Enumitem パッケージ

@monaqa

2020/02/24

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 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（2020 年 2 月 24 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]
- 定義リストを作成できる

1 2020 年 2 月現在，標準ではサポートされていません．

- ネストごとに箇条書きのスタイルを変更できる
- ユーザ自身がスタイルを容易に拡張できる

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． これは一般の箇条書きを生成するためのブロックコマンドであり，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が得られます．結果から分かる通り，番号付き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」で番号を振る，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． これを**スタイル指定子**と呼ぶことにします． 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，代わりに以下のような箇条書きが表示されます．

(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（詳細は第3.2節参照）．

`+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ためのコードと結果の例です．

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

- hoge
  - hoge1
  - hoge2
- fuga
- piyo

特に目新しいとは感じなかったでしょう．実際，これもまた第 1 引数を置き換えたものに過ぎません．LaTeX では番号付きの箇条書きとそうでない箇条書きとを環境名で区別しますが，enumitem パッケージでは統一的に扱うことができます．

これまでの例では，`+genlistings` の第 1 引数を変えることで容易く一つの文書中に複数のスタイルを実現できることを示してきました．しかし，実用的な文書で登場する箇条書きごとにスタイルを変えることは稀です．一つ一つの箇条書きにスタイルを設定するより，寧ろ同一のスタイルの箇条書きを簡素なコマンドで使い回したいことが多いでしょう．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)` と同様に振る舞うようにしたければ，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 +genlisting を用いて書いたもの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
}
```

先程述べた通り， これは enumitem パッケージの機能というよりも SATYSFI の言語仕様によるものです．したがって， ここでは let-block などのプリミティヴについての詳細な説明を行いません．コマンドの定義方法が気になる方は，SATYSFI の作者である Takashi Suwa 氏によって執筆された The SATYSFIbook を参照されることを推奨します．

なお，enumitem パッケージにおいても標準の itemize パッケージ同様に，+listing コマンドが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 を定義する際に用いられた指定子は label-bullet であり，+listing コマンドを用いると以下のようなスタイルで出力されます． この挙動はユーザが改めてプリアンブルで +listing を再定義することで上書きすることができます．

- hoge
  - hoge1
  - hoge2
- fuga
- piyo

なお，+enumerate コマンドも enumitem パッケージによって定義されていますが，+listing と比べて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

ここでは，enumitem パッケージが提供するコマンド及び指定子を列挙します．

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ものが定義されています．

- `+genlisting`
- `+genlistings`
- `\genlisting`
- `\genlistings`
- `\listing-from-block`
- `+gendescription`
- `\gendescription`

`+genlisting` は前章で散々説明したとおり，箇条書きを作成するための最も基本的なコマンドです．`+genlisting(style-indicator){item}` でスタイル指定した箇条書きを生成します．`+genlistings` はより細かい指定ができるコマンドです．具体的には，ネストの深さに応じてラベルのスタイルを変更できるようになります．ただし， 少しばかり enumitem パッケージの詳細（`style-indicator` がどんな型であるか， など）を知る必要があるため，説明は第4.3節に譲ります．

`\genlisting` 及び `\genlistings` はそれぞれ `+genlisting` 及び `+genlistings` のインラインコマンド版であり， インラインテキスト中に箇条書きを埋め込むことができます．また，`\listing-from-block` はブロックコマンドとして定義された箇条書きを，体裁はそのままにインラインコマンドに変換するときに便利なコマンドです． 例えば既に `+mylisting` というコマンドが定義済みであるとします．そのとき，mylisting のインラインコマンド版は以下のように作成できます．

```
let-inline \mylisting item = {
  \listing-from-block<+mylisting(item);>
}
```

`+gendescription` は，LaTeX でいうところの description 環境を実現するコマンドです． このコマンドに限っては他のコマンドとインターフェースが大きく異なるため， 後の3.3節にて使い方を述べます．

更に，デフォルトの箇条書き環境として，以下のコマンドが定義されています．標準の itemize パッケージと名前が同じですが，改ページオプションはついていません．

- `+listing`
- `+enumerate`
- `\listing`
- `\enumerate`
- `+description`
- `\description`

3.2 スタイル指定子一覧

スタイル指定子は，`+genlisting` コマンドの第 1 引数に入れることでインデックスを指定するためのパラメータです．どのような種類があるかについては，実際にインデックスを見たほうが早いでしょう．

- 番号付き箇条書き環境
  - 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`
- 番号なし箇条書き環境
  - `label-bullet`
  - `label-white-bullet`

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[*2]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，ユーザがスタイル指定子を新たに定義したりスタイル指定子へ明示的に引数を渡したりする必要は生じないため，基本的には列挙型と同様の感覚で扱うことができます．逆に，ユーザ側が `context -> int -> inline-boxes` 型の関数を定義すれば，それをそのまま新たなスタイル指定子として使うことができるということでもあります．

3.3 定義リストの作成

LaTeX や HTML において標準で用意されている箇条書きは 3 種類ありました．順序なしリスト（番号のない箇条書き），順序リスト（番号付き箇条書き），そして定義リストです．

順序なしリスト
: 順序を表すインデックスのない箇条書きです． 通常，LaTeX では itemize 環境，HTML では `<ul>` タグにより組まれます．

順序リスト
: 順序を表すインデックスのある箇条書きです．通常，LaTeX では enumerate 環境，HTML では `<ol>` タグにより組まれます．

2 すなわち，現在の item が最初から数えて何番目にあるか．

定義リスト
: 各インデックス毎に固有の単語が付与されている箇条書きです．通常，LaTeX では description 環境，HTML では <dl> タグにより組まれます．
  この説明で用いられているリストは定義リストです．

1 番目及び 2 番目のリストは SATySFi 標準の itemize パッケージでも実現することが出来ますし，enumitem パッケージでも実現できることは今までに述べたとおりです．それに対し，3 番目の定義リストを実現する手段は標準の itemize パッケージでは用意されていませんでした．

Enumitem では，定義リストを実現する +gendescription コマンドを用意しています．このコマンドを用いれば汎用性の高い定義リストを記述することができます．具体例を以下に示します．

```
+gendescription(|
  nextline = false;
  title-inner-gap = 10pt;
  inner-indent = (fun title-wid -> title-wid);
  title-func =
    (fun ctx title -> read-inline ctx {\textbf{#title;}} );
|){
  * 英語のパングラム
    ** The quick brown fox jumps over the lazy dog.

  * 日本語のパングラム
    ** いろはにほへとちりぬるを
       わかよたれそつねならむ
       うゐのおくやまけふこえて
       あさきゆめみしゑひもせすん
    ** いろはにほへとちりぬるを
       わかよたれそつねならむ
       うゐのおくやまけふこえて
       あさきゆめみしゑひもせすん
```

```
}
```

英語のパングラム　The quick brown fox jumps over the lazy dog.

日本語のパングラム　いろはにほへとちりぬるをわかよたれそつねならむうゐのおくやまけふこえてあさきゆめみしゑひもせすん
いろはにほへとちりぬるをわかよたれそつねならむうゐのおくやまけふこえてあさきゆめみしゑひもせすん

具体例を見れば分かる通り，`+genlisting` などのコマンドとはかなりインターフェースが異なります．`+gendescription` は 2 つの引数 `record` と `inner` をとり，`record` で description コマンドの体裁を設定し，`inner` に表示したい内容を並べます．

SATYSF$_{\mathrm{I}}$ には現在，LaTeX の `\item[label]` のように箇条書きの item 毎にラベルを変更するためのマークアップを提供していません． そこで，`+gendescription` コマンドでは `inner` 部分を以下のように定めることで，SATYSF$_{\mathrm{I}}$ の箇条書き構文のインターフェースに沿った定義リストを実現しました．

- 1 番目の深さの item（`*` が 1 つ並んでいるところ）に，定義リストの各 item に相当する見出しを書く．
- 各見出しの子に相当する item（`*` が 2 つ並んでいるところ）に，定義リストの本文を書く． 複数の子 item を同じ親の下に並べて書くことで， 複数の段落を記述することができる．

説明を読むよりも前述の具体例を見たほうが使用法が掴みやすいでしょう．

また，`+gendescription` コマンドでは第 1 引数でレコードを渡すことにより， 柔軟に定義リストを構成することができます． レコードの値は 4 種類あり， これらは全てオ

プションではなく必須引数です．

nextline　`bool` 型の値を指定する．
　各 item のタイトルと本文の間に改行を挟むかどうか．true ならば改行を行う．false ならば改行を行わない．

title-inner-gap　`length` 型の値を指定する．
　各 item のタイトルと本文の間に挿入する最小の空白幅． どちらかというと「各 item のタイトルの右に常においておく余白」というイメージが近い．

inner-indent　`length -> length` 型の関数を指定する．
　「タイトルの幅を引数として，本文インデント量を返す関数」を指定する．わざわざ関数を指定するようにしているのは，タイトル幅に応じてインデントを動的に変えられるようにするため．たとえば `(fun _ -> 10pt)` を指定すれば，本文ではタイトルの幅の長さに関わらず常に 10pt のインデントが行われる（ただし，タイトルが記述されている行を除く）．また，`(fun title-wid -> title-wid)` を指定すれば，タイトル幅（title-inner-gap を含む）と同じだけのインデントが行われる．

title-func　`context -> inline-text -> inline-boxes` 型の関数を指定する．
　「テキスト処理文脈及びタイトルのインラインテキストを引数として，タイトルのインラインボックス列を返す関数」を指定する． 最も簡単なのは `read-inline` 関数をそのまま指定すること．この関数内部で `\textbf` のコマンドを挟んだり，テキスト処理文脈をいじってフォントを変更したりすることで，各 item のタイトルだけを太字にする，といった処理が可能になる．

　定義リストを書くたびにこれら 4 つの要素が埋まったレコードを指定しなければならない面倒だ，と感じたかもしれません．尤もです．しかし例によって，プリアンブルで SATYSFI のコマンド定義を適切に行えば，面倒な記述を一度書くだけでよくなります．たとえばデフォルトで用意されている `+description` コマンドは以下のように定義されています．

```
let-block +description item = '<
  +gendescription(|
    nextline = true;
    title-inner-gap = 5pt;
    inner-indent = (fun title-wid -> 20pt);
    title-func = read-inline;
  |)(item);
>
```

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　簡単な例：「問 1.」のスタイルを定義する

まずは単純な例として，「**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．インデックスは太字であり，数字部分はインデックスによってインクリメントされていくものとします．

先程述べたように，スタイル指定子の正体は `context -> int -> inline-boxes` 型の関数でした．そこで，「テキスト処理文脈 `ctx` 及び インデックス値 `idx` が与えられたとき，指定されたテキスト処理文脈の中で太字の「問 (`idx`) を返す」ような関数を渡すことを考えます．そのような関数は以下のように定義できます．

```
let label-toi ctx idx =
  let it-num = embed-string (arabic idx) in
  let ib-label = read-inline ctx {\textbf{問 #it-num;.}\ } in
  ib-label
```

ただこのように定義するだけで，もう `+genlisting(label-toi)` の形で箇条書きを使えるようになります．折角使えるようになったので，練習問題でも出してみましょう．

問 1. 序数をインデックスに持つような箇条書きコマンド `+enumerate-ordinal` を作成してみよ．つまり，インデックス値が 1 のときは「1st」，2 のときは「2nd」，3 のときは「3rd」といったラベルが出力されるようにせよ．ただし，簡単のためにインデックス値 N は $N \leq 10$ を満たすとする．

問 2. インデックス値が 3 の倍数と 3 が付く数字のときだけアホなフォントになる箇条書きコマンド `+enabeatsumerate` を作成してみよ．なお，インデックス値 N は $N \leq 40$ を満たすとする．また，何をアホなフォントとするかについては個人で勝手に決めてよい．

第2章で「enumitem パッケージでは番号付き箇条書きと番号の付いていない箇条書きを統一的に扱うことができる」と述べました．実際，番号のついていないスタイル指定子も簡単に作成できます．ラベルをインデックス値によらないものにすればよいのですから，インデックス値についての定数関数を指定すればよいのです．ここでは例を示すことはしませんが，気になる方は是非 enumitem パッケージ内部の `label-bullet` の実

装を覗いてみてください．関数の内部で，第 2 引数 idx は一度も使われていません．

4.2　特定の形式の数字を使う

第3.2節で紹介したとおり，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，5 種類の to- 型関数が用意されています．

Enumitem.to-arabic	整数をアラビア数字の inline text に変換する．
Enumitem.to-roman	整数をローマ数字（小文字）の inline text に変換する．
Enumitem.to-Roman	整数をローマ数字（大文字）の inline text に変換する．
Enumitem.to-alph	整数をアルファベット（小文字）の inline text に変換する．
Enumitem.to-Alph	整数をアルファベット（大文字）の inline text に変換する．

この関数は，スタイル指定子をユーザ定義する際にも役立ちます．たとえば整数値を大文字のローマ数字へと変換するのはユーザがプリアンブルで定義するには多少面倒ですが，Enumitem.to-Roman 関数を使えば面倒な変換を代わりに行ってくれます．以下は，大文字のローマ数字を四角の枠で囲ったラベルの指定子の定義と，その定義を用いて生成された箇条書きの出力結果です．

```
let-inline ctx \simple-frame it =
  let pads = (2pt, 2pt, 2pt, 2pt) in
  let decos = HDecoSet.simple-frame-stroke 1pt (Color.black) in
  inline-frame-breakable pads decos (read-inline ctx it)

let label-Roman-framed ctx idx =
```

```
  let label-width = (get-font-size ctx) *' 3.0 in
  let it-num = Enumitem.to-Roman idx in
  let ib-label = read-inline ctx {\simple-frame{#it-num;}\ } in
  let (wid-label, _, _) = get-natural-metrics ib-label in
  inline-skip (label-width -' wid-label) ++ ib-label
```

Ⅰ こんなふうにできる.

Ⅱ こんなふうにできる.

Ⅲ こんなふうにできる.

Ⅳ こんなふうにできる.

先程の「問 1.」の例に比べて多少複雑になっているのは，ラベルを右揃えにし，本文の開始位置を揃えているためです．そのためにはラベルのインラインボックス列の横幅を取る必要がありますが，これは `get-natural-metrics` プリミティブで実現できます．

4.3 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[*3]．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
      *** fuga11
      *** fuga12
```

3 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．

```
    ** fuga2
}
```

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おり，以下のようなスタイルとなっています.

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