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

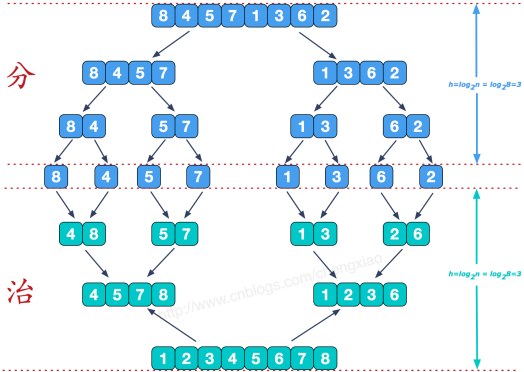

图: 算法示意图（https://www.cnblogs.com/chengxiao/p/6194356.html）

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}