latex.ltx リーディング
第 8 回資料

東大 TEX 愛好会

2015 年 11 月 9 日

1 エラーメッセージまわりその 1 （大浦）

lterror.dtx に由来する部分を読んでいく。長いので，すこしずつ。今日は定義の前半部を掲載するが，この全部を読み切ろうというわけでもない。すこしずつ，続けられるペースで読んでいきたいから...

1.1 定義 (Part 1)

```
\expandafter\let\csname ver@autoerr.sty\endcsname\fmtversion
\let\MessageBreak\relax
\DeclareRobustCommand{\GenericInfo}[2]{%
   \begingroup
      \def\MessageBreak{^^J#1}%
      \set@display@protect
      \immediate\write\m@ne{#2\on@line.}%
   \endgroup
}
\DeclareRobustCommand{\GenericWarning}[2]{%
   \begingroup
      \def\MessageBreak{^^J#1}%
      \set@display@protect
      \immediate\write\@unused{^^J#2\on@line.^^J}%
   \endgroup
}
\bgroup
\lccode`\@=`\ %
\lccode`\~=`\ %
\lccode`\}=`\ %
\lccode`\{=`\ %
\lccode`\T=`\T%
\lccode`\H=`\H%
\catcode`\ =11\relax%
\lowercase{%
\egroup%
\dimen@\ifx\@TeXversion\@undefined4\else\@TeXversion\fi\p@%
\ifdim\dimen@>3.14\p@%
\DeclareRobustCommand{\GenericError}[4]{%
\begingroup%
\immediate\write\@unused{}%
\def\MessageBreak{^^J}%
\set@display@protect%
\edef%
\@err@ %
{{#4}}%
\errhelp
```

```
\@err@ %
\let
\@err@ %
\@empty
\def\MessageBreak{^^J#1}%
\def~{\errmessage{%
#2.^^J^^J%
#3^^J%
Type H <return> for immediate help%
\@err@ %
}}%
~%
\endgroup}%
\else%
\DeclareRobustCommand{\GenericError}[4]{%
\begingroup%
\immediate\write\@unused{}%
\def\MessageBreak{^^J}%
\set@display@protect%
\edef%
\@err@ %
{{#4}}%
\errhelp
\@err@ %
\let
\@err@ %
\errmessage
\def\MessageBreak{^^J#1}%
\def~{\typeout{! %
#2.^^J^^J%
#3^^J%
Type H <return> for immediate help.}%
\@err@ %
{}}%
~%
\endgroup}%
\fi}%
```

1.2 定義冒頭

```
\expandafter\let\csname ver@autoerr.sty\endcsname\fmtversion
\let\MessageBreak\relax
```

877 行目は，\ver@autoerr.sty というコマンドを \fmtversion の別名として定義しているだけである。

878 行目は，あとで散々出てくる \MessageBreak にとりあえず \relax を代入して備えているだけである。あとから \MessageBreak には色々なものが代入される。\MessageBreak は，その名の通り，（エラー）メッセージ同士の区切りにどのようなものを使うかということである。

1.3 GenericInfo

```
\DeclareRobustCommand{\GenericInfo}[2]{%
   \begingroup
      \def\MessageBreak{^^J#1}%
      \set@display@protect
      \immediate\write\m@ne{#2\on@line.}%
   \endgroup
}
```

ここで定義されるコマンドは，\GenericInfo である。これを，引数を二つ取る堅牢なコマンドとして定義

する。その中身はというと，さっき用意した \MessageBreak として ^^J#1，つまり改行＋第一引数を指定し，\set@display@protect を実行し，すぐに.log ファイルに，#2\on@line. を書くということである。\m@ne は第 7 回資料にある通り，マイナス 1 のことであり，\write の後に負の数が来た場合には \write の第二引数は.log ファイル送りになるので，このような挙動を示す。

\set@display@protect については，別途 latex.ltx に定義がある。

```
763 \def\set@display@protect{\let\protect\string}
```

つまり，単に \protect を，\string の別名として定義する，という意味である。

1.4 GenericWarning

```
879 \DeclareRobustCommand{\GenericWarning}[2]{%
880   \begingroup
881     \def\MessageBreak{^^J#1}%
882     \set@display@protect
883     \immediate\write\@unused{^^J#2\on@line.^^J}%
884   \endgroup
885 }
```

先ほどとの違いは，2 点ある。1 つめはわかりやすいところだが，表示されるメッセージの内容が，^^J#2\on@line.^^J となり，先ほどより前後の改行が多いということである。2 つめは，\m@ne が \@unused になっているということである。

\@unused は，latex.ltx の 1339 行目に次のようにして定義されている。

```
879 \newwrite\@unused
```

そして，\newwrite は，第 7 回資料で述べられたとおり，\newcount 系のコマンドであり，次のように定義されている。

```
332 \def\newwrite{\alloc@7\write\chardef\sixt@@n}
```

また，ここに登場する \alloc@は，次のように定義されているのであった。

```
334 \def\alloc@#1#2#3#4#5{\global\advance\count1#1\@ne
335 \ch@ck#1#4#2% make sure there's still room
336 \allocationnumber\count1#1%
337 \global#3#5\allocationnumber
338 \wlog{\string#5=\string#2\the\allocationnumber}}
```

そして，ここに登場する \ch@ck は，次のように定義されていた。

```
334 \gdef\ch@ck#1#2#3{%
335 \ifnum\count1#1<#2\else
336 \errmessage{No room for a new #3}%
337 \fi}
```

では，

```
\newwrite\@unused
```

がどのように展開されていくのか追っていくことにしよう。

第一段階

```
\alloc@7\write\chardef\sixt@@n\@unused
```

第二段階

```
\alloc@7\write\chardef\sixt@@n\@unused
```

第三段階

```
\global\advance\count17\@ne
```

```
\ch@ck7\sixt@@n\write
\allocationnumber\count17%
\global\chardef\@unused\allocationnumber
\wlog{\string\@unused=\string\write\the\allocationnumber}
```

\advance が実行され,\count17 の値が 1 増える。\count17 の初期値は-1 なので，ここで \count17 は 0 になることが多かろう。

第四段階

```
\ifnum\count17<\sixt@@n\else
\errmessage{No room for a new \write}%
\fi
\allocationnumber\count17%
\global\chardef\@unused\allocationnumber
\wlog{\string\@unused=\string\write\the\allocationnumber}
```

ここの \ifnum で，現在割り当てようとしているレジスタの番号が，16 より小さいかを判別し，これを超過するようであればエラーが出る。このチェックを通り抜けた場合，\allocationnumber に今から割り当てるレジスタの番号を格納し，

```
\global\chardef\@unused\allocationnumber
```

によって，レジスタに \@unused を割り当てる。最後の \wlog により，.log ファイルに \@unused が定義されたことを書き込む。

この構成方法からわかるとおり，\@unused に入っている値は，0 以上 15 以下の数である。

\write の第一引数として，負の数または 15 より大きい値が指定された時は.log ファイル送りになるので，\@unused が上限値の 15 に達していた時でさえ.log ファイルに書き込まれ知らせてくれたりはしない。一方，\write の第一引数として，0 以上の数が指定された時にはターミナル送りになるので，今回は必ずターミナル送りとなる。そりゃユーザに対する Warning がターミナル送りにされなかったらそれは困るでしょう。

1.5 GenericError

Info と Warning を見てきたが，次に大物の Error である。Error はタイプセットの流れを止めるわけだから，それこそ定義が大きくなるのは仕方がない話である。今日のところは，\GenericError 定義部のおおまかな構造を述べるにとどめ，詳細な定義は次回以降に述べることにする。定義を再掲しておくと，次のようになっている。本当に中かっこの数が一緒かどうか確かめるのでさえ一苦労である（当然ちゃんとあっているのだが)。

```
893 \bgroup
894 \lccode`\@=`\ %
895 \lccode`\~=`\ %
896 \lccode`\}=`\ %
897 \lccode`\{=`\ %
898 \lccode`\T=`\T%
899 \lccode`\H=`\H%
900 \catcode`\ =11\relax%
901 \lowercase{%
902 \egroup%
903 \dimen@\ifx\@TeXversion\@undefined4\else\@TeXversion\fi\p@%
904 \ifdim\dimen@>3.14\p@%
905 \DeclareRobustCommand{\GenericError}[4]{%
906 \begingroup%
907 \immediate\write\@unused{}%
908 \def\MessageBreak{^^J}%
909 \set@display@protect%
910 \edef%
911 \@err@ %
912 {{#4}}%
```

```
\errhelp
\@err@ %
\let
\@err@ %
\@empty
\def\MessageBreak{^^J#1}%
\def~{\errmessage{%
#2.^^J^^J%
#3^^J%
Type H <return> for immediate help%
\@err@ %
}}%
~%
\endgroup}%
\else%
\DeclareRobustCommand{\GenericError}[4]{%
\begingroup%
\immediate\write\@unused{}%
\def\MessageBreak{^^J}%
\set@display@protect%
\edef%
\@err@ %
{{#4}}%
\errhelp
\@err@ %
\let
\@err@ %
\errmessage
\def\MessageBreak{^^J#1}%
\def~{\typeout{! %
#2.^^J^^J%
#3^^J%
Type H <return> for immediate help.}%
\@err@ %
{}}%
~%
\endgroup}%
\fi}%
```

まず最初に，

```
\lccode`\@=`\ %
\lccode`\~=`\ %
\lccode`\}=`\ %
\lccode`\{=`\ %
\lccode`\T=`\T%
\lccode`\H=`\H%
\catcode`\ =11\relax%
\lowercase{%
\egroup%
```

でおなじみの（？）カテゴリコード変更である。`\lowercase` がでてくるあたりが `latex.ltx` らしい技巧ではなかろうか。

```
\dimen@\ifx\@TeXversion\@undefined4\else\@TeXversion\fi\p@%
\ifdim\dimen@>3.14\p@%
```

ここで，TEX のバージョンが 3.14 より新しいかどうかを判別して，それによって `\GenericError` の定義を変える条件分岐をしている。そして，残りは 927 行目の `\else` で区切られて，それぞれの条件におけるエラー生成コマンドを定義しているのである。

2 \DeclareRobustCommand の実装（大門）

生来的に保護付（natively-protected）なマクロを定義するための制御綴である，\DeclareRobustCommand の実装を考えていく。

```
\def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
\def\declare@robustcommand#1{%
   \ifx#1\@undefined\else\ifx#1\relax\else
      \@latex@info{Redefining \string#1}%
   \fi\fi
   \edef\reserved@a{\string#1}%
   \def\reserved@b{#1}%
   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
   \edef#1{%
      \ifx\reserved@a\reserved@b
         \noexpand\x@protect
         \noexpand#1%
      \fi
      \noexpand\protect
      \expandafter\noexpand\csname
         \expandafter\@gobble\string#1 \endcsname
   }%
   \let\@ifdefinable\@rc@ifdefinable
   \expandafter\new@command\csname
      \expandafter\@gobble\string#1 \endcsname
}
\def\x@protect#1{%
   \ifx\protect\@typeset@protect\else
      \@x@protect#1%
   \fi
}
\def\@x@protect#1\fi#2#3{%
   \fi\protect#1%
}
```

\newcommand または \renewcommand の機能と共通する部分（第一引数の省略やアスタリスクの有無など）については，既にそれらの実装を調べた回の資料で示されているので，今回は制御綴の保護に関する部分を中心に考えていく。

2.1 具体例での展開追跡

具体例として，以下の TEX ソースの展開・実行を追うことにする。

```
\DeclareRobustCommand{\ほげ}{ふが}

-> \@star@or@long\declare@robustcommand{\ほげ}{ふが}

   (\@star@or@long の展開により, \let\l@ngrel@x\long が実行される)

-> \declare@robustcommand{\ほげ}{ふが}

-> \ifx#1\@undefined\else\ifx\ほげ\relax\else
      \@latex@info{Redefining \string\ほげ}%
   \fi\fi
   \edef\reserved@a{\string\ほげ}%
   \def\reserved@b{\ほげ}%
   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
```

```
\edef\ほげ{%
  \ifx\reserved@a\reserved@b
    \noexpand\x@protect
    \noexpand\ほげ%
  \fi
  \noexpand\protect
  \expandafter\noexpand\csname
    \expandafter\@gobble\string\ほげ \endcsname
}%
\let\@ifdefinable\@rc@ifdefinable
\expandafter\new@command\csname
  \expandafter\@gobble\string\ほげ \endcsname{ふが}
```

ここまでは簡単に展開されるので，この先の挙動を丁寧に考えていくことにする。

まず最初の 3 行の \ifx による分岐では **\ほげ**が定義済みかどうかを調べ，定義済みであれば Redefining する旨を報告する作業をしている。その先では，\reserved@a と \reserved@b にそれぞれ制御綴名とその置換えテキストを代入し，それらが一致するかどうかを比較している。

2.1.1 \reserved@a と \reserved@b の比較について

まず，\reserved@a については，

```
\edef\reserved@a{\string\ほげ}%
```

とあるように‘**\ほげ**’という制御綴名の文字列が代入される。対して \reserved@b は，

```
\def\reserved@b{\ほげ}%
\edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
```

とあるように，一旦 **\ほげ**そのものに等値した後，2 行目の \edef の部分で **\ほげ**の置換えテキストを代入している。ここを理解するにはまず \meaning の挙動を理解せねばならないが，\meaning とは引数にとる制御綴の意味を表す文字列に展開される TEX プリミティブで，具体例として

```
\def\Are#1{#1 is Are.}
\meaning\Are\par
\meaning\@undefined
```

を実行すれば，出力として

```
macro:#1->#1 is Are.
undefined
```

が得られる。(特殊な設定をしない場合，> はフォントの問題で¿と表示されると思われる。)

また，\strip@prefix とは latex.ltx に以下のように定義されている制御綴である。

```
\def\strip@prefix#1>{}
```

これは \meaning の展開結果である文字列のうち，macro から->までの部分を取り除く働きを持つので，結局のところ \reserved@b には置換えテキスト（または‘undefined’）が代入されることになる。

これらが，**\ほげ**のが \edef で定義される部分の中の \ifx で比較されることになる。

■疑問 1 \reserved@a と \reserved@b が一致するのは，どのような状況なのだろうか。言い換えると，制御綴名とその置換えテキストが一致する場合とはどういう場合なのだろうか。

2.1.2 \reserved@a と \reserved@b が一致しない場合

とりあえず \ifx が偽である場合を考える。特に，新しい制御綴を宣言する際には \reserved@b に‘undefined’が代入されるので，こちらの場合に相当する。このとき，

```
\edef\ほげ{%
  \ifx\reserved@a\reserved@b
    \noexpand\x@protect
```

```
      \noexpand\ほげ%
    \fi
    \noexpand\protect
    \expandafter\noexpand\csname
      \expandafter\@gobble\string\ほげ \endcsname
}%
```

の部分を（\edef 内であることに注意しながら）展開することにより，

```
\ほげ <- \protect\ほげ
```

という代入操作が得られる。\protect のあとに続く **\ほげ** は **\ほげ**とは別物であることに注意されたい。**\ほげ** は **\ほげ**の直後に半角空白が続いている制御綴である。ただしこの時点では **\ほげ** （半角空白が直後に続く制御綴）は \relax と等価な状態である（これは \csname の作用による）。

続く以下の部分を再び示すと以下のようになる。

```
\let\@ifdefinable\@rc@ifdefinable
\expandafter\new@command\csname
  \expandafter\@gobble\string\ほげ \endcsname{ふが}
```

ここは至って単純で，**\ほげ**の代わりに **\ほげ** （半角空白が直後に続く制御綴）に対して，\newcommand で行われた作業を実施しているだけである。ただし \DeclareRobustCommand では既存の定義の書き換えも想定されているので，定義可能かどうかを調べる制御綴に \renewcommand 用の \@rc@ifdefinable を割り当てている。この先の挙動は過去に扱ったので省略する。

結局のところ，\DeclareRobustCommand は内部で半角空白が直後に続く制御綴[*1]を \newcommand と同様に定義して，それに \protect を被せる，という操作を行なっていると考えられる。

2.1.3 \reserved@a と \reserved@b が一致する場合

これについては次回考えることにする（時間ありませんでした…）。

[*1] これは \def などの普通の方法ではユーザーは定義できないので，衝突の可能性が低い。