Enumitem パッケージ

@monaqa

目次

1. はじめに

本ドキュメントは，enumitem パッケージ (v2.0.0) の仕様および使い方について述べたものです．Enumitem は v1.x から v2.0 にバージョンアップする際にほぼ全てのコマンドの仕様を変

更しており，v1.x を使っている場合はこのドキュメントに書かれている内容では動きません．旧版のドキュメントを確認してください．

2. Enumitem パッケージの概要

Enumitem は，組版システム SATYSF$_{\mathrm{I}}$ 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．SATYSF$_{\mathrm{I}}$ 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 5 月 23 日現在）．具体的には，enumitem パッケージを使うことで以下のような恩恵を受けられます．

- 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
- 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

3. 基本的な使い方

Enumitem を使うには，当然ですがパッケージを読み込む必要があります．パッケージが正しくインストールされていれば，文書の冒頭に以下の 1 行を追加するだけで読み込むことができます．

```
@require: enumitem
```

Enumitem は標準パッケージと同様，`+listing` 及び `+enumerate` という箇条書きインターフェースを提供します．デフォルトでは以下のように箇条書きを書くことができます．

```
+listing{
  * hoge
  * fuga
    ** fuga1
      *** fuga11
      *** fuga12
```

1 2020 年 5 月現在，標準ではサポートされていません．

```
    ** fuga2
}
```

- hoge
- fuga
  - fuga1
    - fuga11
    - fuga12
  - fuga2

```
+enumerate{
  * hoge
  * fuga
    ** fuga1
      *** fuga11
      *** fuga12
    ** fuga2
}
```

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

このように番号つき箇条書き環境のネストもサポートしており，ネストの深さによってインデックスの体裁を変えることができます.

また，文章の途中で箇条書きをはさむ場合は `\listing` や `\enumerate` コマンドを使うことで改段落を伴わずに箇条書きを挿入できます.

```
+p{
  寿限無，寿限無，五劫のすりきれ，
  海砂利水魚の
  \listing{
    * 水行末
    * 雲来末
    * 風来末
  }
  食う寝るところに住むところ ...
}
```

寿限無，寿限無，五劫のすりきれ，海砂利水魚の

- 水行末
- 雲来末
- 風来末

食う寝るところに住むところ ...

4. +listing における体裁の指定

ここまでの使用法は標準パッケージと大きく変わりませんでしたが，enumitem パッケージが提供する +listing などのコマンドではインデックスの体裁を指定されたものに変更することができます.

4.1. オプション引数を用いたインデックスの体裁指定

+listing 及び +enumerate では， 以下のように 1 つのオプション引数を受け付けることができます.

```
+listing?:(Enumitem.white-bullet){
  * hoge
    ** hoge1
    ** hoge2
  * fuga
  * piyo
```

```
}
```

- hoge
  - hoge1
  - hoge2
- fuga
- piyo

```
+enumerate?:(Enumitem.dot-arabic){
  * hoge
    ** hoge1
    ** hoge2
  * fuga
  * piyo
```

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

このように，オプション引数に応じてインデックスの体裁を変更することができます．上コードにおける `white-bullet` や `dot-arabic` というのは，`Enumitem` と呼ばれるモジュール内で定義されている関数であり，`idxfmt` 型を持ちます．モジュール内で定義されている `idxfmt` 型の関数は他にも多数存在し，それらをオプション引数に渡すことで箇条書きを模様替えできるという仕組みです．

なお，実際のマークアップで変数名の前に `Enumitem.` を付けるのは多少面倒と感じられるかもしれません．このような場合，`Enumitem` モジュールを open することで名前空間を省略することができます．

```
@require: enumitem
```

```
open Enumitem
in
document(| 中略 |)'<
  +listing?:(white-bullet){
    * hoge
    * fuga
  }
>
```

このように書くと，Enumitem モジュール内で定義された関数の前に Enumitem. を付ける必要がなくなります． その代わり， 他のパッケージやユーザで定義した関数名と衝突しないようにするのは使用者の責任となります．Enumitem パッケージでは，+listing や +enumerate といったコマンドについては名前空間を省略できるようにしていますが，raw-arabic などの関数名は open しない限りモジュール名を省略できないようにしています．

また，箇条書きを書くときにいちいち同じオプション引数を付けるのは面倒だと感じられるかもしれません． その場合は最初に +listing のデフォルトの挙動を変更してもよいでしょう． プリアンブル部分に以下のように書くと， 以下のコードでは +listing を用いた場合は常に白丸がラベルとして使われるようになります．

```
let-block +listing item = '<
  +Enumitem.listing?:(Enumitem.white-bullet)(item);
>
```

余談ですが，+listing と +enumerate の違いはオプション引数を省略したときのデフォルトの体裁のみであり， オプション引数を設定すると内部での処理は全く同一となります．したがって，+listing?:(Enumitem.dot-arabic) のように +listing で番号付き箇条書きを書くことも，+enumerate?:(Enumitem.bullet) のように +enumerate で番号のない箇条書きを書くことも可能ではあります[*2]．

4.2. インデックスの体裁を定める変数名一覧

以下は Enumitem モジュール内で定義されている体裁指定用変数の一覧です．番号付きの箇条書きを行いたい場合は，以下の変数を使用できます．

2 もちろん仕組み上そうなっているというだけであり，番号のない箇条書きには +listing を，番号付きには +enumerate を，などと使い分けたほうが可読性は向上するでしょう．

- アラビア数字系

  1 raw-arabic

  2. dot-arabic

  (3) paren-arabic

  [4] bracket-arabic

- ローマ数字系

  i raw-roman

  II raw-Roman

  iii. dot-roman

  IV. dot-Roman

  (v) paren-roman

  (VI) paren-Roman

  [vii] bracket-roman

  [VIII] bracket-Roman

- アルファベット系

  a raw-alph

  B raw-Alph

  c. dot-alph

  D. dot-Alph

  (e) paren-alph

  (F) paren-Alph

  [g] bracket-alph

  [H] bracket-Alph

また，番号の無い箇条書きには以下の変数を使用できます．

- bullet
- white-bullet

4.3. インデックスの体裁をユーザ定義する方法

ここまでの説明で，「dot-arabic 等は idxfmt 型の関数だ」 と述べましたが， そもそも

idxfmt 型とは何でしょうか．実は idxfmt 型とは int -> int list -> context -> inline-boxes という型のシノニムであり，

- 箇条書きの深さ (int)
- 箇条書きのインデックス (int list)
- 現在の本文のテキスト処理文脈 (context)

を入力として，箇条書きのインデックスを描画するインラインボックス列 (inline-boxes) を返す関数を表すものです．これは必ずしも予めモジュール内で用意された関数である必要はありません．つまり，idxfmt 型を持つ関数を自作して +listing コマンドのオプション引数に渡せば，自分の定義したインデックス表示を持つ箇条書きを作成することができます．

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 dpt (idx :: _) ctx =
  let it-num = embed-string (arabic idx) in
  let ib-label = read-inline ctx {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3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．

第4.2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.

4.4. 便利な関数

ネストの深さに応じて箇条書きのラベルを変えたい，という要求は珍しくありません．前節で紹介したようにインデックスの体裁を自身で定義すれば当然そういった箇条書きも作成できますが，change-by-depth 関数を用いることでもっと簡単に実現できます.

```
+listing?:(
  change-by-depth [bracket-Alph; white-bullet; paren-arabic]
){
  * hoge
  * fuga
    ** fuga1
      *** fuga11
      *** fuga12
    ** fuga2
}
```

[A] hoge

[B] fuga

- fuga1
  (1) fuga11
  (2) fuga12
- fuga2

5. 動的なフラグを用いたラベル操作

5.1. +xgenlisting コマンド

たとえば，以下のような To-Do リストを作成したくなったとします．

□ ミルクを買う．

□ The SATYSFIbook を読む．

☑ SATYSFI を完全に理解する．

□ 課題を解く．

ここで， ラベルを作成する関数は既に用意されているとします． たとえば以下の関数 `square-label is-checked ctx` は，第 1 引数 `is-checked` が true のときチェック済みの，false のときチェック済みでないチェックボックスを描画する関数です．

```
let square-label is-checked ctx =
  let fs value = (get-font-size ctx) *' value in
  let fsize = fs 1.0 in
  let gr-square (x, y) =
    stroke 0.5pt Color.black
      (Gr.rectangle (0pt, 0pt) (fs 0.5, fs 0.5 ))
      |> shift-graphics (x, y +' fs 0.15)
  in
  let gr-mark-done (x, y) =
    stroke 0.5pt Color.black (
      Gr.poly-line (fs (0.0 -. 0.1), fs 0.45)
```

```
        [(fs 0.15, fs 0.15); (fs 0.75, fs 0.85)])
        |> shift-graphics (x, y +' fs 0.15)
  in
  let gr point =
    if is-checked then
      [gr-square point; gr-mark-done point]
    else
      [gr-square point]
  in
  inline-skip 10pt ++ (inline-graphics fsize fsize 0pt gr)
```

今までに紹介された +genlisting を用いても，上のように「3 番目のみチェックが付いたリスト」を実現することはできます．以下のようにパターンマッチや if 文などを用いて，チェックを付けたい場所のみ場合分けして処理すればよいのです．

```
+genlisting(fun depth idx ctx -> (
  let checked = match idx with
    | 3 -> true
    | _ -> false
  in
  square-label checked ctx
))(default-item){
  * ミルクを買う.
  * The \SATySFi;book を読む.
  * \SATySFi; を完全に理解する.
  * 課題を解く.
}
```

しかしこれはあまり直観的ではなく，また編集もしやすいとはいえません．「ミルクを買う」にチェックを付けたいとき，「ミルクを買う」というテキストから離れた場所を編集しなければならないのは手間ですし，「どこにチェックが付いているのか」をひと目で判断することができません．また，今回はアイテムの数が 4 つだったため数えるのも楽でしたが，もっと長いリストの途中にチェックをつけるためにいちいち数えなければならないのは手間です．このように，体裁指定用変数を用いたレイアウトの指定は最初から最後まで一貫した

規則を持つ箇条書きの生成には適しているものの，例外があったり，動的にレイアウトを変更したかったりするケースにはあまり適していません．

Enumitem パッケージではこのようなケースに対応するため，本文中でパラメータに値をセットし，その値をラベルに反映させる方法を提供しています．たとえば先程の To-Do リストは，プリアンブルにて以下のように定義された `+todo-list` 及び `\done` コマンドで作成したものです．

```
let done = EnumitemBase.make-param false
let-inline \done = {\set-item(done)(true);}
let-block +todo-list item = '<
  +xgenlisting(
    fun depth idx ctx -> square-label (Param.get done) ctx
  )(default-item)(item);
>
```

この `+todo-list` 及び `\done` コマンドを使うと，先程の箇条書きは以下のようにシンプルに書くことができます．

```
+todo-list{
  * ミルクを買う.
  * The \SATySFi;book を読む.
  * \done; \SATySFi; を完全に理解する.
  * 課題を解く.
}
```

- ☐ ミルクを買う.
- ☐ The SATySFibook を読む.
- ☑ SATySFi を完全に理解する.
- ☐ 課題を解く.

このように，本文に `\done;` というコマンドが付いているアイテムに限って，チェックボックスにチェックマークが付くようになります．`\done` の位置はどこにあってもかまいません．

何が起きたのか，もうすこし詳細に説明します．ポイントは以下の 4 点です．

- 箇条書きのアイテム内で一時的な値を保持するための器（パラメータ）を定義することができる．
- パラメータには，本文中で `\set-item(param)(value);` と打つことで一時的に値を設定することができる．
- `+xgenlisting` を使うと，第 1 引数でラベルのスタイルを指定する際に，`Param.get` 関数でパラメータの値を受け取り， その値に応じてラベルの出力を変更することができる．
- `\set-item` で変更した値はそのアイテムが終わると破棄され，デフォルト値にもどる．

今回定義した `done` パラメータは bool 型を持つ値でしたが， 実際には `int`，`inline-text`，`int -> int -> context -> inline-boxes` 型など様々な型を持つパラメータを作成することができます． したがって， 上記の例のようにフラグとして使うだけではなく， パラメータの値に応じてインデックスの値を変更したり，ラベルそのもののレイアウトを変えたり，と多彩なカスタマイズが可能となります．

5.2. +xgenlisting の注意点

`+xgenlisting` は便利なコマンドですが，`+genlisting` にはない欠点が存在します． それは「本文に副作用のあるコマンドを入れると（おそらく）不具合が起きる」ということです．たとえば， 内部でカウンタをインクリメントさせる `\footnote` を `+genlisting` の本文で用いると，（実装にもよるとは思いますが）脚注の数字が 2 つインクリメントされる可能性があります．

理由は実装上の事情にあります． 実は，「本文中にあるコマンドを読み取ってラベルに反映する」という行為は少し不自然なのです． なぜなら本来，ラベルを組み終わってはじめて後続する本文のテキスト幅が分かり，本文を組むことができるようになるからです． ラベルを組まないと本文が組めない，しかし本文を読まないとラベルを組めない，というのが本実装を困難にする点でした．

そこで，`+xgenlisting` ではその問題を解決するため，「ラベルを組む前に `read-inline` で本文を読み，読み終わったものは一度破棄する」という方式で実装を行いました．本文中のコマンドは `read-inline` で読まれることにより評価され，`\set-item` がある場合はパラメータが一時的な値へとセットされます． その後であれば既にユーザ指定が分かっているためラベルを組むことができ，ラベルを組んでから再度改めて本文を組むことができます．

そしてこの方法だと （お気付きの通り）， 本文は `read-inline` によって 2 度評価されます． 副作用のないコマンドであれば何度評価されても結果は変わりませんから問題あり

ませんが，副作用のあるコマンド，特に 1 度だけ呼ばれることを想定しているコマンドを `+xgenlisting` の中に入れると，このことによって挙動がおかしくなるのです．

6. +gendescription コマンド

工事中

7. おわりに

バグ報告・ PR などお待ちしています．