latex.ltx リーディング
第 4 回資料

東大 TEX 愛好会

2015 年 6 月 5 日（2015 年 8 月 19 日 版）

1 \newcommand の動作（中編）

まず，\newcommand 関連の定義を再掲する．

```
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%
```

```
}
\long \def \@yargd@f#1#2{%
  \def \reserved@a ##1#1##2##{%
    \expandafter\def\expandafter#2\reserved@b ##1#1%
    }%
  \l@ngrel@x \reserved@a 0##1##2##3##4##5##6##7##8##9###1%
}
\long\def\@reargdef#1[#2]{%
  \@yargdef#1\@ne{#2}}
\def\renewcommand{\@star@or@long\renew@command}
\def\renew@command#1{%
  \begingroup \escapechar\m@ne\xdef\@gtempa{{\string#1}}\endgroup
  \expandafter\@ifundefined\@gtempa
     {\@latex@error{\noexpand#1undefined}\@ehc}%
     \relax
  \let\@ifdefinable\@rc@ifdefinable
  \new@command#1}
```

今回は前回に引き続き，引数が複数存在する場合，第一引数がオプションの場合，アスタリスクが付いている場合における \newcommand の動作を考えていく．

1.1 例 2：複数の引数をもつ場合

次のような例を考える．

```
\newcommand{\ぴよ}[3]{好きな食べ物は#1と#2と#3です．}
```

上のコードの展開・実行を順次追っていこう．

```
\newcommand{\ぴよ}[3]{好きな食べ物は#1と#2と#3です．}

-> \@star@or@long\new@command{\ぴよ}[3]{好きな食べ物は#1と#2と#3です．}

-> \@ifstar{\let\l@ngrel@x\relax\new@command}{\let\l@ngrel@x\long\new@command}{\ぴよ}[3]{
    好きな食べ物は#1と#2と#3です．}
```

ここで，今回はアスタリスクがついていないため，\@ifstar の展開[*1]により \let\l@ngrel@x\long\new@command が実行されるため，\l@ngrel@x には \long が代入され，以下では \new@command の展開に入る．

```
\new@command{\ぴよ}[3]{好きな食べ物は#1と#2と#3です．}

-> \@testopt{\@newcommand\ぴよ}0[3]{好きな食べ物は#1と#2と#3です．}
```

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である．

この挙動を詳しく追うと，\@testopt の展開により，

```
\kernel@ifnextchar[{\@newcommand\ぴよ}{\@newcommand\ぴよ [{0}]}[3]{好きな食べ物は#1と#2と#3で
    す．}
```

が得られ，今回は直後にトークン “[” が続いているため，\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・実行により次が得られるようになっている．

```
\@newcommand\ぴよ [3]{好きな食べ物は#1と#2と#3です．}
```

話を戻そう．\@newcommand の展開を考えていくと，

```
\@newcommand\ぴよ [3]{好きな食べ物は#1と#2と#3です．}

-> \kernel@ifnextchar [{\@xargdef\ぴよ [3]}{\@argdef\ぴよ [3]}{好きな食べ物は#1と#2と#3です．}
```

[*1] 詳細は本筋から外れるため割愛する．

```
-> \@argdef\ぴよ [3]{好きな食べ物は#1と#2と#3です．}

-> \@ifdefinable \ぴよ{\@yargdef\ぴよ\@ne{3}{好きな食べ物は#1と#2と#3です．}}

-> \ifx\@ne\tw@
     \def\reserved@b#11{[##1]}%
   \else
     \let\reserved@b\@gobble
   \fi
   \expandafter
     \@yargd@f \expandafter{\number 3}\ぴよ{好きな食べ物は#1と#2と#3です．}
```

を得る．

この途中で再び \kernel@ifnextchar が登場するが，これは第一引数の規定値を定める [] の有無を調べている．

また，\@ifdefinable という制御綴も途中で登場するが，これは制御綴 \ぴよ が定義可能かどうかを調べる機能を持つようである[*2]．今回はもちろん \ぴよ が定義可能であることを想定しているので，\@yargdef 以下が展開される．

さらに前回紹介した規則により，## トークンは \def の実行時に # に展開されていることに注意されたい．

さて，今回 \ifx は偽であるため \let の実行により \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{}
```

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，

```
\expandafter\@yargd@f\expandafter{\number 3}\ぴよ{好きな食べ物は#1と#2と#3です．}

-> \@yargd@f{3}\ぴよ{好きな食べ物は#1と#2と#3です．}

-> \def \reserved@a #13#2#{%
      \expandafter\def\expandafter\ぴよ\reserved@b #13%
      }%
   \l@ngrel@x \reserved@a 0#1#2#3#4#5#6#7#8#9#3{好きな食べ物は#1と#2と#3です．}
```

が得られる[*3]．

この先では，まず最初の \def の実行により，

```
\reserved@a#13#2# -> \expandafter\def\expandafter\ぴよ\reserved@b #13
```

というトークンの置換が発生する．次に 3 行目を実行すると，

```
\l@ngrel@x \reserved@a 0#1#2#3#4#5#6#7#8#9#3{好きな食べ物は#1と#2と#3です．}

-> \long \reserved@a 0#1#2#3#4#5#6#7#8#9#3{好きな食べ物は#1と#2と#3です．}

-> \long\expandafter\def\expandafter\ぴよ\reserved@b 0#1#2#3{好きな食べ物は#1と#2と#3です．}

-> \long\def\ぴよ#1#2#3{好きな食べ物は#1と#2と#3です．}
```

となる．\reserved@a の引数として，

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

[*2] 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．

[*3] 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．

が代入されている事に注意されたい．\reserved@a のパラメーターテキスト中の 3 は，パターンマッチ文字列として使用されている．

以上より，TEX プリミティブのコードである

```
\long\def\ぴよ#1#2#3{好きな食べ物は#1と#2と#3です．}
```

に無事置換することができた．

1.2 例 3：複数の引数をもち，第一引数がオプショナルな場合

```
\newcommand{\ふが}[3][カレー]{好きな食べ物は#1と#2と#3です．}
```

\@newcommand の展開までは，先ほどの例 2 と全く同様なので割愛する．このとき，

```
\kernel@ifnextchar [{\@xargdef\ふが [3]}{\@argdef\ふが [3]}[カレー]{好きな食べ物は#1と#2と#3です．}
```

を得る．ここで，\kernel@ifnextchar の展開を考えると，今回は \@xargdef の側が読み込まれるため，展開・実行後は

```
\@xargdef\ふが [3][カレー]{好きな食べ物は#1と#2と#3です．}
```

であり，さらに \@xargdef の定義（602 行目）を参照すると，

```
\@ifdefinable\ふが{%
  \expandafter\def\expandafter\ふが\expandafter{%
      \expandafter
      \@protected@testopt
      \expandafter
      \ふが%
      \csname\string\ふが\endcsname
      {カレー}}%
    \expandafter\@yargdef
      \csname\string\ふが\endcsname
       \tw@
       {3}%
       {好きな食べ物は#1と#2と#3です．}}
```

を得る．よって，\@ifdefinable によって制御綴 \ふが が定義可能かどうかを調べ，可能である場合以下が展開される．

```
\expandafter\def\expandafter\ふが\expandafter{%
      \expandafter
      \@protected@testopt
      \expandafter
      \ふが%
      \csname\string\ふが\endcsname
      {カレー}}%
    \expandafter\@yargdef
      \csname\string\ふが\endcsname
       \tw@
       {3}%
       {好きな食べ物は#1と#2と#3です．}
```

このコードの展開・実行を順次追っていくと，次のようになる．

```
\expandafter\def\expandafter\ふが\expandafter{%
        \expandafter
        \@protected@testopt
        \expandafter
        \ふが%
        \csname\string\ふが\endcsname
        {カレー}}%
     \expandafter\@yargdef
        \csname\string\ふが\endcsname
         \tw@
         {3}%
         {好きな食べ物は#1と#2と#3です．}

-> \def\ふが{\@protected@testopt\ふが\\ふが{カレー}}%
   \expandafter\@yargdef\csname\string\ふが\endcsname\tw@{3}{好きな食べ物は#1と#2と#3です．}

-> \def\ふが{\@protected@testopt\ふが\\ふが{カレー}}%
   \@yargdef\\ふが\tw@{3}{好きな食べ物は#1と#2と#3です．}

   (以下\@yargdef の展開に入る)

-> \def\reserved@b#11{[##1]}%
   \expandafter\@yargd@f \expandafter{\number 3}\\ふが{好きな食べ物は#1と#2と#3です．}

-> \@yargd@f{3}\\ふが{好きな食べ物は#1と#2と#3です．}

   (以下\@yargd@f の展開に入る)

-> \def \reserved@a #13#2#{\expandafter\def\expandafter\\ふが\reserved@b #13}
   \l@ngrel@x \reserved@a 0#1#2#3#4#5#6#7#8#9#3{好きな食べ物は#1と#2と#3です．}

-> \long\expandafter\def\expandafter\\ふが\reserved@b 0#1#2#3{好きな食べ物は#1と#2と#3です
    ．}

-> \long\def\\ふが [#1]#2#3{好きな食べ物は#1と#2と#3です}
```

途中で，\\ふが という制御綴が \csname および \string により（無理やり）定義されている事に注目されたい．その内実は，

```
\long\def\\ふが [#1]#2#3{好きな食べ物は#1と#2と#3です}
```

であり，さらに \ふが は

```
\def\ふが{\@protected@testopt\ふが\\ふが{カレー}}
```

と定義された．

また，\reserved@b の展開（最終行）にも注意されたい．これは，

```
\def\reserved@b#11{[##1]}
```

と定義されている．今回では，

```
\reserved@b 0#1#2#3
```

のうち，0# が第一引数に相当する．展開時には [##1] 中のトークン列 ## が # に展開されることにより，結果として

```
[#1]#2#3
```

が得られている．

1.2.1 制御綴が使用された際の処理

さて実際に，\newcommand がソース中で読み込まれた際の動作を考えよう．今回は例として，

```
\newcommand{\ふが}[3][カレー]{好きな食べ物は#1と#2と#3です．}
```

と予め定義された状況を考える．まず確認だが，これにより，

```
\def\ふが{\@protected@testopt\ふが\\ふが{カレー}}
\def\\ふが [#1]#2#3{好きな食べ物は#1と#2と#3です．}
```

と定義される．

展開・実行の様子を詳しく見てみると，次のようになる．まずは第一引数が与えられている場合を考えよう．

```
\ふが [ラーメン]{そば}{つけ麺}

-> \@protected@testopt\ふが\\ふが{カレー}[ラーメン]{そば}{つけ麺}

-> \ifx\protect\@typeset@protect
    \expandafter\@testopt
  \else
    \@x@protect\ふが%
  \fi\\ふが{カレー}[ラーメン]{そば}{つけ麺}
```

さてここで，\@typeset@protect という制御綴が登場する．

■疑問 1　\@typeset@protect の意味，用法がよくわからない．

source2e.pdf（LaTeX 2_ε プリミティブの命令の意味や使用法が解説されているテキスト）を参照したところ，これは \DeclarRobustCommand などの，ロバストな制御綴の定義に関わるスイッチであると推察される．状況によって，\protect または \relax と等価になるようである．
「普通の」制御綴を定義する場合については基本的に \protect と等価で考えて良いと思われる（理由は後述）．この制御綴の正式な意味は，次回以降 \DeclarRobustCommand などの定義を考えていく中で理解していきたい．

まず，\@typeset@protect が \protect に等しい場合，すなわち \ifx が真である場合を考える．この時 TeX は

```
\ifx\protect\@typeset@protect
    \expandafter\@testopt
  \else
    \@x@protect\ふが%
  \fi\\ふが{カレー}[ラーメン]{そば}{つけ麺}
```

を先頭から読み込んでゆくが，まず \expandafter により \@testopt が展開されないまま，\else が読まれ，以下 \fi まで展開がスキップされるため，結局，if 文の展開後には

```
\@testopt\\ふが{カレー}[ラーメン]{そば}{つけ麺}
```

が得られる．あとは，\@testopt の定義に従う事により，

```
\\ふが [ラーメン]{そば}{つけ麺}
```

が無事に得られる．

次に，\@typeset@protect が \relax に等しい場合では，TeX は

```
\ifx\protect\@typeset@protect
    \expandafter\@testopt
  \else
    \@x@protect\ふが%
  \fi\\ふが{カレー}[ラーメン]{そば}{つけ麺}
```

の展開を行う際に，\@x@protect の定義（763 行目）

```
\def\@x@protect#1\fi#2#3{%
   \fi\protect#1%
}
```

に従い，if 文の外に

```
\protect\ふが [ラーメン]{そば}{つけ麺}
```

を展開することになる．\protect から察せられる通り，これは制御綴のロバストな定義に何かしら関わっていると考えられる．これら 2 例から分かる通り，通常の制御綴の定義を考える際には，\@typeset@protect は \protect と等価であると考えれば差し障りが無いといえる[*4]．

また，第一引数が省略された場合の挙動も簡単に推察できる．要するに，先の例から [ラーメン] を取り去って考えれば，

```
\@testopt\\ふが{カレー}{そば}{つけ麺}
```

の展開により

```
\\ふが [カレー]{そば}{つけ麺}
```

が正しく得られる．

1.3 例 4：\newcommand*の動作（アスタリスクが付く場合）

これは非常に単純で，1 回展開後の

```
\@star@or@long\new@command*{\hoge}[1]{hage#1}
```

で登場する \@star@or@long の展開・実行により \l@ngrel@x が \relax と等価になる（これまでの全ての例では，\l@ngrel@x は \long と等価であった）．これにより，最後の \@yarg@f の展開時に，

```
\relax\def\hoge#1{hage#1}
```

となり，引数が段落をまたぐ事を禁止するようになる．

[*4] もちろん，次週以降 \@typeset@protect の正式な意味が理解できた時点で，見解が変わるかもしれない．

2 LaTeX の環境入門

LaTeX の環境（environment）は文書の論理構造化の上で重要な役割を果たす LaTeX の特徴的な機構の一つである．本節では，環境を利用する際に使用する命令（\begin と \end）に注目しながら，その実態の理解を試みることにする．

2.1 環境の実体

はじめに，わかりにくいところは全部飛ばして \begin と \end の定義を簡単に俯瞰してみる．ここでは，一般の環境を LaTeXenv と表現することにする．

```
3944 \def\begin#1{%
3945   \@ifundefined{#1}%
3946     {\def\reserved@a{\@latex@error{Environment #1 undefined}\@eha}}%
3947     {\def\reserved@a{\def\@currenvir{#1}%
3948      \edef\@currenvline{\on@line}%
3949      \csname #1\endcsname}}%
3950   \@ignorefalse
3951   \begingroup\@endpefalse\reserved@a}
3952 \def\end#1{%
3953   \csname end#1\endcsname\@checkend{#1}%
3954   \expandafter\endgroup\if@endpe\@doendpe\fi
3955   \if@ignore\@ignorefalse\ignorespaces\fi}
3956 \def\@checkend#1{\def\reserved@a{#1}\ifx
3957      \reserved@a\@currenvir \else\@badend{#1}\fi}
3958 \let\@currenvline\@empty
```

\begin の定義冒頭 3945〜3929 行目は \reseved@a の定義を行うことに終始している．そして，定義された \reseved@a は 3951 行目で実行されている．では，\reseved@a にはどのようなことが定義されているのだろうか．見ての通り，\LaTeXenv が定義されているか否かでその定義が変化している．すなわち，\LaTeXenv が未定義であった場合には「Environment LaTeXenv undefined」というエラーを吐き出すように定義され，\LaTeXenv が定義済みであった場合には \@currenvir と \@currenvline を定義して \csname LaTeXenv\endcsname（\LaTeXenv）を実行するように定義される．

一方，\end の定義を見ると，はじめに \csname endLaTeXenv\endcsname （\endLaTeXenv）を実行してからいくつかの処理を行うようになっている．

つまり，まとめると \begin{LaTeXenv} ... \end{LaTeXenv} という一連の環境利用命令は，

（環境を始める内部手続き）\LaTeXenv ... \endLaTeXenv（環境を終える内部手続き）

と書き換えられる．

2.2 終了のチェック

さて，以下ではたったいま「内部手続き」と称し，ブラックボックス化して述べた部分の処理について多少詳しく分析していくことにする．

まずは \end の定義中に登場する \@checkend を考える．この制御綴は前ページに掲載した 3956,3957 行目で定義されている．この定義はとても単純で，\ifx を用いることにより，引数に与えた文字列（\end に渡した文字列）と \begin の中で定義された \@currenvir に格納された文字列が一致するかどうかを調べ，一致しない場合は \@badend{#1} が展開されることになる．\@badend 自体は次のように定義されており，要するにエラーを吐き出させる．

```
1074 \gdef\@badend#1{%
1075   \@latex@error{\protect\begin{\@currenvir}\@currenvline
1076                       \space ended by \protect\end{#1}}\@eha}
```

なお，\begin の \reserved@a より前に \begingroup，\end の \@checkend より後に \endgroup があることに注意されたい．これにより，ある環境中で別の環境が入り込み，一時的に \@currenvir が書き換えられたとしても（入り込んだ環境がきちんと閉じられていれば）問題なく条件分岐を行うことができる．

2.3 条件文とスイッチ

TEX のプリミティブに \iftrue や \iffalse というものがある．これらはそれぞれ，「常に真の処理」と「常に偽の処理」を行わせる．そんなもの必要なのかと一瞬思うが，「可変命令への代入」ということを考えると役に立つことがわかる．すなわち，この仕組みはちょうどスイッチのように使用できるのである．こうした例が \begin と \end に 2 種類含まれているので，見ていくことにする．

最初に，\if@ignore の周辺について考える．関連する定義を（今回関係する箇所に限らず）以下に記す．

```
1389 \def\@ignorefalse{\global\let\if@ignore\iffalse}
1390 \def\@ignoretrue {\global\let\if@ignore\iftrue}
1391 \@ignorefalse
1392 \let\ignorespacesafterend\@ignoretrue
```

1391 行目で \@ignorefalse が宣言されていることから，\if@ignore には \iffalse が代入されているのが初期状態であることがわかる．その上で，\begin の中で改めて \@ignorefalse が宣言され，環境開始前（\LaTeXenv の展開前）の段階では確実に \if@ignore には \iffalse が代入された状態となっている．したがって，このまま環境の終了時（\endLaTeXenv の展開後）まで，この状況が変化しない限り，\end の定義の 3995 行目は「何もしない」ことになる．逆に，環境の終了時までに \@ignoretrue が宣言されるなどしてこの状況が崩され，\if@ignore に \iftrue が代入された状態となっていた場合[*5]，\end の定義の 3995 行目の \@ignorefalse\ignorespaces が展開されることになる．ここで，\ignorespaces は後続の空白を無視するプリミティブである．

■疑問 2　どのような場合にどうして，環境終了後に \ignorespaces が必要となるのだろうか．

[*5] latex.ltx 内では \enddisplaymath, \endequation, \enddisplaymath, \endeqnarray の定義末尾に \@ignoretrue が存在している．

次に，\if@endpe 周りを考えよう．面白いことに，\if@ignore と似た仕組みであるにも関わらず，こちらは別の方法を用いてその用意がなされている．関連する定義を以下に掲載しておく．

```
4449 \def\@doendpe{\@endpetrue
4450     \def\par{\@restorepar\everypar{}\par\@endpefalse}\everypar
4451               {{\setbox\z@\lastbox}\everypar{}\@endpefalse}}
4452 \newif\if@endpe
4453 \@endpefalse
```

ここで，\newif はさきほどの \if@ignore 周りのスイッチ機構と同様のものを自動的に生成する LaTeX のマクロである（今回は，その定義は割愛する）．すなわち，\newif\ifHOGE と宣言すると，自動的に \HOGEtrue と \HOGEfalse および条件判断のための制御綴 \ifHOGE が使用可能になる．

話を元に戻そう．もう一度 1 ページ目の定義を眺めてもらうと，スイッチの用意方法は異なるが，\if@endpe 周りと \if@ignore 周りではほとんど同じようなことが行われていることがわかる．\newif によるスイッチの作成の直後，4453 行目でも \@endpefalse が宣言されているが，\begin の定義のほぼ末尾の位置（3951 行目）で再び \@endpefalse が宣言されている．したがって，この状況が変化しない限りは \end の定義中 3954 行目の \@doendpe が展開されることはないが，もし環境の \begin と \end の間のどこかで \@endpetrue の状態になった場合[*6]には \@doendpe が展開される．この \@doendpe に関する詳細は今回は扱わないことにするが，*The LaTeX 2ε Sources* によると環境に後続する段落でインデントが挿入されることを防止しているようである．

■疑問 3　次の 2 通りの TeX 文の違いは何か．

```
\expandafter\endgroup\if@endpe\@doendpe\fi
\if@endpe\@doendpe\fi\endgroup
```

上の TeX 文では，\expandafter の作用により \if@endpe は \endgroup よりも前に展開される．したがって，このスイッチは「環境内」が \if@endpetrue な状況であるかどうかで \@doendpe を展開するかどうかを決定する．そして，実際に「環境内」が \if@endpetrue であった場合，\@doendpe は「環境外」にて実行される．
一方，下の TeX 文では，スイッチ \if@endpe が「環境内」で \if@endpetrue な状況であるかどうかで \@doendpe の展開可否を決定する点は同様だが，実際に「環境内」が \if@endpetrue あった場合，\@doendpe は「環境内」にて実行されるため，この点に違いが生じる．
【参考】http://tex.stackexchange.com/questions/24495/what-exactly-does-doendpe-do

■疑問 4　なぜ \if@endpe の作成には \newif を用い，\if@ignore の作成には用いていないのか．

[*6] \@endpetrue は latex.ltx 内では \@doendpe と \endtrivlist の定義中でのみ登場しており，\endtrivlist の方はいくつかの環境の終了命令（\endcenter, \endflushleft, \endflushright など）の定義内で登場している．