

# latex.ltx リーディング

## 第3回資料

東大 TeX 愛好会

2015年5月8日 (2015年5月18日版)

### 1 L<sup>A</sup>T<sub>E</sub>X 流カウンターの親子関係

前は「L<sup>A</sup>T<sub>E</sub>X 流カウンター入門」ということで、L<sup>A</sup>T<sub>E</sub>X 流カウンターの基本的な3機能について検討した。今回は、その4つ目の機能である「L<sup>A</sup>T<sub>E</sub>X 流カウンターの親子関係」について考える。

#### 1.1 親カウンターと子カウンターの挙動

はじめに、そもそも「親子関係」とはどういうものなのか確認しておく。親子関係(従属関係)は子カウンターの定義時に設定され、以後親カウンターの値が1増えると、子カウンターの値は0にリセットされるようになる。この仕組みが利用されている身近な例として section カウンターと subsection カウンターがある。また、一つの親カウンターに複数の子カウンターが存在する場合もあり、この場合親カウンターの値が1増えると、すべての子カウンターの値が0にリセットされる必要がある。

なお、以下では親カウンターを `LaTeXcounterP`、子カウンターを `LaTeXcounterC` で一般に表現するものとする。

#### 1.2 `\newcounter` の仕掛け

実際に「親カウンターの値が1増えると、子カウンターの値が0にリセットされる」仕組みを考える前に、`\newcounter` 命令で行われるそのための準備(仕掛け)を見ていく。前回の資料にも登場したが、ここでもう一度 `\newcounter` の定義箇所を掲載する。

---

```
1819 \def\newcounter#1{%
1820   \expandafter\@ifdefinable \csname c@#1\endcsname
1821     {\@definecounter{#1}}%
1822   \@ifnextchar[{\@newctr{#1}}{}}
```

---

このうち今回取り扱うのは最後の1行である。すなわち、`\@ifnextchar` の分岐によって、「親カウンターが指定されたとき」のみ実行される `\@newctr` の中身を覗き見ることになる。

---

```
1824 \def\@newctr#1[#2]{%
1825   \@ifundefined{c@#2}{\@nocounterr{#2}}{\@addtoreset{#1}{#2}}
```

---

説明が前回と重複する箇所については簡単に済ませるが、1825行目の前半部分は「指定された親カウンターが本当に存在しているかを調べ、存在しなかった場合はエラーを吐く」という動作をする。したがって、さらに追跡を要するのは `\@addtoreset` 命令のみとなる。

---

```
1842 \def\@addtoreset#1#2{\expandafter\@cons\csname cl@#2\endcsname {#1}}
```

---

ここで `\cl@...` という前回は登場した形が気になる。これは、可変命令を含むリセットリストである。ここでは `\cl@LaTeXcounterP` となっていることから、親カウンターの(もつ)リセットリストであることがわか

る。さて、「可変命令を含むリセットリスト」が果たして何なのかを理解するためにも `latex.ltx` の読解を続けたい。

---

```
579 \def\@cons#1#2{\begingroup\let\@elt\relax\xdef#1{#1\@elt #2}\endgroup}
580 \def\@car#1#2\@nil{#1}
581 \def\@cdr#1#2\@nil{#2}
```

---

今登場しているのは `\@cons` だけだが、ついでなので `\@car` と `\@cdr` の定義もまとめて引用した。`\@car` と `\@cdr` はそれぞれリストの「最初の要素」と「最初以外の要素」\*1を取り出す命令である（第1回資料の3ページも参照のこと）。

一方の `\@cons` はリストに新たな（可変命令付の）要素を加えるための命令である。その使用方法を見やすくするため、先ほど登場した `\@addtoreset` の定義の中身を見やすい形で書き直してみる。

```
\@cons\cl@LaTeXcounterP{\LaTeXcounterC}
```

続いて、ここから1回展開した形を書き下す（見やすいよう適宜改行とインデントを加えた）。

```
\begingroup
  \let\@elt\relax
  \xdef\cl@LaTeXcounterP{\cl@LaTeXcounterP\@elt\LaTeXcounterC}
\endgroup
```

指摘するまでもなく気付くことかもしれないが、この `\@elt` が可変命令である。先に種明かしをしてしまうと、通常は `\@elt` には `\relax` が代入されており（上のコードの2行目）、リセットリストに含まれるカウンターをリセットする際にこの `\@elt` にカウンター値を0にリセットするための命令が代入され、実行される。

後半に登場する `\xdef` はもちろん `\global\edef` の短縮形である。`\@elt` はコピーされた `\relax` が展開不可能であることから、そのまま残る。あとは特に説明をしなくても `\@cons` がリストの末尾に新たな要素を加える命令であることがわかるだろう。

### 1.3 カウンター値の更新

`\newcounter` で子カウンター `LaTeXcounterC` を新設する際に、親カウンターのリセットリスト `\cl@LaTeXcounterP` に `\@eltLaTeXcounterC` という可変命令付の要素が付加されるという「準備」が行われていることがわかったところで、いよいよ「親カウンターの値が1増えると、子カウンターの値が0にリセットされる」仕組みを見ていくことにする。これに該当する  $\text{\LaTeX}$  流カウンターの更新命令は `\stepcounter` と `\refstepcounter` の2種類があるが、後者は相互参照に関わるものなので、今回は `\stepcounter` のみを扱う。

---

```
1826 \def\stepcounter#1{%
1827   \addtocounter{#1}\@one
1828   \begingroup
1829     \let\@elt\@stpelt
1830     \csname cl@#1\endcsname
1831   \endgroup}
1832 \def\@stpelt#1{\global\csname c@#1\endcsname \z@}
```

---

準備をしっかりと行ってきたので、この定義の解釈は特に難しくない。1827行目は前回扱った `\addtocounter` 命令を用いて、対象のカウンター (`LaTeXcounterP`) の値を1つ大きくしている。1828~1831行目は可変命令 `\@elt` への制御綴り代入の影響が外部に及ばないように `\begingroup` と `\endgroup` で挟まれていて、その内部で行われていることは `\@elt` に `\@stpelt` を代入してリセットリスト `\cl@LaTeXcounterP` を実行している。`\@stpelt` は1832行目の定義を見ればわかる通り、引数に与えられたカウンターの値を0にするものであるため、これにより子カウンターが一斉にリセットされることになる。

---

\*1 「要素」の定義は、`\@car` と `\@cdr` の定義を見るに「引数の取り方」のそれに一致するらしい。「引数の取り方」が「トークン化ルール」と一致しないことに注意。

## 2 \newcommand の動作 (前半)

latex.ltx のうち、\newcommand に関連する部分を抜き出すと次のようになる。

```
589 \def\@star@or@long#1{%
590   \@ifstar
591   {\let\l@ngrel@x\relax#1}%
592   {\let\l@ngrel@x\long#1}}
593 \let\l@ngrel@x\relax
594 \def\newcommand{\@star@or@long\new@command}
595 \def\new@command#1{%
596   \@testopt{\@newcommand#1}0}
597 \def\@newcommand#1[#2]{%
598   \kernel@ifnextchar [{\@xargdef#1[#2]}%
599     {\@argdef#1[#2]}}
600 \long\def\@argdef#1[#2]#3{%
601   \@ifdefinable #1{\@yargdef#1\@ne{#2}{#3}}
602 \long\def\@xargdef#1[#2] [#3]#4{%
603   \@ifdefinable#1{%
604     \expandafter\def\expandafter#1\expandafter{%
605       \expandafter
606         \@protected@testopt
607         \expandafter
608         #1%
609         \csname\string#1\endcsname
610         {#3}}%
611     \expandafter\@yargdef
612     \csname\string#1\endcsname
613     \tw@
614     {#2}%
615     {#4}}
616 \long\def\@testopt#1#2{%
617   \kernel@ifnextchar [{#1}{#1[#{#2}]}
618 \def\@protected@testopt#1{%%
619   \ifx\protect\@typeset@protect
620     \expandafter\@testopt
621   \else
622     \@x@protect#1%
623   \fi}
624 \long \def \@yargdef #1#2#3{%
625   \ifx#2\tw@
626     \def\reserved@b##11{[#####1]}%
627   \else
628     \let\reserved@b\@gobble
629   \fi
630   \expandafter
631   \@yargd@f \expandafter{\number #3}#1%
632 }
633 \long \def \@yargd@f#1#2{%
634   \def \reserved@a ##1#1##2##{%
635     \expandafter\def\expandafter#2\reserved@b ##1#1%
636   }%
637   \l@ngrel@x \reserved@a 0##1##2##3##4##5##6##7##8##9##1%
638 }
639 \long\def\@reargdef#1[#2]{%
640   \@yargdef#1\@ne{#2}}
641 \def\renewcommand{\@star@or@long\renew@command}
642 \def\renew@command#1{%
643   \begingroup \escapechar\m@ne\xdef\@gtempa{\string#1}\endgroup
644   \expandafter\@ifundefined\@gtempa
645     {\@latex@error{\noexpand#1undefined}\@ehc}%
646   \relax
```

```
647 \let\@ifdefinable\rc@ifdefinable
648 \newcommand#1}
```

今回は幾つかの例を通じて、`\newcommand` の動作について考えていく。

## 2.1 例 1：アスタリスク・引数共に無い場合

```
\newcommand{\ほげ}{ふが}
```

上のソースコードは、まず `\newcommand` (594 行目) の展開により、

```
\@star@or@long\newcommand{\ほげ}{ふが}
```

となる。さらに `\@star@or@long` (589 行目) を展開すると、

```
\ifstar{\let\l@ngrel@x\relax\newcommand}{\let\l@ngrel@x\long\newcommand}{\ほげ}{ふが}
```

となる。今回はアスタリスクがついていないため、`\ifstar` の展開<sup>\*2</sup>により

```
\let\l@ngrel@x\long\newcommand{\ほげ}{ふが}
```

が得られる。これを展開することにより、`\l@ngrel@x` に `\long` が代入され (これは後ほど再登場する)、続けて `\newcommand` (595 行目) が展開されるため、

```
\@testopt{\@newcommand\ほげ}0{ふが}
```

となる。`\@testopt` は、616 行目の定義を読めばわかる通り、引数の個数を指定する `[]` の有無を調べる制御綴となっている。従って、この展開により、

```
\kernel@ifnextchar[{\@newcommand\ほげ}{\@newcommand\ほげ [0]}]{ふが}
```

が得られるが、今回は直後にトークン “[” が続いていないため、`\kernel@ifnextchar` (これは、800 行目で別途 `\let\kernel@ifnextchar\@ifnextchar` と定義されている) の展開により次のようになる。

```
\@newcommand\ほげ [0]{ふが}
```

これを 597 行目に従い素直に展開すると、

```
\kernel@ifnextchar [{\@xargdef\ほげ [0]}{\@argdef\ほげ [0]}]{ふが}
```

を得る。先ほどと同様に `\kernel@ifnextchar` の展開を考えると、今回は `\argdef` が読み込まれるため (この `\kernel@ifnextchar` は、第一引数の規定値を定める `[]` の有無を調べている)、展開後は

```
\@argdef\ほげ [0]{ふが}
```

であり、さらに `\@argdef` の定義 (600 行目) を参照すると、

```
\@ifdefinable \ほげ{\@yargdef\ほげ\@ne{0}}{ふが}
```

を得る。よって、`\@ifdefinable` によって制御綴 `\ほげ` が定義可能かどうかを調べ<sup>\*3</sup>、可能である場合 `\@yargdef` 以下が展開され、

```
\ifx\@ne\tw@
  \def\reserved@b##11{[####1]}%
\else
  \let\reserved@b\@gobble
\fi
\expandafter\@yargdef \expandafter{\number 0}\ほげ{ふが}
```

となる。

さて、今回 `\ifx` は偽であるため `\reserved@b` には `\@gobble` が代入される。`\@gobble` 自体は次のように定義されている。

```
725 \long\def \@gobble #1{}
```

\*2 詳細は本筋から外れるため割愛する。

\*3 この詳しい挙動もまた、本筋から外れるため今回は割愛する。

単に、「後ろの1トークンを展開させずに吸収する」制御綴のようである。  
話を戻そう。TeXは上の例の最終行の展開に入り、

```
\@yargd@f{0}\ほげ{ふが}
```

が得られるが\*4、次に \@yargd@f (633行目)の展開に入る。

ここで、 \@yargd@f の定義を再掲しよう。

---

```
633 \long \def \@yargd@f#1#2{%
634   \def \reserved@a ##1##2##{%
635     \expandafter\def\expandafter#2\reserved@b ##1#1%
636     }%
637   \l@ngrel@x \reserved@a 0##1##2##3##4##5##6##7##8##9##1%
```

---

これに基づくと、今回は

```
#1 <- 0
#2 <- \ほげ
```

が代入される事がわかる。

■疑問1 ##1 や ##2 などは、引数を持つマクロの内部で、更に引数を持つマクロを定義するために登場している  
と理解できるが、634行目の ##1##2## など、最後の ## の挙動がわからない。

『TeXブック』を参照した所、##のように、#を2つ続けて使用した場合、TeXはその展開によって#  
トークン1個を生成することがわかった。以下ではそれに基づいて考える。

■疑問2 それだけでなく、 \@yargd@f 展開時の、#の取り去り方の規則もわからない。

上と同様、展開時に「#が取り去られる」と考えるのではなく、##を展開すると#が生成される、と考  
えるのが正しい。

\@yargd@f をそのまま展開すると、

```
\def \reserved@a #10#2#\{ \expandafter\def\expandafter\ほげ\reserved@b #10}%
\l@ngrel@x \reserved@a 0#1#2#3#4#5#6#7#8#9#0{ふが}
```

が得られる。特に一番最後、###1については、まず先頭の##が#に展開され、次に#1に0を代入されるこ  
とにより、#0を得ている事に注意したい。

さて次に \reserved@a の定義を読んでいくと、一行目の展開終了後には、

```
\reserved@a#10#2# -> \expandafter\def\expandafter\ほげ\reserved@b #10
```

となる。

---

\*4 この \number は、引数の個数を数字でなくカウンタで与えた時のために挿入されていると思われる。

■疑問3 `\reserved@a#10#2#` の最後の # の意味がわからない.

The *T<sub>E</sub>Xbook* の 204 頁に、以下のような記述がある.

A special extension is allowed to these rules: If the very last character of the parameter text is #, so that this # is immediately followed by {, T<sub>E</sub>X will behave as if the { had been inserted at the right end of both the parameter text and the replacement text. For example, if you say `\def#1#{\hbox to #1}`, the subsequent text `\a3pt{x}` will expand to `\hbox to 3pt{x}`, because the argument of `\a` is delimited by a left brace.

従って、今回は `\reserved@a` の第二引数は、パターンマッチ文字列 0 の後から中括弧 { が現れるまで全てということになる.

さて次の行だが、普通に `\l@ngrel@x` が `\long` に置換されたのち、`\reserved@a` が展開される. ここで、T<sub>E</sub>X は `0#1#2#3#4#5#6#7#8#9#0` を前から順に読んでいくわけだが、まずいきなりパターンマッチ文字 0 に出会うため、T<sub>E</sub>X は第一引数を空とし、残りの文字列 `#1#2...` から { までのトークン列を第二引数として読み込んでいく.

結局 `\reserved@a` の引数は、

```
#1 <-  
#2 <- ##1##2##3##4##5##6##7##8##9##0
```

となる. 従って、展開結果は次の通り.

```
\long \expandafter\def\expandafter\ほげ\reserved@b 0{ふが}
```

あとは単純で、`\reserved@b` (いまは `\@gobble` の定義がコピーされている) の働きにより、

```
\def\ほげ{ふが}
```

となり、無事に展開が完了する. # まわりの正確な挙動および、引数を持つ場合の `\newcommand` の動作などは、次週扱うことにする.