enumitem パッケージ

monaqa

2019/09/07

目次

1　enumitem パッケージの概要

enumitem は，組版用言語 SATYSFI において，豊富な箇条書きリストや番号付きのリストを出力するためのパッケージです．SATYSFI には標準で itemize というパッケージが標準で用意されていますが，enumitem パッケージでは itemize パッケージと比較してより豊富な機能を提供します（2019 年 11 月 2 日現在）．enumitem パッケージを用いることで，具体的に以下のような恩恵を受けることができます：

- デフォルトで豊富なスタイルを選択できる
- 番号付き箇条書き環境をネストさせることができる[*1]
- ネストごとに箇条書きのスタイルを変更できる
- ユーザーによって容易にスタイルを拡張できる

1 2019 年 9 月現在，標準ではサポートされていません．

以下，基本的な使い方，具体的なコマンドの一覧，そしてカスタマイズの方法について，順に説明していきます．

2　基本的な使い方

enumitem パッケージが提供する最も基本的なコマンドは `+genlisting` です．これは箇条書きを生成するためのコマンドであり，例えば，本文のブロックテキスト内に以下のように打つことができます．

```
+genlisting(label-arabic-dot){
  * hoge
    ** hoge1
    ** hoge2
  * fuga
  * piyo
}
```

すると，以下のように上から順に「アラビア数字 + ドット」で番号が振られた箇条書きの結果が得られます．結果から分かる通り，箇条書きのネストもサポートしています．

1. hoge
    1. hoge1
    2. hoge2
2. fuga
3. piyo

すでにお気づきかもしれませんが,「アラビア数字 + ドット」で番号をふる，という箇条書きスタイルは `+genlisting` コマンドの第 1 引数 `label-arabic-dot` によって指定されています． この指定子を適切に付け替えることで自由自在にラベルを変更することができます．たとえば先程の例で第 1 引数を `label-roman-paren` とすれば，以下のような箇条書きが代わりに表示されます．

(i) hoge
    (i) hoge1
    (ii) hoge2
(ii) fuga

(iii) piyo

この場合，「ローマ数字 + パーレン（丸括弧）」で番号をふる，という指定を第 1 引数の指定子 `label-roman-paren` で与えたことになります．このような指定子は他にもいくつかあります（詳細は後述）．

`+genlisting` は番号付きの箇条書きだけでなく，番号のつかない箇条書きもサポートしています．以下は番号のつかない箇条書きを出力する例です．

```
+genlisting(label-white-bullet){
  * hoge
    ** hoge1
    ** hoge2
  * fuga
  * piyo
}
```

これもまた第 1 引数を置き換えたものに過ぎず，目新しさはないでしょう．このように書くと，以下のような結果が得られます．

- hoge
  - hoge1
  - hoge2
- fuga
- piyo

これまでの例で，`+genlistings` の第 1 引数を変えることで一つの文書内で簡単に複数のスタイルを実現できるということを示してきました．しかし，実用的な文書においては，登場する箇条書きごとにスタイルを変えるというのは稀です．一つ一つの箇条書きにスタイルを設定するよりも，同一のスタイルの箇条書きを簡素なコマンドで使いまわしたいことのほうが多いでしょう．幸い，SATYSFI では LaTeX と同じようにコマンドを自分で定義することができます．もし `+mylisting` が `+genlisting(label-white-bullet)` と同様に動作するようにしたければ，プリアンブル部分に以下のように書くだけで実現できます．

```
let-block +mylisting item = '<
  +genlisting(label-white-bullet)(item);
```

```
>
```

このように一度 +mylisting コマンドを定義してしまえば，先ほどと同じ箇条書きをより簡単に実現できます．

```
+mylisting{
  * hoge
    ** hoge1
    ** hoge2
  * fuga
  * piyo
}
```

標準の itemize パッケージ同様，+listing コマンド及び +enumerate コマンドが enumitem パッケージにおいてもデフォルトで定義されています． これも，やはり上で述べたものと同様の方法で +genlisting コマンドから定義されています．+listing を定義する際のラベル指定はそれぞれ label-bullet であり，以下のようなスタイルになっています．

- hoge
  - hoge1
  - hoge2
- fuga
- piyo

+enumerate コマンドの定義がどうなっているのか気になるかもしれませんが，これはもう少しばかり複雑なことを行っているので，説明は後ろの章に譲ることとします．

3　コマンドとスタイル指定子

ここからは，enumitem パッケージが提供するコマンドとスタイルの一覧について述べます．

3.1　コマンド一覧

enumitem パッケージでは，箇条書きを作成するためのコマンドとして以下のコマンドが定義されています．

2019 年 10 月 27 日現在，LaTeX でいうところの description 環境に相当するコマンドはまだ実装されていません．

3.2 スタイル指定子一覧

インデックスを指定するためのパラメータです． これらは，実際にインデックスを見たほうが早いでしょう．

- アラビア数字系
  - 1 `label-arabic-raw`
  - 2. `label-arabic-dot`
  - (3) `label-arabic-paren`
  - [4] `label-arabic-bracket`
- ローマ数字系
  - i `label-roman-raw`
  - II `label-Roman-raw`
  - iii. `label-roman-dot`
  - IV. `label-Roman-dot`
  - (v) `label-roman-paren`
  - (VI) `label-Roman-paren`
  - [vii] `label-roman-bracket`
  - [VIII] `label-Roman-bracket`
- アルファベット系
  - a `label-alph-raw`
  - B `label-Alph-raw`
  - c. `label-alph-dot`
  - D. `label-Alph-dot`
  - (e) `label-alph-paren`
  - (F) `label-Alph-paren`
  - [g] `label-alph-bracket`
  - [H] `label-Alph-bracket`

これらの正体が何であるか，気になる方もいると思います．`+genlisting` の第 1 引数に指定する列挙型のごとく振る舞っていますが，その実体は「現在のテキスト処理文脈とインデックス値を受け取り，ラベルとすべきインラインボックス列を返す」役割を持った `context -> int -> inline-boxes` 型の関数です． 後述するようなカスタマイズ

を行わない限りユーザがこの関数を再定義したり明示的に引数を渡したりする機会はないため，本文中では列挙型と同じように扱うことができます．逆に言えば，ユーザ側が `context -> int -> inline-boxes` 型の関数を定義したならば，それがそのままユーザ定義された新たなインデックス指定パラメータとなります．

4　カスタマイズ

ここでは，enumitem パッケージを用いて自身の好みの箇条書き環境を作成するにはどのようにすればよいかについて述べます．内部の実装や型についても触れます．

4.1　数字の装飾を変えてみる

前章のコマンド一覧で紹介したとおり，enumitem パッケージではすでに 20 以上のインデックス指定関数が定義されています．しかし，それらはそれぞれ独立に定義されているわけではなく，効率的にコードを使い回すように定義しています．現に番号付きのインデックス指定関数を見れば，それらは実際には 2 つの要素の掛け合わせで構成されている（数学的には直積）ことが容易にわかるでしょう．具体的には，たとえば `label-arabic-raw` 関数は以下のように定義されています：

```
let label-arabic-raw = Enumitem.label-raw Enumitem.to-arabic
```

ここで `Enumitem.label-raw` は `(int -> inline-text) -> context -> int -> inline-boxes` の型を持つ関数であり，一方で `Enumitem.to-arabic` は `int -> inline-text` の型を持つ関数です．

enumitem パッケージ内では，このように 5 種類の to- 型関数（`Enumitem.to-arabic`, `Enumitem.to-roman`, `Enumitem.to-Roman`, `Enumitem.to-alph`, `Enumitem.to-Alph`）と 4 種類の label- 型関数（`Enumitem.label-raw`, `Enumitem.label-dot`, `Enumitem.label-paren`, `Enumitem.label-bracket`）を定義することにより，20 種類のインデックス指定関数を生み出しています．

この性質をうまく使えば，ユーザ定義の箇条書き環境をかんたんに定義することができます．たとえば，「**問 1.**」のようなインデックスを持つ箇条書き環境を定義してみましょう．1 はアラビア数字ですから，`Enumitem.to-arabic` 関数をそのまま用いることができます．

```
let label-toi num-format ctx idx =
  let label-width = (get-font-size ctx) *' 3.0 in
  let it-num = num-format idx in
```

```
  let ib-label = read-inline ctx {\textbf{問 #it-num;.}\ } in
  let (wid-label, _, _) = get-natural-metrics ib-label in
  inline-skip (label-width -' wid-label) ++ ib-label

let-block +toi item = '<
  +genlisting(label-toi Enumitem.to-arabic)(item);
>
```

`+toi` コマンドを用いて作成した箇条書きの例を下に示します.

問 **1.** こんな感じ.

問 **2.** こんなふうにできる.

4.2 数字の書式を変える

すでに見たとおり，デフォルトでは大きく分けてアラビア数字，ローマ数字，アルファベットの 3 種類に基づいた数字を用いてラベルを振ることができます．しかし，場合によってはそれ以外の書式が必要になることがあります．たとえばカタカナを用いて「(ア), (イ), (ウ), ...」といったインデックスは日本語の文書においてよく見かけます.

4.3 自由な箇条書きを定義する

上では既存のコマンドを使い回す方法から説明しましたが，これは実装を少し楽にするためのやり方であって，本来は `context -> int -> inline-boxes` 型の関数さえ用意すれば，自身の好きなラベルに付け替えることができます．以下の例は，テキストそのものを変えるのではなく，テキストの処理文脈を変更した例です．ユーザー定義された関数も通常の指定子と全く同様に，`+genlisting` コマンドの第 1 引数として呼び出すだけで使うことができます.

```
let label-colerful ctx idx =
  let label-width = (get-font-size ctx) *' 3.0 in
  let r = (idx - 1) mod 4 in
  let clr = Color.rgb (float r /. 4.) .5 .5 in
  let ctx-colerful = ctx |> set-text-color clr in
  let ib-label = read-inline ctx-colerful {hoge.\ } in
  let (wid-label, _, _) = get-natural-metrics ib-label in
  inline-skip (label-width -' wid-label) ++ ib-label
```

```
+genlisting(label-colerful){
  * こんなことすらできてしまいます.
  * こんなことすらできてしまいます.
  * こんなことすらできてしまいます.
  * こんなことすらできてしまいます.
}
```

hoge. こんなことすらできてしまいます.

hoge. こんなことすらできてしまいます.

hoge. こんなことすらできてしまいます.

hoge. こんなことすらできてしまいます.

さらに言えば，インデックスの値に応じて関数の戻り値を変える必要すらありません．実際，番号の付かない箇条書き環境を実現するためのラベルは，いずれもインデックスにかかわらず常に一定のインラインボックス列を返すような関数によって実現されています．

4.4 箇条書きのネストの深さによってスタイルを変える

ネストの深さに応じてラベルのスタイルを変える，という体裁は比較的頻繁に見かけます．箇条書きコマンドの中で `\genlisting` コマンドを用いれば実現できなくはありませんが，折角 SATYSF$_{\mathrm{I}}$ ではネストされた箇条書きを表現するための糖衣構文が標準で用意されているので，できればそれを使って完結に書きたいところです．

enumitem パッケージでは，そのような需要に応えるために `+genlistings` コマンドを用意しています．`+genlistings` コマンドは `+genlisting` コマンドと異なり，第 1 引数には `int -> context -> int -> inline-boxes` 型の関数を指定します．これは，ネストの深さに対応する変数も追加した 3 変数の関数を渡すことでインデックスのスタイルを指定することを意味しています．これにより，ネストの深さによってスタイルを変える事ができるようになります．百聞は一見に如かず．例を見てみましょう．

```
+genlistings(fun depth ctx idx -> (
  match depth with
  | 0 -> label-arabic-paren ctx idx
  | 1 -> label-alph-bracket ctx idx
  | _ -> label-roman-dot    ctx idx
)){
```

```
  * hoge
  * fuga
    ** fuga1
      *** fuga11
      *** fuga12
    ** fuga2
}
```

このように，depth, ctx, idx の順に引数を受け取る関数を入れます[*2]．depth はネストの深さであり，0 から始まって入れ子の内部になればなるほど数字がインクリメントされていきます．ctx, idx はおなじみのテキスト処理文脈及びインデックスです．ここまで読んだ方にはすでに想像がついているかもしれませんが，出力結果は以下のようになります．

(1) hoge
(2) fuga
    [a] fuga1
        i. fuga11
        ii. fuga12
    [b] fuga2

上のコード例では分かりやすさを優先して少し冗長に書きました．もう少し簡潔に，以下のように書いても同じ結果が得られます．

```
+genlistings(fun depth -> (
  match depth with
  | 0 -> label-arabic-paren
  | 1 -> label-alph-bracket
  | _ -> label-roman-dot
)){
  * hoge
  * fuga
    ** fuga1
```

2 今回は紙面節約のためにプリアンブル部分で関数を定義するのではなく引数に無名関数を直接入れていますが，今までの `+genlisting` コマンドの説明のときと同様に，プリアンブルで定義された関数を入れることも当然可能です．

```
      *** fuga11
      *** fuga12
    ** fuga2
}
```

先程説明を後回しにしたデフォルトコマンド +enumerate も，genlistings コマンドによってネスト深さごとに体裁が変わるように定義されています．