函数

- **\int_set:Nn**
- **\dim_set:Nn**
- **\fp_set:Nn**
- **\bool_set_true:N**
- **\bool_set_false:N**

- tl: 凭据表
- str：字符串
- int：整型
- fp：浮点数
- seq：队列
- dim：尺度/长度
- bool：布尔型

- N：接收一个命令，传递命令本身。
- n：接收一个凭据表。

使用变量：使用use结尾的函数

- `\int_use:N`
- `\dim_use:N`
- `\fp_use:N`
- `\tl_use:N`
- `\str_use:N`

- tl: 凭据表
- str：字符串
- int：整型
- fp：浮点数
- seq：队列
- dim：尺度/长度
- bool：布尔型

- N：接收一个命令，传递命令本身。
- n：接收一个凭据表。

声明函数

使用\cs_set:Npn来声明函数。

```
\ExplSyntaxOn
\cs_set:Npn \my_function:n #1 {
    你输入了：#1
}
\par\my_function:n {一}
\par\my_function:n {二}
\ExplSyntaxOff
```

你输入了：一
你输入了：二

查阅函数文档

获取 LaTeX3 文档

- 搜索“CTAN l3kernel”
- 点击“The LATEX3 interfaces”

链接：`http://mirrors.ctan.org/macros/latex/contrib/l3kernel/interface3.pdf`

- 每一个章节对应一个 LaTeX3 库
- 每一个章节内的二级章节对应一系列功能类似的函数

LaTeX3 文档中的函数条目

`\tl_set:Nn` `\tl_set:Nn` ⟨*tl var*⟩ {⟨*tokens*⟩}

`\tl_set:(NV|Nv|No|Nf|Nx|cn|cV|cv|co|cf|cx)`

`\tl_gset:Nn`

`\tl_gset:(NV|Nv|No|Nf|Nx|cn|cV|cv|co|cf|cx)`

Sets ⟨*tl var*⟩ to contain ⟨*tokens*⟩, removing any previous content from the variable.

LaTeX3 文档中预定义的变量（scratch variables）

15.13 Scratch token lists

`\l_tmpa_tl`
`\l_tmpb_tl`

Scratch token lists for local assignment. These are never used by the kernel code, and so are safe for use with any LaTeX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

`\g_tmpa_tl`
`\g_tmpb_tl`

Scratch token lists for global assignment. These are never used by the kernel code, and so are safe for use with any LaTeX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

- 在文档比较庞大时，尽量避免使用这些变量以防止冲突

案例：加法

```
\ExplSyntaxOn
\int_new:N \l_my_tmpa_int
\int_new:N \l_my_tmpb_int
\int_set:Nn \l_my_tmpa_int {200}
\int_set:Nn \l_my_tmpb_int {10}
\int_eval:n {\l_my_tmpa_int + \l_my_tmpb_int}
\ExplSyntaxOff
```

```
210
```

基于整数的循环

```
\ExplSyntaxOn
\int_step_inline:nn {20} {
    #1,~
}
\ExplSyntaxOff
```

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,

基于整数的循环

改变起始数值

```
\ExplSyntaxOn
\int_step_inline:nnn {10} {20} {
    #1,~
}
\ExplSyntaxOff
```

10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,

基于整数的循环

将循环变量保存在凭据表中

```
\ExplSyntaxOn
\int_step_variable:nNn {20} \l_tmpa_tl {
    \tl_use:N \l_tmpa_tl,~
}
\ExplSyntaxOff
```

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,

基于整数的循环

二重循环

```
\ExplSyntaxOn
\int_step_inline:nn {5} {
    \int_step_inline:nn {5} {
        (#1,##1), ~
    }
}
\ExplSyntaxOff
```

(1,1), (1,2), (1,3), (1,4), (1,5), (2,1), (2,2), (2,3), (2,4), (2,5), (3,1), (3,2), (3,3), (3,4), (3,5), (4,1), (4,2), (4,3), (4,4), (4,5), (5,1), (5,2), (5,3), (5,4), (5,5),

案例：$1 + 2 + \ldots + 100 = ?$

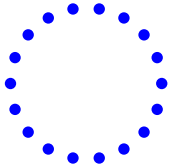
```
\ExplSyntaxOn
\int_set:Nn \l_tmpa_int {0}
\int_step_inline:nn {100} {
    \int_add:Nn \l_tmpa_int {#1}
}
\int_use:N \l_tmpa_int
\ExplSyntaxOff
```

5050

案例：圆上的点

圆的参数方程：$\begin{cases} x = r\cos\theta \\ y = r\sin\theta \end{cases}$

```
\ExplSyntaxOn
\begin{tikzpicture}
\int_step_inline:nnn {0} {17} {
    \fp_set:Nn \l_tmpa_fp {20 * (#1) *
     ↪  \c_one_degree_fp}
    \node[minimum~width=1.5mm, fill=blue,
        draw=none, circle, inner~sep=0pt] at
    (\fp_eval:n {cos(\l_tmpa_fp)},
     \fp_eval:n {sin(\l_tmpa_fp)}) {};
}
\end{tikzpicture}
\ExplSyntaxOff
```



整数判断

```
\ExplSyntaxOn
\cs_set:Npn \my_if_less_than_two:n #1 {
    \int_compare:nNnTF {#1} < {2} {
        \zhnumber{#1} 小于二
    } {
        \zhnumber{#1} 大于等于二
    }
}
\par\my_if_less_than_two:n {1}
\par\my_if_less_than_two:n {2}
\par\my_if_less_than_two:n {3}
\ExplSyntaxOff
```

一小于二
二大于等于二
三大于等于二

整数判断

```
\ExplSyntaxOn
\cs_set:Npn \my_if_less_than_two:n #1 {
    \int_compare:nTF {#1 <= 2} {
        \zhnumber{#1} 小于等于二
    } {
        \zhnumber{#1} 大于二
    }
}
\par\my_if_less_than_two:n {1}
\par\my_if_less_than_two:n {2}
\par\my_if_less_than_two:n {3}
\ExplSyntaxOff
```

一小于等于二
二小于等于二
三大于二

布尔判断

- 使用**\bool_if:nTF**可以进行布尔判断；其表达式参数支持&&, ||, ()等逻辑运算符
- 一般的判断语句还有_p变体，例如**\int_compare_p:n**, **\bool_if_p:n**等。这些函数不是根据判断结果执行分支，而是直接返回判断结果为真或为假
- 这些“判别式”（predicate）可以帮助我们构建复杂的逻辑语句

案例：偶数判断

```
\ExplSyntaxOn
\cs_gset:Npn \my_if_even_p:n #1 {
    \int_compare_p:nNn {\int_mod:nn {#1}{2}} = {0}
}
\cs_set:Npn \my_even_check:n #1 {
    \bool_if:nTF { \my_if_even_p:n {#1}} {
        \zhnumber{#1}是偶数
    } {
        \zhnumber{#1}是奇数
    }
}
\par \my_even_check:n{1}
\par \my_even_check:n{2}
\par \my_even_check:n{3}
\ExplSyntaxOn
```

一是奇数
二是偶数
三是奇数

案例：双偶数判断

```
\ExplSyntaxOn
\cs_set:Npn \my_double_even_check:nn #1#2 {
    \bool_if:nTF { \my_if_even_p:n {#1} && \my_if_even_p:n {#2} } {
        \zhnumber{#1}和\zhnumber{#2}都是偶数
    } {
        \zhnumber{#1}和\zhnumber{#2}不都是偶数
    }
}
\par \my_double_even_check:nn {1} {3}
\par \my_double_even_check:nn {1} {2}
\par \my_double_even_check:nn {2} {4}
\ExplSyntaxOn
```

一和三不都是偶数
一和二不都是偶数
二和四都是偶数

条件循环语句

诸如\int_do_while:nNnn, \bool_do_while:nn等语句每一次循环就进行一次判断，直到判断为假。

```
\ExplSyntaxOn
\int_set:Nn \l_tmpa_int {1}
\int_set:Nn \l_tmpb_int {0}
\int_do_while:nNnn {\l_tmpa_int} < {101} {
    \int_add:Nn \l_tmpb_int {\l_tmpa_int}
    \int_incr:N \l_tmpa_int
}
\int_use:N \l_tmpb_int
\ExplSyntaxOff
```

5050

条件循环语句

诸如\int_do_until:nNnn, \bool_do_until:nn等语句每一次循环就进行一次判断，直到判断为真。

```
\ExplSyntaxOn
\int_set:Nn \l_tmpa_int {1}
\int_set:Nn \l_tmpb_int {0}
\int_do_until:nNnn {\l_tmpa_int} > {100} {
    \int_add:Nn \l_tmpb_int {\l_tmpa_int}
    \int_incr:N \l_tmpa_int
}
\int_use:N \l_tmpb_int
\ExplSyntaxOff
```

5050

LaTeX3 教程三：宏展开

项子越

ziyue.alan.xiang@gmail.com

https://github.com/xziyue/latex3-chinese-video

2021 年 7 月 9 日

控制宏展开的意义

在定义命令的时候，LaTeX 把函数体的原文保存在定义里。在每次调用命令时，其所使用的变量值可能改变。

```
\newcommand{\myvar}{一}
\newcommand{\mycmd}{%
    值为：\myvar%
}
\par\mycmd
\renewcommand{\myvar}{二}
\par\mycmd
```

值为：一
值为：二

利用宏展开技巧，我们可以把**\myvar**的值写入**\mycmd**的定义中，从而使用每次调用**\mycmd**的结果一致。

控制宏展开的意义

利用**\uppercase**命令可以将英文字符变成大写。

```
1 \uppercase{abcde}
```

ABCDE

但是**\uppercase**只会将它所遇到的字符变成大写，它所遇到的变量中的字符不会变成大写。

```
1 \newcommand{\myvar}{abcde}
2 \uppercase{abcde\myvar}
```

ABCDEabcde

利用宏展开技巧，我们可以让**\uppercase**处理命令中的字符。

复习：各种参数类型

- N：接收一个命令，传递命令本身。
- V：与 N 类似，但是传递命令的值。
- n：接收一个凭据表。
- o：与 n 类似，但是对凭据表内的内容进行一次展开。
- x: 与 n 类似, 但是对凭据表内的内容进行递归展开。
- T/F：与 n 类似，用于判断语句中，根据判断结果执行 T/F 代码。
- c：接收一个凭据表，返回以其为名字的命令。
- p：参数列表（#1#2...）

法一：选择正确的函数变体

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl {测试}
\tl_set:Nn \l_tmpb_tl {\l_tmpa_tl}
\cs_meaning:N \l_tmpb_tl
\ExplSyntaxOff
```

macro:->\l_tmpa_tl

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl {测试}
\tl_set:NV \l_tmpb_tl \l_tmpa_tl
\cs_meaning:N \l_tmpb_tl
\ExplSyntaxOff
```

macro:-> 测试

法一：选择正确的函数变体

```
\ExplSyntaxOn
\newcommand{\myvar}{再测试}
\tl_set:Nn \l_tmpa_tl {测试\myvar}
\tl_set:NV \l_tmpb_tl \l_tmpa_tl
\cs_meaning:N \l_tmpb_tl
\ExplSyntaxOff
```

macro:-> 测试\myvar

```
\ExplSyntaxOn
\newcommand{\myvar}{再测试}
\tl_set:Nn \l_tmpa_tl {测试\myvar}
\tl_set:Nx \l_tmpb_tl {\l_tmpa_tl}
\cs_meaning:N \l_tmpb_tl
\ExplSyntaxOff
```

macro:-> 测试再测试

法二：使用\exp_args:函数

- 有时候 LaTeX3 并没有提供我们想要的函数变体
- 有时我们想控制传统 LaTeX 命令的展开

这时我们可以使用**\exp_args:**系列函数。

\exp_args:NABCD

- N是我们想控制展开的命令
- A是第一个参数要展开的类型；B是第二个参数要展开的类型；C是第三个参数要展开的类型……

法二：使用\exp_args:函数

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl {测试}
\tl_set:Nn \l_tmpb_tl {\l_tmpa_tl}
\cs_meaning:N \l_tmpb_tl
\ExplSyntaxOff
```

macro:->\l_tmpa_tl

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl {测试}
\exp_args:NNV \tl_set:Nn \l_tmpb_tl
  ↪ \l_tmpa_tl
\cs_meaning:N \l_tmpb_tl
\ExplSyntaxOff
```

macro:-> 测试

法二：使用\exp_args:函数

```
\ExplSyntaxOn
\newcommand{\myvar}{再测试}
\tl_set:Nn \l_tmpa_tl {测试\myvar}
\exp_args:NNV \tl_set:Nn \l_tmpb_tl
  ↪ \l_tmpa_tl
\cs_meaning:N \l_tmpb_tl
\ExplSyntaxOff
```

macro:-> 测试\myvar

```
\ExplSyntaxOn
\newcommand{\myvar}{再测试}
\tl_set:Nn \l_tmpa_tl {测试\myvar}
\exp_args:NNx \tl_set:Nn \l_tmpb_tl
  ↪ {\l_tmpa_tl}
\cs_meaning:N \l_tmpb_tl
\ExplSyntaxOff
```

macro:-> 测试再测试

法二：使用\exp_args:函数

```
\ExplSyntaxOn
\newcommand{\myvar}{abcde}
\par\uppercase{abcde\myvar}
\par\exp_args:Nx\uppercase{abcde\myvar}
\ExplSyntaxOff
```

ABCDEabcde
ABCDEABCDE

法二：使用\exp_args:函数

\exp_args:函数可以用来部分展开参数（仅控制前 N 个参数的展开）

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl {mycmdname}
\cs_set:cpn {\l_tmpa_tl} {
    调用我的函数
}
\mycmdname
\ExplSyntaxOff
```

调用我的函数

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl {mycmdname}
\exp_args:Nc \newcommand{\l_tmpa_tl}{
    调用我的函数
}
\mycmdname
\ExplSyntaxOff
```

调用我的函数

法三：使用\cs_generate_variant:Nn 函数

- **\cs_generate_variant:Nn**函数可以用于生成新的函数变体
- 用户自定义的函数所接收的参数类型一般是N或n；利用**\cs_generate_variant:Nn**，我们可以把这些函数的参数类型变成其它类型
- **\cs_generate_variant:Nn**函数只可用于使用 LaTeX3 命名法的函数

法三：使用\cs_generate_variant:Nn 函数

```
\ExplSyntaxOn
\cs_gset:Npn \my_func:nn #1#2 {
    \tl_set:Nn \l_tmpa_tl {#1#2}
    \cs_meaning:N \l_tmpa_tl
}
\newcommand{\myvar}{变量内容}
\my_func:nn {输入} {\myvar}
\ExplSyntaxOff
```

macro:-> 输入\myvar

```
\ExplSyntaxOn
\newcommand{\myvar}{变量内容}
\cs_generate_variant:Nn \my_func:nn {nV}
\my_func:nV {输入} \myvar
\ExplSyntaxOff
```

macro:-> 输入变量内容

法三：使用\cs_generate_variant:Nn 函数

```
\ExplSyntaxOn
\newcommand{\myvarvar}{内容}
\newcommand{\myvar}{变量内容\myvarvar}
\par\my_func:nV {输入} \myvar
\cs_generate_variant:Nn \my_func:nn {nx}
\par\my_func:nx {输入} {\myvar}
\ExplSyntaxOff
```

---

macro:-> 输入变量内容\myvarvar
macro:-> 输入变量内容内容

法三：使用\cs_generate_variant:Nn 函数

当找不到合适的**\exp_args:**函数时，我们可以使用**\cs_set_eq:NN**把传统 LaTeX 函数变成 LaTeX3 命名法的函数，然后再使用**\cs_generate_variant:Nn**来生成新的变体。

```
1  \ExplSyntaxOn
2  \newcommand*{\mycmd}{abcd}
3  \par\cs_meaning:N \mycmd
4  \newcommand{\myvar}{mycmd}
5  \newcommand{\myvarvar}{efgh}
6  \cs_set_eq:NN \apptocmd:Nnnn \apptocmd
7  \cs_generate_variant:Nn \apptocmd:Nnnn
   ↪ {cVnn}
8  \apptocmd:cVnn {\myvar} \myvarvar {} {}
9  \par\cs_meaning:N \mycmd
10 \ExplSyntaxOff
```

```
macro:->abcd
macro:->abcdefgh
```

页面标注系统

假设需要给用户设计一个命令`\pagenote{...}`，该命令允许用户进行一些标注，并且把标注的内容和页号记录下来，最后用`\showpagenote`命令输出。

例如：用户在第 9 页写下：`\pagenote{我的笔记}`，那么`\showpagenote`就会输出：

第 9 页：我的笔记

页面标注系统

```
\ExplSyntaxOn
\tl_new:N \g_my_pagenote_tl
\cs_gset:Npn \pagenote #1 {
    \tl_gput_right:Nn \g_my_pagenote_tl {
        {第\thepage 页： #1}
    }
}
\cs_gset:Npn \showpagenote {
    \int_step_inline:nn {\tl_count:N \g_my_pagenote_tl} {
        \par\tl_item:Nn \g_my_pagenote_tl {##1}
    }
}
\cs_gset:Npn \clearpagenote {
    \tl_gclear:N \g_my_pagenote_tl
}
\ExplSyntaxOff
```

页面标注系统

在第 18 页我写下：

```
1 \pagenote{笔记一}
```

页面标注系统

在第 19 页我写下：

```
1 \pagenote{笔记二}
```

页面标注系统

在第 20 页我想输出所有标注：

```
\showpagenote
```

第 20 页：笔记一
第 20 页：笔记二

为什么页号是错的？

```
\ExplSyntaxOn
\cs_meaning:N \g_my_pagenote_tl
\clearpagenote
\ExplSyntaxOff
```

macro:->{第\thepage 页：笔记一}{第\thepage 页：笔记二}

因为**\thepage**没有被展开！

页面标注系统

改进\pagenote的实现：

```
\ExplSyntaxOn
\cs_gset:Npn \pagenote #1 {
    \tl_gput_right:Nx \g_my_pagenote_tl {
        {第\thepage 页： #1}
    }
}
\ExplSyntaxOff
```

页面标注系统

在第 22 页我写下：

```
1 \pagenote{笔记一}
```

页面标注系统

在第 23 页我写下：

```
1 \pagenote{笔记二}
```

页面标注系统

在第 24 页我想输出所有标注：

```
\showpagenote
\clearpagenote
```

第 22 页：笔记一
第 23 页：笔记二

这样的实现有什么问题？
假设用户在他们的笔记内有数学公式（`$\operatorname{myop}(a, b)$`），x展开会在**`\operatorname`**接收参数之前先将其函数体展开，最终导致文档无法编译。

页面标注系统

解决方案：使用**\exp_not:N**或**\exp_not:n**。前者将会避免展开下一个命令; 后者会避免展开下一个组。
改进后的实现：

```
\ExplSyntaxOn
\cs_gset:Npn \pagenote #1 {
    \tl_gput_right:Nx \g_my_pagenote_tl {
        {第\thepage 页:  \exp_not:n {#1}}
    }
}
\ExplSyntaxOff
```

页面标注系统

在第 26 页我写下：

```
1 \pagenote{笔记一}
```

页面标注系统

在第 27 页我写下：

```
1 \pagenote{笔记二}
```

页面标注系统

在第 28 页我写下：

```
1 \pagenote{一条重要公式: $x = \operatorname{sgn}(\frac{a}{b})$}
```

页面标注系统

在第 29 页我想输出所有标注：

```
\showpagenote
```

第 26 页：笔记一
第 27 页：笔记二
第 28 页：一条重要公式：$x = \mathrm{sgn}(\frac{a}{b})$

归并排序

现在我们利用 LaTeX 来实现一个经典的递归算法：归并排序

伪代码：

```
MergeSort(arr[], l,  r)
If r > l
     1. Find the middle point to divide the array into two halves:
             middle m = l+ (r-l)/2
     2. Call mergeSort for first half:
             Call mergeSort(arr, l, m)
     3. Call mergeSort for second half:
             Call mergeSort(arr, m+1, r)
     4. Merge the two halves sorted in step 2 and 3:
             Call merge(arr, l, m, r)
```

归并排序

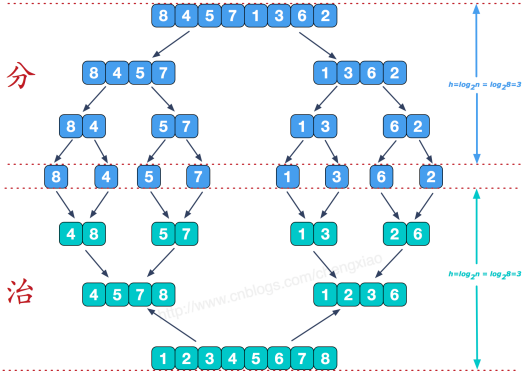


图: 算法示意图（https://www.cnblogs.com/chengxiao/p/6194356.html）

归并排序

需要利用的几个工具与知识点：

- 常用的set赋值函数仅影响组内的值，把函数体封装在组内可以使每一层的函数拥有自己独立的局部变量（类似于其它编程语言中的栈）
- **\tl_head:n** 返回凭据表的头一个元素
- **\tl_tail:n** 返回凭据表的尾部元素（去除第一个元素）
- **\tl_range:nnn** 返回凭据表的一个区间内的所有元素
- LaTeX 中的递归函数一般无法直接返回值，我们可以把要返回的值存在一个全局变量中

归并排序

```
\ExplSyntaxOn
\tl_new:N \g_merge_result_tl
\cs_gset:Npn \my_merge:nn #1#2 {
    \group_begin:
        \bool_set_true:N \l_tmpa_bool
        \tl_if_empty:nT {#1} { \bool_set_false:N \l_tmpa_bool
            \tl_gput_right:Nn \g_merge_result_tl {#2} }
        \tl_if_empty:nT {#2} { \bool_set_false:N \l_tmpa_bool
            \tl_gput_right:Nn \g_merge_result_tl {#1} }
        \bool_if:NT \l_tmpa_bool {
            \int_compare:nNnTF {\tl_head:n {#1}} < {\tl_head:n {#2}}
                { \tl_gput_right:Nx \g_merge_result_tl {{\tl_head:n {#1}}}
                \exp_args:Nx \my_merge:nn {\tl_tail:n {#1}} {#2} }
                { \tl_gput_right:Nx \g_merge_result_tl {{\tl_head:n {#2}}}
                \exp_args:Nnx \my_merge:nn {#1} {\tl_tail:n {#2}} }
        }
    \group_end:
}
\ExplSyntaxOff
```

归并排序

```
\ExplSyntaxOn
\tl_new:N \g_merge_sort_result_tl
\cs_gset:Npn \my_merge_sort:n #1 {
    \iow_term:n {#1}
    \group_begin:
        \int_compare:nNnTF {\tl_count:n {#1}} > {1} {
            \int_set:Nn \l_tmpa_int {\int_div_truncate:nn {\tl_count:n {#1}}{2}}
            \exp_args:Nx \my_merge_sort:n { \tl_range:nnn {#1} {1} {\l_tmpa_int} }
            \tl_set_eq:NN \l_tmpa_tl \g_merge_sort_result_tl
            \exp_args:Nx \my_merge_sort:n { \tl_range:nnn {#1} {\l_tmpa_int + 1} {\tl_count:n
             ↪  {#1}} }
            \tl_clear:N \g_merge_result_tl
            \exp_args:NVV \my_merge:nn \l_tmpa_tl \g_merge_sort_result_tl
            \tl_gset_eq:NN \g_merge_sort_result_tl \g_merge_result_tl
        } { \tl_gset:Nn \g_merge_sort_result_tl {#1} }
    \group_end:
}
\ExplSyntaxOff
```

归并排序

```
\ExplSyntaxOn
\my_merge_sort:n {{1}{3}{5}{2}{4}{6}}
\par\cs_meaning:N \g_merge_sort_result_tl
\my_merge_sort:n {{1}{2}{1}{2}{1}{2}}
\par\cs_meaning:N \g_merge_sort_result_tl
\my_merge_sort:n {{9}{8}{7}{6}{5}{4}{3}{2}{1}}
\par\cs_meaning:N \g_merge_sort_result_tl
\my_merge_sort:n {{1}}
\par\cs_meaning:N \g_merge_sort_result_tl
\ExplSyntaxOff
```

macro:->{1}{2}{3}{4}{5}{6}
macro:->{1}{1}{1}{2}{2}{2}
macro:->{1}{2}{3}{4}{5}{6}{7}{8}{9}
macro:->{1}

LaTeX3 教程四：常用库

项子越

ziyue.alan.xiang@gmail.com

https://github.com/xziyue/latex3-chinese-video

2021 年 12 月 17 日

凭据表（token list)

- 用于存储文档，命令等内容
- 提供多种使用及修改的接口
- 可用于存储参数供其它函数调用
- 主要缺点：无法存储换行符

凭据表的基本操作

对凭据表变量赋值

```
\ExplSyntaxOn
\tl_gclear:N \g_tmpa_tl % 清除旧值
\tl_gset:Nn \g_tmpa_tl {新值}
\cs_meaning:N \g_tmpa_tl
\ExplSyntaxOff
```

macro:-> 新值

在左侧追加

```
\ExplSyntaxOn
\tl_gput_left:Nn \g_tmpa_tl {一个}
\cs_meaning:N \g_tmpa_tl
\ExplSyntaxOff
```

macro:-> 一个新值

在右侧追加

```
\ExplSyntaxOn
\tl_gput_right:Nn \g_tmpa_tl {被定义}
\cs_meaning:N \g_tmpa_tl
\ExplSyntaxOff
```

macro:-> 一个新值被定义

凭据表的基本操作

遍历

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl {ab{cd}\P e}
\tl_map_inline:Nn \l_tmpa_tl {
    \par 元素：#1
}
\ExplSyntaxOff
```

元素：a
元素：b
元素：cd
元素：¶
元素：e

凭据表的基本操作

按元素序号访问

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl {ab{cd}\P e}
\par\tl_count:N \l_tmpa_tl
\par\tl_item:Nn \l_tmpa_tl {1}
\par\tl_item:Nn \l_tmpa_tl {2}
\par\tl_item:Nn \l_tmpa_tl {3}
\par\tl_item:Nn \l_tmpa_tl {4}
\par\tl_item:Nn \l_tmpa_tl {5}
\ExplSyntaxOff
```

```
5
a
b
cd
¶
e
```

凭据表的基本操作

取头/取尾

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl {abcde}
\par\tl_head:N \l_tmpa_tl
\par\tl_tail:N \l_tmpa_tl
\ExplSyntaxOff
```

a
bcde

遍历

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl {ab{cd}\P e}
\tl_map_inline:Nn \l_tmpa_tl {
    \par 元素：#1
}
\ExplSyntaxOff
```

元素：a
元素：b
元素：cd
元素：¶
元素：e

凭据表与字符串

字符串库 (string) 与凭据表十分相似。两者的区别主要在于字符串中所存储的所有内容都会以原样输出。

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl
  ↪ {$\frac{\alpha}{\beta}$}
\tl_use:N \l_tmpa_tl
\ExplSyntaxOff
```

$\frac{\alpha}{\beta}$

```
\ExplSyntaxOn
\str_set:Nn \l_tmpa_str
  ↪ {$\frac{\alpha}{\beta}$}
\str_use:N \l_tmpa_str
\ExplSyntaxOff
```

$\frac {\alpha }{\beta }$

中英文数字转换

```
\ExplSyntaxOn
\tl_new:N \g_doc_chn_num_tl
\tl_gset:Nn \g_doc_chn_num_tl {一二三四五六七八九}
\cs_gset:Npn \doc_arabic_to_chn_a:n #1 {
    \tl_item:Nn \g_doc_chn_num_tl {#1}
}
\par\doc_arabic_to_chn_a:n {1}
\par\doc_arabic_to_chn_a:n {3}
\par\doc_arabic_to_chn_a:n {5}
\ExplSyntaxOff
```

一
三
五

中英文数字转换

```
\ExplSyntaxOn
\tl_new:N \g_doc_chn_dec_tl
\tl_gset:Nn \g_doc_chn_dec_tl {十百千万}
\cs_gset:Npn \doc_arabic_to_chn:n #1 {
  \tl_set_eq:NN \l_tmpa_tl \g_doc_chn_dec_tl
  \tl_clear:N \l_tmpb_tl
  \doc_arabic_to_chn_b:nN {#1} \l_tmpa_tl      \tl_use:N \l_tmpb_tl
}
\cs_gset:Npn \doc_arabic_to_chn_b:nN #1#2 {
  \int_set:Nn \l_tmpa_int {\int_div_truncate:nn {#1}{10}}
  \int_set:Nn \l_tmpb_int {\int_mod:nn {#1}{10}}
  \int_compare:nNnT {\l_tmpb_int} > {0} {
      \tl_put_left:Nx \l_tmpb_tl {\doc_arabic_to_chn_a:n {\l_tmpb_int}}
  }
  \int_compare:nNnT {\l_tmpa_int} > {0} {
      \tl_put_left:Nx \l_tmpb_tl {\tl_head:N #2}
      \tl_set:Nx #2 {\tl_tail:N #2}
      \exp_args:Nx \doc_arabic_to_chn_b:nN {\int_use:N \l_tmpa_int} #2
  }
}
\ExplSyntaxOff
```

中英文数字转换

```
\ExplSyntaxOn
\par\doc_arabic_to_chn:n {5}
\par\doc_arabic_to_chn:n {15}
\par\doc_arabic_to_chn:n {615}
\par\doc_arabic_to_chn:n {4615}
\par\doc_arabic_to_chn:n {84615}
\par\doc_arabic_to_chn:n {20}
\par\doc_arabic_to_chn:n {320}
\ExplSyntaxOff
```

五
一十五
六百一十五
四千六百一十五
八万四千六百一十五
二十
三百二十

接收文本参数的格式化

假设我们想设计一个命令，它能根据一个参数来选择如何格式化文字。比如`\fmt{cu}{abc}`会将 abc 加粗；`\fmt{xie}{abc}`会使用斜体；`\fmt{xian}{abc}`会使用下划线等。

```
\ExplSyntaxOn
\newcommand{\fmt}[2]{
    \str_if_eq:nnT {#1} {cu} {\textbf{#2}}
    \str_if_eq:nnT {#1} {xie} {\textit{#2}}
    \str_if_eq:nnT {#1} {xian}
     ↪ {\underline{#2}}
}
\par\fmt{cu}{abc}
\par\fmt{xie}{abc}
\par\fmt{xian}{abc}
\ExplSyntaxOff
```

abc
abc
abc

`\tl_if_eq:`不仅比较字符，还会比较字符的类别码 (category code)。因此可能出现字符看上去相等但是程序判断为不等的情况。

接收文本参数的格式化

```
\ExplSyntaxOn
\newcommand{\fmt}[2]{
    \str_case:nnTF {#1} {
        {cu} {\textbf{#2}}
        {xie} {\textit{#2}}
        {xian} {\underline{#2}}
    }{
        \textcolor{green}{\ (good)}
    }{
        \textcolor{red}{#2\ (bad)}
    }
}
\par\fmt{cu}{abc}
\par\fmt{xie}{abc}
\par\fmt{xian}{abc}
\par\fmt{123}{abc}
\ExplSyntaxOff
```

abc (good)
abc (good)
abc (good)
abc (bad)

序列表 (sequence)

- 用于存储并管理一系列的元素
- 可以按序号访问，也可以只访问首元素或尾元素
- 可以把多个元素连接起来

序列表的基本操作

在右侧追加

```
\ExplSyntaxOn
\seq_gclear:N \g_tmpa_seq
\seq_gput_right:Nn \g_tmpa_seq {一}
\seq_gput_right:Nn \g_tmpa_seq {个}
\seq_gput_right:Nn \g_tmpa_seq {元素}
\par\seq_use:Nn  \g_tmpa_seq {,~}
\ExplSyntaxOff
```

一, 个, 元素

在左侧追加

```
\ExplSyntaxOn
\seq_gput_left:Nn \g_tmpa_seq {不只}
\par\seq_use:Nn  \g_tmpa_seq {,~}
\ExplSyntaxOff
```

不只, 一, 个, 元素

序列表的基本操作

获取最左侧元素

```
\ExplSyntaxOn
\seq_get_left:NN \g_tmpa_seq \l_tmpa_tl
\tl_use:N \l_tmpa_tl
\ExplSyntaxOff
```

不只

获取最右侧元素

```
\ExplSyntaxOn
\seq_get_right:NN \g_tmpa_seq \l_tmpa_tl
\tl_use:N \l_tmpa_tl
\ExplSyntaxOff
```

元素

\seq_pop_left: 以及 **\seq_pop_right:**的功能类似，但是会从序列中移除元素。

序列表的基本操作

遍历

```
\ExplSyntaxOn
\seq_map_inline:Nn \g_tmpa_seq {
    \par 元素：#1
}
\ExplSyntaxOff
```

元素：不只
元素：一
元素：个
元素：元素

序列表的基本操作

按序号访问

```
\ExplSyntaxOn
\par\seq_item:Nn \g_tmpa_seq {1}
\par\seq_item:Nn \g_tmpa_seq {3}
\ExplSyntaxOff
```

不只
个

序列表的本质是凭据表

```
\ExplSyntaxOn
\cs_meaning:N \g_tmpa_seq
\ExplSyntaxOff
```

macro:->\s__seq __seq_item:n {不只}__seq_item:n {一}__seq_item:n {个}__seq_item:n {元素}

序列表的基本操作

连接序列表中的元素

```
\ExplSyntaxOn
\par\seq_use:Nn \g_tmpa_seq {, }
\par\seq_use:Nn \g_tmpa_seq {——}
\ExplSyntaxOff
```

不只，一，个，元素
不只——一——个——元素

制作正弦表

我们希望制作如下表格：

sin 0° = 0	sin 1° = 0.01745240643728352	sin 2° = 0.03489949670250098
sin 3° = 0.05233595624294385	sin 4° = 0.06975647374412532	sin 5° = 0.08715574274765819
sin 6° = 0.1045284632676535	sin 7° = 0.1218693434051475	sin 8° = 0.1391731009600655
sin 9° = 0.1564344650402309	sin 10° = 0.1736481776669304	sin 11° = 0.1908089953765449
sin 12° = 0.2079116908177594	sin 13° = 0.2249510543438651	sin 14° = 0.2419218955996678
sin 15° = 0.2588190451025208	sin 16° = 0.2756373558169993	sin 17° = 0.2923717047227368

制作正弦表

```
\ExplSyntaxOn
\seq_gclear:N \g_tmpa_seq
\int_step_inline:nnn {0} {18} {
    \seq_put_right:Nn \l_tmpa_seq {
        $\sin #1^\circ = \fp_eval:n {sin(#1 * \c_one_degree_fp)}$
    }
    \int_compare:nNnT {#1} > {0} {
        \int_compare:nNnT {\int_mod:nn {#1 + 1} {3}} = {0} {
            \seq_gput_right:Nx \g_tmpa_seq {
                \seq_use:Nn \l_tmpa_seq {&}
            }
            \seq_clear:N \l_tmpa_seq
        }
    }
}
\ExplSyntaxOff
```

制作正弦表

```
\ExplSyntaxOn
\tiny
\centering
\begin{tabular}{lll}
\toprule
\seq_use:Nn \g_tmpa_seq {\\}
\\ \bottomrule
\end{tabular}
\par
\ExplSyntaxOff
```

sin 0° = 0	sin 1° = 0.01745240643728352	sin 2° = 0.03489949670250098
sin 3° = 0.05233595624294385	sin 4° = 0.06975647374412532	sin 5° = 0.08715574274765819
sin 6° = 0.1045284632676535	sin 7° = 0.1218693434051475	sin 8° = 0.1391731009600655
sin 9° = 0.1564344650402309	sin 10° = 0.1736481776669304	sin 11° = 0.1908089953765449
sin 12° = 0.2079116908177594	sin 13° = 0.2249510543438651	sin 14° = 0.2419218955996678
sin 15° = 0.2588190451025208	sin 16° = 0.2756373558169993	sin 17° = 0.2923717047227368

随机列表

设计两个命令：

- **\AddToList**：把元素加入列表
- **\ShowList**：以随机顺序输出列表中的元素

```
\ExplSyntaxOn
\seq_gclear:N \g_tmpa_seq
\cs_gset:Npn \AddToList #1 {
    \seq_gput_right:Nn \g_tmpa_seq {#1}
}
\ExplSyntaxOff
```

随机列表

```
\ExplSyntaxOn
\cs_gset:Npn \ShowList {
    \seq_shuffle:N \g_tmpa_seq
    \begin{itemize}
        \seq_map_inline:Nn \g_tmpa_seq {
            \item ##1
        }
    \end{itemize}
}
\ExplSyntaxOff
```

随机列表

```
\AddToList{两}
\AddToList{只}
\AddToList{老虎}
\AddToList{跑}
\AddToList{得}
\AddToList{快}
\ShowList
```

- 快
- 跑
- 只
- 两
- 老虎
- 得

逗号分隔表 (comma-separated list)

- 接口与序列基本相同
- 区别：可以用逗号分隔的内容来赋值

```
\ExplSyntaxOn
\clist_set:Nn \l_tmpa_clist {a,b,cd,\P,e}
\clist_map_inline:Nn \l_tmpa_clist {
    \par 元素：#1
}
\ExplSyntaxOff
```

元素：a
元素：b
元素：cd
元素：¶
元素：e

逗号分隔表 (comma-separated list)

按序号访问

```
\ExplSyntaxOn
\clist_set:Nn \l_tmpa_clist {a,b,cd,\P,e}
\par\clist_item:Nn \l_tmpa_clist {1}
\par\clist_item:Nn \l_tmpa_clist {2}
\par\clist_item:Nn \l_tmpa_clist {3}
\par\clist_item:Nn \l_tmpa_clist {4}
\par\clist_item:Nn \l_tmpa_clist {5}
\ExplSyntaxOff
```

a
b
cd
¶
e

接收文本参数的格式化

```
\ExplSyntaxOn
\newcommand{\fmt}[2]{
    \group_begin:
    \clist_set:Nn \l_tmpa_clist {#1}
    \clist_map_inline:Nn \l_tmpa_clist {
        \str_case:nn {##1} {
            {cu} {\bfseries}
            {xie} {\itshape}
            {hong} {\color{red}}
        }
    } #2
    \group_end:
}
\ExplSyntaxOff
\par\fmt{cu}{abc}
\par\fmt{xie}{abc}
\par\fmt{hong}{abc}
\par\fmt{cu,hong}{abc}
\par\fmt{xie,hong}{abc}
\par\fmt{cu,xie,hong}{abc}
```

abc

abc

abc

abc

abc

abc

属性表 (property list)

- 与 Python 中的字典（dict）类似，提供键—值访问

属性表的基本操作

设置值

```
\ExplSyntaxOn
\prop_gput:Nnn \g_tmpa_prop {key1} {val1}
\prop_gput:Nnn \g_tmpa_prop {key2} {val2}
\ExplSyntaxOff
```

取回值

```
\ExplSyntaxOn
\par\prop_item:Nn \g_tmpa_prop {key1}
\par\prop_item:Nn \g_tmpa_prop {key2}
\ExplSyntaxOff
```

val1
val2

取回值的函数还有**\prop_get:**及**\prop_pop**等。

属性表的基本操作

遍历

```
\ExplSyntaxOn
\prop_map_inline:Nn \g_tmpa_prop {
    \par 键：#1 \quad 值：#2
}
\ExplSyntaxOff
```

键：key1　值：val1
键：key2　值：val2

遍历输出的顺序是未定义的。

属性表的基本操作

属性表也可通过类似逗号分隔表的方式初始化。

```
\ExplSyntaxOn
\prop_set_from_keyval:Nn \l_tmpa_prop {
    1= 一,
    2= 二,
    3= 三
}

\prop_map_inline:Nn \l_tmpa_prop {
    \par 键：#1 \quad 值：#2
}
\ExplSyntaxOff
```

键：1　值：一
键：2　值：二
键：3　值：三

缩写管理

设计两个命令：

- **\AddAcronym**：用于添加缩写到系统中
- **\Acro**：用于取回缩写的全称

```
\ExplSyntaxOn
\prop_new:N \g_doc_acro_prop
\newcommand{\AddAcronym}[2]{
    \prop_gput:Nnn \g_doc_acro_prop {#1} {#2}
}
\newcommand{\Acro}[1]{
    \textbf{\prop_item:Nn \g_doc_acro_prop
    ↪ {#1}}
}
\ExplSyntaxOff

\AddAcronym{gmm}{高斯混合模型}
\AddAcronym{em}{期望最大化算法}

使用\Acro{em}来优化\Acro{gmm}
```

使用**期望最大化算法**来优化**高斯混合模型**