まえがき

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊ **注意** ＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

このドキュメントは、Re:VIEW のサンプル原稿を兼ねています。自分の原稿を書くときは、サンプルの原稿ファイルと画像ファイルを消してください。

▼サンプルファイル（原稿と画像）の消し方（コマンドラインを知らない人はごめん！）

```
$ ls -1 *.re                    # 原稿ファイルの一覧
chap00-preface.re               #       まえがき
chap01-starter.re               #       第1章 Starterの機能
chap02-faq.re                   #       第2章 FAQ
chap99-postscript.re            #       あとがき
$ rm *.re                       # 原稿のファイルを消す
$ rm -r images/chap0?-*         # 画像ファイル（が入ったディレクトリ）も消す
$ vi catalog.yml                # 各章のファイル名を変更する
```

本文での、1 行あたりの文字数の確認：
一二三四五六七八九十一二三四五六七八九十一二三四五六七八九十一二三四五六七八九十一二三四五六七八九十一二三四五六七八九十

埋め込みコードでの、1 行あたりの文字数の確認：

```
 123456789_123456789_123456789_123456789_123456789_123456789_12345›
›6789_123456789_
一二三四五六七八九十一二三四五六七八九十一二三四五六七八九十一二三四五六七八九十
```

＊＊＊＊＊＊＊＊＊＊＊ 以下、まえがきのサンプル ＊＊＊＊＊＊＊＊＊＊＊

本書を手に取っていただき、ありがとうございます。

本書は、XXX についてわかりやすく解説した本です。この本を読めば、XXX の基礎的な使い方が身につきます。

■本書で得られること

- XXX についての基礎的な使い方

■対象読者

- XXX について興味がある人
- XXX の入門書を読んだ人（まったくの初心者は対象外です）

■前提知識

- Linux についての基礎知識
- 何らかのプログラミング言語の基礎知識

■問い合わせ先

- URL: https://www.example.com/
- Mail: support@example.com
- Twitter: @example

目次

第1章
Re:VIEW Starterの独自機能

素の Re:VIEW と比べて、Re:VIEW Starter（以下「Starter」とする）はいろんな機能強化やバグフィックスをしています。この章では、それらについて解説します[*1]。

また本章では Re:VIEW の機能や LaTeX の用語が説明なく使われます。これらが分からなければ読み飛ばして、分かる箇所だけ読んでください。

1.1 原稿本文を書くための機能

強調

- 「@<b>{...}」は、明朝体のまま太字になります（Re:VIEW のデフォルト）。
- 「@{...}」は、ゴシック体になります（Starter 拡張）。
- 「@{...}」は、太字のゴシック体になります（Starter 拡張）。
- 「@<B>{...}」は、「@{...}」のショートカットです（Starter 拡張）。

▼サンプル

```
いろはfghij　@<b>{いろはfghij}　@<em>{いろはfghij}　@<strong>{いろはfghij}
@<B>{いろはfghij}
```

表示結果：

いろは fghij **いろは fghij** いろは fghij **いろは fghij** **いろは fghij**

なおソースコードの一部を太字にしたいときは、「@{...}」ではなく「@<b>{...}」を使ってください。なぜなら「@{...}」だとゴシック体

[*1] 本章では、Re:VIEW や LaTeX についての説明はしません。Re:VIEW の書き方についてはマニュアル（https://github.com/kmuto/review/blob/master/doc/format.ja.md）やチートシート（https://qiita.com/froakie0021/items/b0f4ba5f242bbd571d4e）を参照してください。

になってしまうのに対し、「@<b>{...}」だとタイプライタ体のまま太字になるからです。

▼サンプル

```
//emlist{
タイプライタ体（通常）： 0123456789 ijkl IJKL !"$%&()*,-./:;?@[\]`|~
タイプライタ体（太字）： @<b>{0123456789 ijkl IJKL !"$%&()*,-./:;?@[\\]›
›`|~}
ゴシック体　　（太字）： @<strong>{0123456789 ijkl IJKL !"$%&()*,-./:;?›
›@[\\]`|~}
//}
```

表示結果：

タイプライタ体（通常）： 0123456789 ijkl IJKL !"$%&()*,-./:;?@[\]`|~
タイプライタ体（太字）： **0123456789 ijkl IJKL !"$%&()*,-./:;?@[\]`|~**
ゴシック体　　（太字）： 0123456789 ijkl IJKL !"$%&()*,-./:;?@[\]`|~

目立たせないための「@<weak>{}」

強調とは逆に、テキストを目立たせないための「@<weak>{}」という命令も用意しました。いわゆる「おまじない」のコードを目立たなくさせるときに使うといいでしょう。

次のは PHP のおまじないを目立たなくした例です。

▼サンプル

```
//list[][PHPサンプルコード]{
@<weak>{<?php}
function fib($n) {
    return $n <= 1 ? $n : fib($n-1) + fib($n-2);
}
@<weak>{?>}
//}
```

表示結果：

▼PHP サンプルコード

```
<?php
function fib($n) {
    return $n <= 1 ? $n : fib($n-1) + fib($n-2);
```

```
}
?>
```

次のは Java のおまじないを目立たなくした例です。

▼サンプル

```
//list[][Javaサンプルコード]{
@<weak>$public class Example {$
    @<weak>$public static void main(String[] args) {$
        System.out.println("Hello!");
    @<weak>$}$
@<weak>$}$
//}
```

表示結果：

▼Java サンプルコード

```
public class Example {
    public static void main(String[] args) {
        System.out.println("Hello!");
    }
}
```

番号つきリストの機能強化

Re:VIEW では番号つきリストを次のように書きます。

▼サンプル

```
 1. XXX
 2. YYY
 3. ZZZ
```

表示結果：

1. XXX
2. YYY
3. ZZZ

この書き方には次の欠点があります。

- 数字の番号はつきますが、「A.」や「a.」などは使えません。
- また番号つきリストを入れ子にできません。

そこで Starter では別の書き方を用意しました。

▼サンプル

```
数字による番号つきリスト

 - 1. XXX
 - 2. YYY
 - 3. ZZZ

大文字による番号つきリスト

 - A. XXX
 - B. YYY
 - C. ZZZ

小文字による番号つきリスト

 - a. XXX
 - b. YYY
 - c. ZZZ
```

表示結果：

数字による番号つきリスト

1. XXX
2. YYY
3. ZZZ

大文字による番号つきリスト

A. XXX
B. YYY
C. ZZZ

小文字による番号つきリスト

a. XXX
b. YYY
c. ZZZ

「1.」や「A.」や「a.」のあとに必ず半角空白が必要です。実は半角空白があれば、そ

の前に書いた文字列がそのまま出力されます。なので次のような書き方もできます。箇条書きのように見えますが、「・」がついてないことに注意してください。

▼サンプル

```
- (A) 項目A
- (B) 項目B
- (C) 項目C

- 甲：山田太郎
- 乙：佐藤花子
```

表示結果：

(A) 項目 A
(B) 項目 B
(C) 項目 C

甲: 山田太郎
乙: 佐藤花子

また入れ子にできます。

▼サンプル

```
- (A) 作業A
-- (A-1) 作業A-1
-- (A-2) 作業A-2
```

表示結果：

(A) 作業 A
  (A-1) 作業 A-1
  (A-2) 作業 A-2

箇条書きとの混在もできます。

▼サンプル

```
番号つきリスト の中に箇条書き

- A. XXX
```

```
 ** xxx
 ** xxx

箇条書きの中に番号つきリスト

 * XXXX
 -- a. xxx
 -- b. xxx
```

表示結果：

番号つきリストの中に箇条書き

A. XXX
   - xxx
   - xxx

箇条書きの中に番号つきリスト

- XXXX
  a. xxx
  b. xxx

なお数字や大文字や小文字の順番を補正するようなことはしません。たとえば「1.」を連続して書けばそれがそのまま出力されます。

▼サンプル

```
 - 1. XXX
 - 1. YYY
 - 1. ZZZ
```

表示結果：

---

1. XXX
1. YYY
1. ZZZ

ノート

「//note[...]{ ... //}」で、付加情報や注意書きのブロックが書けます。Re:VIEW 標準と比べると、デザインを大きく変更していることと、段落の先頭は 1 文字分インデ

ントされる点が違います。

▼サンプル

```
//note[■注意：印刷所の締切り日を確かめること]{
印刷所の締切りは、技術書典のようなイベントの本番当日よりずっと前です。
通常は約1週間前、また割引きを受けようと思ったら約2週間前が締切りです。
実際の締切り日は印刷所ごとに違うので、必ず確認しておきましょう。

また他の人に原稿のレビューを頼む場合は、さらに1〜2週間必要です。
これも忘れやすいので注意しましょう。
//}
```

表示結果：

■注意：印刷所の締切り日を確かめること

印刷所の締切りは、技術書典のようなイベントの本番当日よりずっと前です。通常は約 1 週間前、また割引きを受けようと思ったら約 2 週間前が締切りです。実際の締切り日は印刷所ごとに違うので、必ず確認しておきましょう。

また他の人に原稿のレビューを頼む場合は、さらに 1〜2 週間必要です。これも忘れやすいので注意しましょう。

実は Re:VIEW では、ノートの中に箇条書きや他のブロック命令を含められません。これは技術同人誌や書籍の執筆において、大変困る欠点です。

なので Starter ではこれを解決し、ノートの中に箇条書きや他のブロック命令を含められるようにしました[*2][*3]。

▼サンプル

```
//note[■ノートサンプル]{

箇条書きを含める例@<fn>{t71o9}。

 * エマ
```

[*2] 以前はこれができなかったので、「====[note] ... ====[/note]」という別の記法が必要でした。今でもこの記法は有効ですが、もう使う必要はありません。

[*3] 昔はノート中のプログラム（「//emlist」や「//cmd」）やターミナル（「//terminal」）がページをまたげないという制限がありましたが、現在はその制限はなくなりました。

```
 * レイ
 * ノーマン

//footnote[t71o9][ノートの中に脚注を含めるサンプル。]

他のブロックを含める例。

//emlist[RubyでHello]{
def hello(name)
  print("Hello, #{name}!\n")
end
//}

//cmd[UNIXでHello]{
$ echo Hello
Hello
//}

//}
```

表示結果：

■ノートサンプル

箇条書きを含める例[*4]。

- エマ
- レイ
- ノーマン

他のブロックを含める例。

▼Ruby で Hello

```
def hello(name)
  print("Hello, #{name}!\n")
end
```

▼UNIX で Hello

```
$ echo Hello
Hello
```

[*4] ノートの中に脚注を含めるサンプル。

なお「//note」機能は Re:VIEW の標準機能であり、Starter はそれを上書きしています。実は Re:VIEW の標準のままだと、次のような表示になります。

▼サンプル

```
//note[印刷所の締切り日を確かめること]{
印刷所の締切りは、技術書典のようなイベントの本番当日よりずっと前です。
通常は約1週間前、また割引きを受けようと思ったら約2週間前が締切りです。
実際の締切り日は印刷所ごとに違うので、必ず確認しておきましょう。

また他の人に原稿のレビューを頼む場合は、さらに1〜2週間必要です。
これも忘れやすいので注意しましょう。
//}
```

表示例（Re:VIEW のデフォルト）：

■メモ: 印刷所の締切り日を確かめること

印刷所の締切りは、技術書典のようなイベントの本番当日よりずっと前です。通常は約 1 週間前、また割引きを受けようと思ったら約 2 週間前が締切りです。実際の締切り日は印刷所ごとに違うので、必ず確認しておきましょう。
また他の人に原稿のレビューを頼む場合は、さらに 1〜2 週間必要です。これも忘れやすいので注意しましょう。

段落の先頭がインデントされてないことが分かります。また、ノート（「//note」）なのになぜかキャプションが「■メモ：」になってる！ おかしいですよね。詳しくは「2.3 ブロック命令」内の「「//info{ ... //}」のキャプションに「■メモ：」がつくんだけど？」(p.58) を参照のこと。

プログラムコード用のコマンドを統一

Re:VIEW では、プログラムコードを書くためのブロックコマンドが複数あります。

//list[*ID*][*caption*][*lang*]

リスト番号あり、行番号なし

//emlist[*caption*][*lang*]

　　リスト番号なし、行番号なし

//listnum[*ID*][*caption*][*lang*]

　　リスト番号あり、行番号あり

//emlistnum[*caption*][*lang*]

　　リスト番号なし、行番号あり

Starter では、これらをすべて「//list[][][]」に統一しました。それ以外のコマンドは、実質的に「//list[][][]」へのエイリアスとなります[*5]。

- 第 1 引数が空だと、「リスト X.X:」のような番号がつきません。つまり「//emlist」と同じです。
- 第 3 引数に「lineno=on」をつけると、行番号がつきます。つまり「//listnum」と同じです。
- 第 1 引数を空にし、第 3 引数に「lineno=on」をつけると、リスト番号がつかず行番号がつきます。つまり「//emlistnum」と同じです。

▼サンプル

```
//list[4k2ny][リスト番号あり]{
def fib(n)
  n <= 1 ? n : fib(n-1) + fib(n-2)
end
//}

//list[][リスト番号なし]{
def fib(n)
  n <= 1 ? n : fib(n-1) + fib(n-2)
end
//}

//list[970bl][リスト番号あり、行番号あり][lineno=on]{
def fib(n)
  n <= 1 ? n : fib(n-1) + fib(n-2)
end
//}

//list[][リスト番号なし、行番号あり][lineno=on]{
def fib(n)
  n <= 1 ? n : fib(n-1) + fib(n-2)
end
```

[*5] 「//emlist」や「listnum」が使えなくなったわけではありません。これらも引き続き使えますが、動作は「//list」を呼び出すだけになりました。

```
//}
```

表示結果：

▼リスト 1.1: リスト番号あり

```
def fib(n)
  n <= 1 ? n : fib(n-1) + fib(n-2)
end
```

▼リスト番号なし

```
def fib(n)
  n <= 1 ? n : fib(n-1) + fib(n-2)
end
```

▼リスト 1.2: リスト番号あり、行番号あり

```
1 def fib(n)
2   n <= 1 ? n : fib(n-1) + fib(n-2)
3 end
```

▼リスト番号なし、行番号あり

```
1 def fib(n)
2   n <= 1 ? n : fib(n-1) + fib(n-2)
3 end
```

リスト番号もキャプションも行番号もつけない場合は、すべての引数を省略して「`//list{ ... //}`」のように書けます。この書き方は Re:VIEW ではエラーになりますが、Starter ではエラーになりません。

▼サンプル

```
//list{
function fib(n) {
    return n <= 1 ? n : fib(n-1) + fib(n-2);
}
//}
```

表示結果：

```
function fib(n) {
    return n <= 1 ? n : fib(n-1) + fib(n-2);
}
```

また「//list」の第 3 引数には、以下のオプションが指定できます。

fold={on|off}
: 長い行を自動で折り返します（詳しくは後述）。デフォルトは on。

foldmark={on|off}
: 折り返したことを表す、小さな記号をつけます。デフォルトは on。

eolmark={on|off}
: すべての行末に、行末であることを表す小さな記号をつけます。「foldmark=on」のかわりに使うことを想定していますが、両方を on にしても使えます。デフォルトは off。

lineno={on|off|*integer*|*pattern*}
: 行番号をつけます。行番号は 1 から始まりますが、整数を指定するとそれが最初の行番号になります。またより複雑なパターンも指定できます（後述）。デフォルトは off。

linenowidth=*integer*
: 行番号の桁数を指定します（詳しくは後述）。0 だと自動計算します。値が 0 以上だと、行番号の分だけプログラムコードの表示幅が狭くなります。値がマイナスだと行番号はページの右余白に書かれるので、プログラムコードの表示幅が狭くなりません。デフォルトは-1。

lang=*name*
: プログラミング言語名を表します。デフォルトはなし。

いくつか補足事項があります。

- 複数のオプションを指定するときは、「,」で区切ってください。たとえば「//list[][][eolmark=on,lineno=on,linenowidth=3]」のようにします。
- オプションの名前だけを指定して値を省略すると、「on」を指定したとみなされます。たとえば「lineno」は「lineno=on」と同じです。
- 「lang=*name*」を指定してもコードハイライトはできません。この制限は将来改善される予定ですが、時期は未定です。
- 「lang=*name*」の場合は、省略形は「lang」ではなく「*name*」です[*6]。またこ

[*6] これは Re:VIEW との互換性を保つために仕方なく決めた仕様なので、できれば「lang={name}」

の省略ができるのは、第 3 引数の最初のオプションに指定した場合だけです。つまり、「ruby,lineno=1」は OK だけど「lineno=1,ruby」はエラーになります。

```
これはOK
//list[][][ruby,lineno=1]{
//}

これはエラー
//list[][][lineno=1,ruby]{
//}

これはOK
//list[][][lineno=1,lang=ruby]{
//}
```

ターミナル画面を表す「//terminal」ブロック

Starter では、ターミナル画面用の新しいブロック命令「//terminal{ ... //}」を用意しました[*7]。これは「//cmd{ ... //}」とよく似ていますが、オプションの指定方法が「//list{ ... //}」と同じになっています。

次の例を見てください。

- 「//cmd」はオプション引数としてキャプションしか取れません。そのためリスト番号をつけられないし、行番号もつけられません。

▼サンプル

```
//cmd[キャプション]{
$ echo foobar
foobar
//}
```

表示結果：

▼キャプション

```
$ echo foobar
foobar
```

と省略せずに書いてください。この省略のせいでオプション名が間違っていても言語名とみなされてしまうので注意してください。

[*7] //terminal 命令の定義は lib/hooks/monkeypatch.rb で行っています。

- 「//terminal」はオプション引数が「//list」と同じです。そのためリスト番号をつけたり、行番号をつけることが簡単にできます。

▼サンプル

```
//terminal[id6789][キャプション][lineno=on]{
$ echo foobar
foobar
//}
```

表示結果：

▼リスト 1.3: キャプション

```
1 $ echo foobar
2 foobar
```

なお Starter では、「//cmd」は実質的に「//terminal」を呼び出しているだけです。なので上で説明したこと以外では、両者の機能は同じです。

プログラムコード中の長い行を自動的に折り返す

Starter では、プログラムやターミナルの中の長い行を自動的に折り返します。

▼サンプル

```
//list[][長い行を含むプログラム例]{
data = <<HERE
123456789_123456789_123456789_123456789_123456789_123456789_12345›
›6789_123456789_123456789_123456789_
HERE
//}

//terminal[][長い行を含む出力例]{
$ ruby foo/bar/baz/testprog.rb
foo/bar/baz/testprog.rb:11:in `func1': undefined local variable o›
›r method `aaabbbccc' for main:Object (NameError)
//}
```

表示結果：

▼長い行を含むプログラム例

```
data = <<HERE
123456789_123456789_123456789_123456789_123456789_123456789_12345›
›6789_123456789_123456789_123456789_
HERE
```

▼長い行を含む出力例

```
$ ruby foo/bar/baz/testprog.rb
foo/bar/baz/testprog.rb:11:in `func1': undefined local variable o›
›r method `aaabbbccc' for main:Object (NameError)
```

いくつか注意事項があります。

- 折り返した行には、折り返したことを表す小さな記号がつきます。これをつけなくない場合は、「//list」や「//terminal」の第 3 引数に「foldmark=off」を指定してください。
- 折り返すはずの箇所が日本語の場合、折り返しを表す記号が挿入されません[*8]。日本語の途中で折り返しをしたい場合は、手動で「@<foldhere>{}」を挿入してください。
- 右端にまだ文字が入るスペースがあるのに折り返しされている（ように見える）場合があります。この場合、プログラムやターミナルの表示幅をほんの少し広げると、右端まで文字で埋まるようになります。詳しくは「2.4 ソースコード」内の「まだ文字が入りそうなのに折り返しされるのはなんで？」(p.63) を参照してください。
- 折り返し機能によって何らかの問題が発生したら、「//list」や「//terminal」の第 3 引数に「fold=off」を指定して折り返し機能をオフにしてください。これは原因の切り分けに役立つでしょう。

折り返し記号のかわりに行末記号

　折り返し箇所が日本語だと折り返し記号がうまく挿入されません。かといって手動で「@<foldhere>{}」を挿入するのも面倒です。

　そのような場合は、折り返し記号をオフにし、かわりに行末記号を入れることを検討してください。

[*8] 英数字なら折り返し改行される位置にハイフンが入ります。このハイフンを強引に置き換えることで、折り返し記号を挿入しています。しかし pLaTeX では日本語だとハイフンが入らないため、折り返し記号も挿入されません。これの解決は難しそうなので、別の方法を模索中。

次がその例です。折り返し記号はありませんが、行末記号があるので、行末記号がない箇所は折り返しされていることがわかります。

▼サンプル

```
//list[][][foldmark=off,eolmark=on]{
def emergency()
  abort '深刻なエラーが発生しました。今すぐシステム管理者に連絡して、
›対処方法を仰いでください。'
end
//}
```

表示結果：

```
def emergency()↲
  abort '深刻なエラーが発生しました。今すぐシステム管理者に連絡して、
対処方法を仰いでください。'↲
end↲
```

プログラムやターミナルの行番号を出力

Starter では、プログラムやターミナルに行番号をつけられます。

▼サンプル

```
//list[][][lineno=on]{
function fib(n) {
  return n <= 1 ? n : fib(n-1) + fib(n-2);
}
//}
```

表示結果：

```
1 function fib(n) {
2   return n <= 1 ? n : fib(n-1) + fib(n-2);
3 }
```

正の整数を指定すると、最初の行番号になります。

▼サンプル

```
//list[][][lineno=98]{
function fib(n) {
  return n <= 1 ? n : fib(n-1) + fib(n-2);
}
//}
```

表示結果：

```
98 function fib(n) {
99   return n <= 1 ? n : fib(n-1) + fib(n-2);
100 }
```

行番号の桁数を指定すると、行番号が余白ではなく内側に表示されます。その分、プログラムコードの表示幅が狭くなってしまいます。

▼サンプル

```
//list[][][lineno=98,linenowidth=5]{
function fib(n) {
  return n <= 1 ? n : fib(n-1) + fib(n-2);
}
//}
```

表示結果：

```
   98: function fib(n) {
   99:   return n <= 1 ? n : fib(n-1) + fib(n-2);
  100: }
```

行番号が灰色で表示されていることにも注目してください。こうすることで、行番号とプログラムコードとの見分けがつきやすくなっています。

行番号の桁数に 0 を指定すると、表示幅を自動計算します。

▼サンプル

```
//list[][][lineno=98,linenowidth=0]{
function fib(n) {
  return n <= 1 ? n : fib(n-1) + fib(n-2);
}
//}
```

表示結果：

```
 98: function fib(n) {
 99:   return n <= 1 ? n : fib(n-1) + fib(n-2);
100: }
```

長い行が折り返されたときは、折り返された行が左端からは始まらず、行番号の表示幅の分だけインデントされます。

▼サンプル

```
//list[][][lineno=1,linenowidth=2]{
data = <<HERE
123456789_123456789_123456789_123456789_123456789_123456789_12345›
›6789_123456789_123456789_123456789_
HERE
//}
```

表示結果：

```
 1: data = <<HERE
 2: 123456789_123456789_123456789_123456789_123456789_123456789_1›
  ›23456789_123456789_123456789_123456789_
 3: HERE
```

行番号を表す、より複雑なパターンを指定できます。

- 「1-10」なら、1 行目から 10 行目まで
- 「1-10&15-18」なら、1 行目から 10 行目までと、1 行空けて 15 行目から 18 行目まで
- 「1-10&15-」なら、1 行目から 10 行目までと、1 行空けて 15 行目から最終行まで

サンプルを見ればどういうことか分かるでしょう。

▼サンプル

```
//list[][][lineno=10&18-20&25-]{
class Hello
  ...(省略)...
  def initialize(name)
    @name = name
```

```
  end
  ...(省略)...
  def hello
    print("Hello #{@name}\n")
  end

end
//}
```

表示結果：

```
10 class Hello
     ...(省略)...
18   def initialize(name)
19     @name = name
20   end
     ...(省略)...
25   def hello
26     print("Hello #{@name}\n")
27   end
28
29 end
```

ラベル指定なしでリスト番号を出力

リスト番号つきでソースコードを表示する場合、「//list」の第 1 引数にラベルを指定します。

▼サンプル

```
//list[samplecode3][サンプル]{
puts "Hello"
//}
```

表示結果：

▼リスト 1.4: サンプル

```
puts "Hello"
```

このラベルは、重複しないよう気をつけなければいけません。リスト番号をあとから参照する場合は重複しないことが必要ですが、単にリスト番号をつけたい場合は重複し

ないラベルを選ぶのは面倒です。特に、すべてのソースコードにリスト番号をつけようと思った場合はかなりの手間になります。

そこで Starter では、「//list[?]」のように第 1 引数を「?」とするだけで、ラベルとしてランダムな文字列が割り当てられるようにしました[*9]。これにより、すべてのソースコードにリスト番号をつけるのが大幅に簡単になりました。

▼サンプル

```
//list[?][サンプル]{
puts "Hello"
//}
```

表示結果：

▼リスト 1.5: サンプル

```
puts "Hello"
```

この機能をサポートしているのは、次のブロック命令です。

- //list[?][*caption*] ... //}
- //listnum[?][*caption*] ... //}
- //terminal[?][*caption*] ... //}

キャプションなしでもリスト番号だけを出力

Re:VIEW では、キャプションがないとリスト番号もつかない仕様です。つまり「//list[][]」の第 1 引数を指定しても、第 2 引数が空ならリスト番号はつきません。キャプションなしでリスト番号だけをつけたい場合は、第 2 引数に全角空白を入れます。

Starter ではこの仕様を変更し、第 1 引数が指定してあれば第 2 引数が空（つまりキャプションが空）でもリスト番号をつけるようにしています。こちらのほうが仕様として自然です。

▼サンプル

```
//list[test7][]{
puts "Hello"
//}
```

[*9] 実装は lib/hooks/monkeypatch.rb の中で ReVIEW::Book::Compilable#content() を上書きして実現しています。

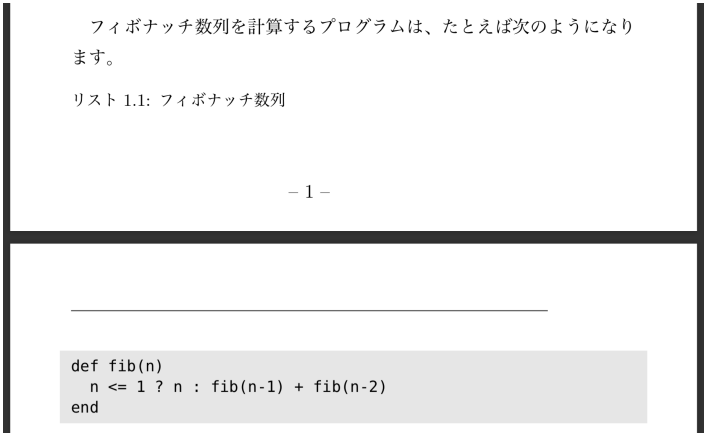
表示結果：

▼リスト 1.6:

```
puts "Hello"
```

プログラムのキャプション直後の改ページを抑制

Re:VIEW では、プログラムやターミナルのキャプション（説明）直後に改ページされてしまうことがあります（図 1.1）。もしこうなると、キャプションが何を説明しているのか分かりにくくなります。



▲図 1.1: キャプションの直後で改ページされた例

Starter ではこれを改善し、キャプションの直後では改ページを起こさないようにしました[*10]。かわりにキャプションの直前で改ページされます。

ただし同じページに脚注が複数行あると、判定を間違えてキャプション直後に改ページされることがあります。これは現在の制限事項です。経験則として、キャプションの前の文章を増やすとなぜか治ることが多いです。

[*10] これは LaTeX の needspace.sty で実現しています。

コラム内の脚注

Re:VIEW では、コラムの中に書いた脚注が消えることがあります。たとえば次のように書いた場合は、脚注が消えます。

▼コラム内の脚注が消えるサンプル

```
==[column] サンプル
本文本文@<fn>{xxx1}本文本文。

//footnote[xxx1][脚注脚注脚注。]
```

こうではなく、次のようにコラムを明示的に閉じてから脚注を書くと、消えずに表示されます。

▼コラム内の脚注が消えないサンプル

```
==[column] サンプル
本文本文@<fn>{xxx1}本文本文。

==[/column]

//footnote[xxx1][脚注脚注脚注。]
```

Starter ではこの問題に対処するために、前者のように書かれた場合でも、後者のように自動変換します。

変換はスクリプト lib/hooks/beforetexcompile.rb が行います。設定ファイルである config.yml に「hook_beforetexcompile: [lib/hooks/beforetexcompile.rb]」という設定を追加しているため、LaTeX コマンドでコンパイルされる前にこのスクリプトが実行されるようになっています。

右寄せ、左寄せ、センタリング

Starter では、右寄せや左寄せやセンタリングをする機能を追加しました。

▼サンプル

```
//textright{
右寄せのサンプル
//}
//textleft{
左寄せのサンプル
//}
//textcenter{
センタリングのサンプル
//}
```

表示結果：

右寄せのサンプル

左寄せのサンプル

センタリングのサンプル

しかし、実は Re:VIEW にも右寄せとセンタリングの機能があることが判明しました。今後はこちらを使うのがいいでしょう[*11]。

▼サンプル

```
//flushright{
右寄せのサンプル
//}
//centering{
センタリングのサンプル
//}
```

表示結果：

右寄せのサンプル

センタリングのサンプル

章の概要

Starter では、章 (Chapter) の概要を表す「//abstract{ ... //}」を用意しています。

▼サンプル

```
//abstract{
この章では、XXXのXXXという機能について説明します。
この機能を理解することで、あとの章が理解できるようになります。
//}
```

[*11] ただし https://github.com/kmuto/review/blob/master/doc/format.ja.md には載ってないので、undocumented な機能です。将来は変更されるかもしれません。

表示結果：

この章では、XXX の XXX という機能について説明します。この機能を理解することで、あとの章が理解できるようになります。

本文と違う見た目にするために、デフォルトでは左右に全角 2.5 文字文の余白を追加し、かつゴシック体で表示します。デザインを変更する場合は、sty/starter.sty で「`\newenvironment{starterabstract}`」を探し、変更してください。

なおこれとよく似た機能として、Re:VIEW には導入文（リード文）を表す「`//lead{ ... //}`」が標準で用意されています。これは主に、詩や物語や聖書からの引用を表すのに使うようです（海外の本にはよくありますよね）。そのため、「`//lead`」は LaTeX での引用を表す「`quotation`」環境に変換されます。

▼サンプル

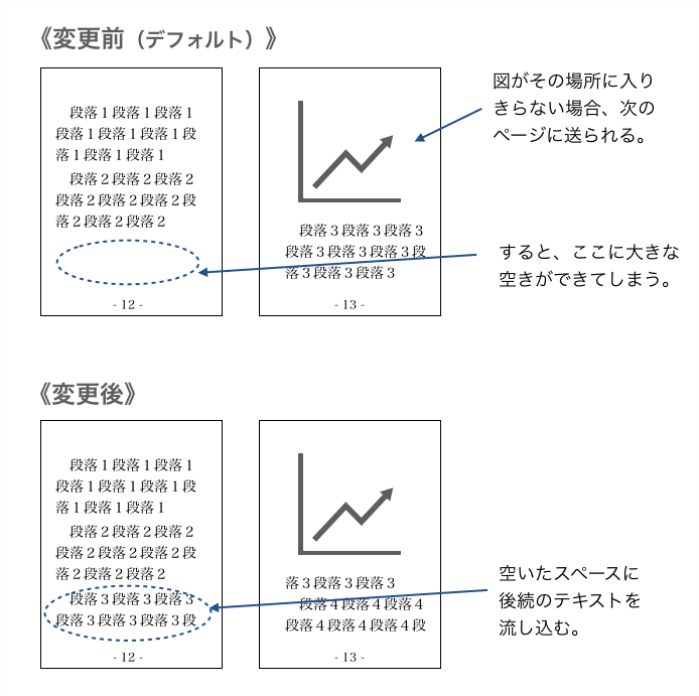
```
//lead{
土に根を下ろし　風と共に生きよう

種と共に冬を越え　鳥と共に春を歌おう
//}
```

表示結果：

土に根を下ろし　風と共に生きよう
種と共に冬を越え　鳥と共に春を歌おう

図が次のページに送られるときにスペースを空けない

Re:VIEW のデフォルトでは、図を入れるときに現在位置に入りきれない場合は、次のページに送られます。それは仕方ないのですが、このとき現在位置に大きな空きができてしまいます（図 1.2 の上）。



▲図 1.2: 図が次のページに送られると、そこに大きな空きができてしまう

これに対する解決策として、Starter では空いたスペースに後続のテキストを流し込む選択肢を用意しています（図 1.2 の下）。

そのためには、Starter のプロジェクト作成ページに「画像が現在位置に入りきらず次のページに回されるとき、大きなスペースを空けない （かわりに後続のテキストを流し込む）」というチェックボックスがあるので、これを選んでください。または、config-starter.yml で「`image_position:`」というオプションに「`h`」を指定してください。

また Starter では「`//image`」コマンドを拡張し、図の挿入位置が指定できるようになっています[*12]。これを指定することで、空いたスペースに後続のテキストを流し込むかどうかを、画像ごとに制御できます。

- 「`//image[][][`**`pos=H`**`]`」なら後続のテキストを流し込まない
  （つまり画像を現在位置に強制的に配置する）
- 「`//image[][][`**`pos=h`**`]`」なら後続のテキストを流し込む

[*12] 実装方法は lib/hooks/monkeypatch.rb を見てください。

（つまり画像が現在位置に入りきらなければ次のページの先頭に配置する）

画像の倍率も指定する場合は、「`//image[][][scale=0.5,pos=H]`」のように指定してください。

ページ下部にも画像を配置する

「`pos=H`」や「`pos=h`」のどちらを選んでも、入りきらない画像は次ページに送られます。そのため、どうしても画像はページ上部に配置されることが多くなり、逆にページ下部には配置されにくくなります。

このバランスの悪さが気になる場合は、（小さい画像を除いて大きめの画像に）「`pos=bt`」を指定してみてください。ここで「`b`」はボトム (bottom)、「`t`」はトップ (top) を表します。つまり、まずページ下部に配置を試み、入らないなら次ページ上部に配置します。これで、全体的に図がページの上部と下部の両方に配置されるはずです。

「次の図」や「以下の図」という表現を止める

すでに説明したように、画像の配置場所として「`pos=H`」**以外**を指定した場合は、後続のテキストが現在位置に流し込まれます。そのため、文章中で「次の図は〜」とか「以下の図では〜」と書いていると、図が次ページに配置された場合、読者が混乱します。

このような事態を避けるために、「次の図は〜」や「以下の図では〜」という表現を止めて、「図 1.1 では〜」のように番号で参照するようにしましょう。面倒でしょうが、仕方がありません。慣れてください。

図のまわりを線で囲む

Starter では、図のまわりをグレーの線で囲むことができます。そのためには「`//image`」の第 3 引数に「`border=on`」を指定します。

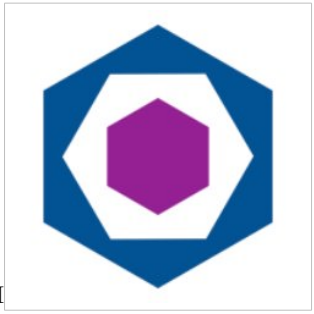
▼サンプル

```
//image[tw-icon][デフォルトの表示][scale=0.5,pos=H]

//image[tw-icon][まわりを線で囲む][scale=0.5,pos=H,border=on]
```

表示結果：

H

▲図 1.3: デフォルトの表示



H

▲図 1.4: まわりを線で囲む

何もしない命令「@<nop>{...}」

Re:VIEW では、「//list{ ... //}」や「//emlist{ ... //}」のようなブロック命令の中で、「@<b>{...}」などのインライン命令が利用できます。そのため、「//emlist{ ... //}」の中で「@<b>{...}」そのものを表示させるには次のようなトリックが必要です。

▼サンプル

```
//emlist{
  @<b>{ABC}
```

```
  @<code>$@$<b>{ABC}
//}
```

表示結果：

ABC
@<b>{ABC}

つまり「@<b>{...}」のうち「@」だけを「@<code>{...}」で囲うわけです。

この方法はうまく動作しますが、そもそもソースコードを表示するための「//emlist{ ... //}」の中で「@<code>{...}」を使うのもおかしな話です。

そこで Starter では、何もしないインライン命令「@<nop>{...}」を用意しました（「nop」は「No Operation」の略です）。これを使うと、引数を何も加工せず表示します。

サンプルを見てみましょう。

▼サンプル

```
//emlist{
1. @<b>{ABC}
2. @<nop>$@<b>{ABC}$
3. @<nop>$@$<b>{ABC}
//}
```

表示結果：

1. **ABC**
2. @<b>{ABC}
3. @<b>{ABC}

1 番目は、「ABC」が太字で表示されてしまいます。今は「@<b>{ABC}」と表示したいので、これは意図とは違います。

2 番目は、「@<b>ABC」と表示されました（波カッコの「{」と「}」が消えています）。実は LaTeX に変換された段階では「@<b>{ABC}」となっていますが、LaTeX では波カッコが特別な意味を持つため、「@<b>{ABC}」ではなく「@<b>ABC」と表示されてしまいます。

3 番目は、「@<b>{ABC}」と表示されました。これが期待した出力です。2 番目との違いは、「@<nop>$...$」の引数に「{」と「}」を含めなかったことです。これによって、LaTeX へ変換するときに「{」と「}」が適切にエスケープされます。

このような落とし穴はありますが、ブロック命令の中でインライン命令そのものを表示したい場合は、「@<nop>{...}」を使ってください。

なお「@<nop>{}」はもともと「@<letitgo>{}」という名前でしたが、長すぎるという意見があったので「@<nop>{}」になりました。後方互換性のため、「@<letitgo>{}」も使えます。

章や項を参照する「@<secref>{}」

Re:VIEW では、「@<hd>{}」を使って節 (Section) や項 (Subsection) を参照できます。しかしこの機能には問題点があります。

- Re:VIEW のデフォルト設定[*13]では、章 (Chapter) と節 (Section) には番号がつくけど、項 (Subsection) には番号がつかない。
- そのため、「@<hd>{}」で項 (Subsection) を参照すると、番号がなくて項タイトルだけになるので文章がとても不自然になる。

サンプルを使って説明しましょう。たとえば次のような原稿ファイルがあったとします。

▼ファイル：chap-pattern.re

```
= デザインパターンの紹介

==[sec-visitor] Visitorパターン

===[subsec-motivation] 動機

===[subsec-structure] 構造

===[subsec-impl] 実装例
```

文章の構造は次のようになっていますね。

- 「デザインパターンの紹介」は章 (Chapter)
  - 「Visitor パターン」は節 (Section)
    - 「動機」と「構造」と「実装例」は項 (Subsection)

さて Re:VIEW のデフォルト設定のままだと、次のように章と節には番号がつくけど、項には番号がつきません。

表示例：

[*13] Re:VIEW のデフォルトでは config.yml で「secnolevel: 2」と設定されています。これが 3 以上でないと、項 (Subsection) に番号がつきません。

第 1 章　デザインパターンの紹介

1.1　Visitor パターン

動機

構造

実装例

このことを踏まえたうえで、節や項を「@<hd>{}」で参照するとどう表示されるでしょうか。

- 節 (Section) には番号がついているので、たとえば「@<hd>{sec-visitor}」のように節を参照すると、「1.1 Visitor パターン」のように表示されます。これだと番号がついているので、読者は節を探しやすいです。
- しかし項 (Subsection) には番号がついていないので、たとえば「@<hd>{subsec-motivation}」や「@<hd>{subsec-structure}」のように項を参照すると、「動機」や「構造」とだけ表示されてしまいます。これだと番号がついていないので、読者は項を探せないでしょう。

▼サンプル（最初の 1 つは節を参照、残り 3 つは項を参照）

```
 * @<hd>{sec-visitor}
 * @<hd>{subsec-motivation}
 * @<hd>{subsec-structure}
 * @<hd>{subsec-impl}
```

表示結果：

- 「1.1 Visitor パターン」
- 「動機」
- 「構造」
- 「実装例」

問題点をもう一度整理しましょう。

- Re:VIEW のデフォルト設定では、項 (Subsection) に番号がつかない。
- そのため、「@<hd>{}」で項を参照するとタイトルだけになってしまい、番号がつかないので読者が項を探せない。

この問題に対し、Starter では「@<secref>{}」という新しい命令を用意しました。この新命令には次のような利点があります。

- 番号のついていない項でも、親となる節を使うことで探しやすい表示をしてくれる。
- その項のページ番号がつくので、該当ページに直接アクセスできる。

次のサンプルを見れば、「@<hd>{}」との違いがすぐに分かるでしょう。

▼サンプル（最初の 1 つは節を参照、残り 3 つは項を参照）

```
* @<secref>{sec-visitor}
* @<secref>{subsec-motivation}
* @<secref>{subsec-structure}
* @<secref>{subsec-impl}
```

表示結果：

- 「1.1 Visitor パターン」(p.1)
- 「1.1 Visitor パターン」内の「動機」(p.1)
- 「1.1 Visitor パターン」内の「構造」(p.1)
- 「1.1 Visitor パターン」内の「実装例」(p.1)

これを見ると、番号がついていない項の前に番号がついている節が置かれていること、またページ番号がついていることが分かります。どちらも@<hd>{}にはない特徴であり、@<hd>{}で参照するより節や項が探しやすくなります。

その他の注意事項です。

- 「@<secref>{}」は、節でも項でも、あるいは目 (Subsubsection) でも参照できます。今まで「@<hd>{}」を使っていた箇所はすべて「@<secref>{}」で置き換えられます。ただし、章 (Chapter) は参照できないので、その場合は「@<chapref>{}」を使ってください。
- 項にも番号をつけるよう設定している場合は、「@<secref>{}」の表示結果は「@<hd>{}」にページ番号をつけたものと同じです。
- 他の章 (Chapter) の節や項を参照する場合は、たとえば
  「@<secref>{chap-pattern|sec-visitor}」や
  「@<secref>{chap-pattern|subsec-impl}」のように書いてください。

わざわざ
「@<secref>{chap-pattern|sec-visitor|subsec-impl}」のように書く必要はありません。

「@<chapref>{}」や「@<hd>{}」をリンクに

Starter では、「@<chapref>{}」や「@<hd>{}」がリンクになるように設定しています。そのために次のような設定をしています。

- config.yml に「chapterlink: true」という設定を追加（最終行）。
- sty/starter.sty で「\reviewsecref」を再定義し、「\hyperref」を使うように変更。

▼リスト：\reviewsecref を再定義

```
\renewcommand{\reviewsecref}[2]{%
  \hyperref[#2]{#1}(p.\pageref{#2})%      % 節や項のタイトルがリンク
  %{#1}(\hyperref[#2]{p.\pageref{#2}})%   % ページ番号がリンク
}
```

これらは Re:VIEW に実装済みの機能であり、Starter はそれらを有効化しただけです。しかしこれらの機能は Re:VIEW のドキュメントには書かれていないので、もしかしたら将来的に変更されるかもしれません。

また Starter の追加機能である「@<secref>{}」でも、リンクが作成されます。

ターミナルでのカーソル

ターミナルでのカーソルを表す機能を用意しました。たとえば次の例では、2 行目の真ん中の「f」にカーソルがあることを表しています。

▼カーソル

```
function fib(n) {
  return n <= 1 ? n : fib(n-1) : fib(n-2);
}
~
~
"fib.js" 3L, 74C written
```

その他

- ブロック命令「//clearpage」で改ページします。また過去との互換性のために、インライン命令「@<clearpage>{}」も使えます。
- 「@<hearts>{}」とすると、「♡」と表示されます。
- 「@<TeX>{}」とすると、「TEX」と表示されます。

- 「@<LaTeX>{}」とすると、「LaTeX」と表示されます。

1.2　レイアウトやデザインに関する変更や拡張

Starter の設定ファイル「config-starter.yml」

Starter では、「config-starter.yml」という設定ファイルを新たに用意しました。この設定ファイルを編集することで、プロジェクトをダウンロードしたあとでもレイアウトやデザインを簡単に変更できます。

たとえば以下のことが変更できます。

- PDF のターゲット（印刷用か、ダウンロード用か）
- 章や節や項のデザイン
- プログラムやターミナルの表示で使う等幅フォント
- ドラフトモード（画像を枠線だけで表示する）

残念ながら、プロジェクト作成時に GUI で設定できたことがすべて「config-starter.yml」でできるわけではありません。しかしなるべく多くのことがこの設定ファイルで変更できるようにするつもりです。

フォントサイズの変更に対応

Re:VIEW 2.5 は、標準では本文のフォントサイズを 9pt や 8pt に指定しても、効いてくれません（まじかー！）。ウソだと思うかも知れませんが、実際に苦しんだ人の証言[*14]があるのでご覧ください。先人の苦労が偲ばれます。

[*14] https://www.slideshare.net/KazutoshiKashimoto/nagoya0927-release の p.21 と p.22。

▲図 1.5: フォントやページサイズを変更できなかった人の証言

この問題は、「geometry.sty」というスタイルファイルをオプションなしで読み込んでいることが原因です[*15]。Starter ではこれを読み込まないように修正している[*16]ため、フォントサイズを 9pt や 8pt に指定すればそのサイズになります。

A5 サイズの指定に対応

Re:VIEW 2.5 は、標準では A5 サイズの指定が効いてくれません（まじかー！）。ウソだと思うかも知れませんが、実際にトラブルに陥った人の証言があります[*17]。

> **トラブル発生！！**
>
> 原稿データチェック、表紙チェック、ともに問題なく終わったかにみえた午後 1 時。
> 何かに気づいたお姉さんの声音が変わりました。
> 「すみません、PDF サイズ……B5 になってます」
> 「えっ……」
> めのまえがまっくらになった。
> セイチョウ・ジャーニーは A5 で制作しているはずなのに、B5 サイズに？？
> わからない！ どうして！ だって何度も確認したはずだ！！

[*15] 簡単に書いてますけど、原因が geometry.sty であることを突き止めるのには大変な時間がかかり、正月休みを潰してしまいました。コノウラミハラサデオクベキカ！
[*16] 修正箇所は、layouts/layout.tex.erb の 50 行目あたりです。
[*17] https://blog.vtryo.me/entry/submit-of-journey

> と度重なる徹夜で脳死寸前の僕はパニック状態になりました。

入稿で明らかになるトラブル！ 怖いですねー。こういう予期せぬトラブルがあるので、締切りギリギリまで作業するのは止めて、余裕をもって入稿しましょう。

さて、A5 にならない問題は 2 種類あります。

- 本文の大きさが A5 サイズにならない。
- 本文の大きさは A5 なのに PDF が A5 サイズにならない。

前者は、「geometry.sty」が原因です。すでに説明したように、Starter では「geometry.sty」を読み込まないようにしているため、この問題は起こりません。

後者は、上で紹介したトラブルですね。これは jsbook.sty のオプションに「papersize」が指定されてないせいです。Starter ではこのオプションを指定しているので、A5 や B5 の指定どおりの PDF が生成されます。

詳しくは、config.yml の「`texdocumentclass:`」を参照してください。

▼config.yml

```
texdocumentclass: ["jsbook",
    #"uplatex,papersize,twoside,b5j,10pt,openright" # B5 10pt 右起こし
    #"uplatex,papersize,twoside,b5j,10pt,openany"   # B5 10pt 両起こし
    #"uplatex,papersize,twoside,a5j,9pt,openright"  # A5  9pt 右起こし
    #"uplatex,papersize,twoside,a5j,9pt,openany"    # A5  9pt 両起こし
    #"uplatex,papersize,oneside,a5j,10pt,openany"   # A5 10pt 両起こし
    "uplatex,papersize,twoside,a5j,9pt,openright"
]
```

本文の幅を全角 40 文字より長くできる

LaTeX の jsbook.cls ファイルを使うと、デフォルトでは本文の幅の最大値が 40 文字までに制限されています。これは、1 行が全角 40 文字より長いと読みづらくなるからという理由だそうです[*18]。

そのため、B5 サイズだとページ左右の余白が広めになってしまいます。ページ数を抑えて印刷代を下げたい人にとって、この余白はコストを増加させる要因です。

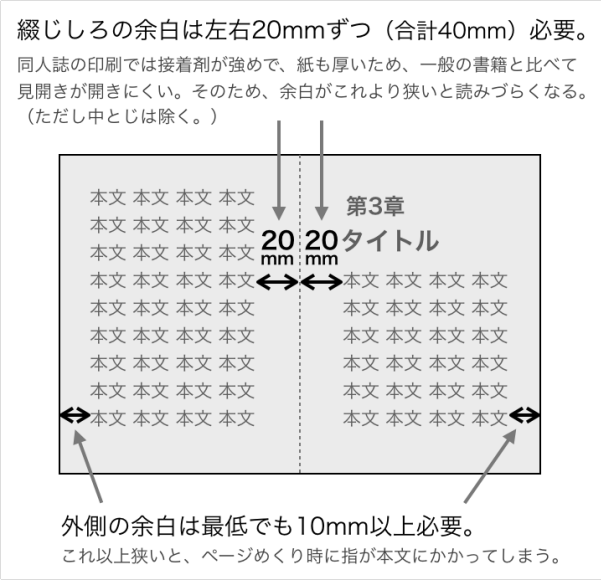
[*18] https://oku.edu.mie-u.ac.jp/~okumura/jsclasses/ に、jsbook.cls の作者である奥村先生が『書籍では 1 行の長さが全角 40 文字を超えないようにしています。』と解説しています。

Starter では sty/mytextsize.sty で本文幅を再設定することで、本文の幅を 40 文字より広くできます。B5 サイズでフォントが 10pt だと、1 行あたり全角 42〜45 文字がいいでしょう。

ただし A5 サイズ（フォント 9pt）では、1 行あたり 40 文字を超えるのはやめたほうがいいです。参考までに市販の技術書だと、A5 サイズで 1 行あたり全角 39 文字にすることが多いようです。

奇数ページと偶数ページで左右の余白を変える

本文の幅を広げる場合でも、左右の余白はちゃんと取りましょう（図 1.6）。



▲図 1.6: 奇数ページと偶数ページで左右の余白を変える

- 本を開いたときの中央（「ノド」という）の余白、つまり左ページの右余白と右ページの左余白は、最低でも 2cm は確保しましょう。そうしないと、ノド近くの文章がとても読みづらくなります。
- 本を開いたときの外側（「小口」という）の余白、つまり左ページの左余白と右ページの右余白は、最低でも 1cm は確保しましょう。そうしないと、ページをめくるときに指が本文にかかってしまい、読みにくいです。

Starter ではこういったことを考慮し、本文の幅を広げる（つまり 1 行あたりの

文字数を増やす）場合には小口側の余白だけを減らすようにしています。詳しくは sty/mytextsize.sty を見てください。

ページ上部の余白を減らし、その分を本文の高さに追加

LaTeX の標準のデザインでは、ページ上部の余白が大きめになっています。ページ数を少しでも減らして印刷代を抑えたい場合は、この余白がとても気になります。

Starter ではこの余白を約 1cm 減らし[*19]、その分を本文の高さに追加しています。詳しくは sty/mytextsize.sty を見てください。

ソースコード表示用のフォントを変更

LaTeX のデフォルトでは、装飾が多めのフォントがソースコードの表示に使われています（図 1.7 の上半分）。このフォントは「0」と「O」や「1」と「l」の区別がつきにくく、また太字にしてもあまり目立たないという欠点があります。

デフォルト（太字にしても目立ちにくい）

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789 Bold String
"'‘(){}[]<>;:,.+-*/%!&|^$?#@~_\
```

beramono フォント（太字が目立つ）

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789 Bold String
"'‘(){}[]<>;:,.+-*/%!&|^$?#@~_\
```

▲図 1.7: ソースコードの表示に使われるフォント

Starter では、ソースコードの表示に使うフォントを、装飾が少ないもの（Bera Mono[*20]）に変更しています（図 1.7 の下半分）。このフォントは「0」と「O」や「1」

[*19] 実は jsbook.cls では「1cm」は 1cm より少し大きく扱われ、厳密に 1cm を指定したい場合は「1truecm」とする必要があります。しかしここではそこまで厳密な 1cm を必要とはしていないので、sty/mytextsize.sty では「1cm」と指定しています。

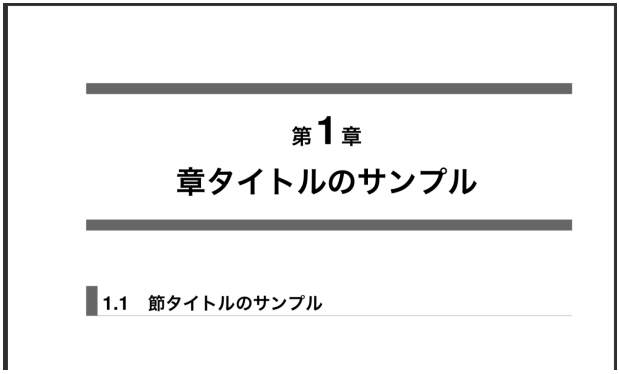
[*20] http://www.tug.dk/FontCatalogue/beramono/ でサンプルが見れます。

と「l」の区別がつきやすく、また太字にしたときも目立つようになっています。ただし「′」（シングルクォート）と「‵」（バッククォート）の区別がつきにくくなっているので注意してください。

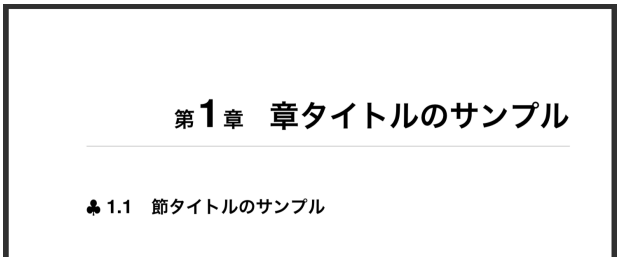
ソースコードの表示に向くフォントとしては、他にも「Inconsolata」[*21]や「Nimbus Mono Narrow」[*22]があります。興味がある人は調べてみてください。

章や節のデザインを変更可能

Starter では、章 (Chapter) や節 (Section) のデザインを変更できます。例を 2 つ挙げておきます（図 1.8、図 1.9）。



▲図 1.8: 章タイトルをセンタリング、上下に太線、節タイトルに下線



▲図 1.9: 章タイトルを右寄せ、下に細線、節タイトルの行頭にクローバー

[*21] http://www.tug.dk/FontCatalogue/inconsolata/ でサンプルが見れます。
[*22] http://www.tug.dk/FontCatalogue/nimbus15mononarrow/ でサンプルが見れます。

これらのデザインを調整するときは、config-starter.yml で設定を変更してください。この設定で飽き足らない場合は sty/starter.sty を編集してください。

なお Starter では、図 1.8 のように章タイトルの上下に太い線を入れた場合でも、まえがきや目次やあとがきのタイトルには太い線を入れないようにしています。これは意図的な仕様です。

章のタイトルページを作成可能

Starter では、章 (Chapter) のタイトルと概要を独立したページにできます（図 1.10）。これは商用の書籍ではよく見かける方法です。

▲図 1.10: 章のタイトルと概要を独立したページにした例（章ごとの目次つき）

やり方は簡単で、章タイトルと概要を書いたあとに「`//makechaptitlepage[toc=section]`」と書くだけです。これで章タイトルページが作られ、背景色がつき、その章の目次もつきます[*23]。

[*23] 実装の詳細は sty/starter.sty の makechaptitlepage コマンドを参照してください。

▼サンプル

```
= Re:VIEW Starter FAQ

//abstract{
残念ながら、Re:VIEWでできないことは、Starterでもたいていできません。

このFAQでは、「何ができないか？ 」を中心に解説します。
//}

//makechaptitlepage[toc=section]
```

ただし、「//makechaptitlepage[toc=section]」はすべての章に書く必要があります。これを書き忘れた章があると、そこだけ章タイトルページが作られません。注意してください。

目次の文字を小さく、行間を狭く

Starter では、目次のデザインを少し変更しています。

- 章 (Section) の文字をゴシック体にしました。項 (Subsection) の文字は明朝体のままなので、これで目次での章と項が見分けやすくなります。
- 項 (Subsection) の文字を少し小さくし、行間を狭くしました。これにより、目次にとられるページ数を少しだけ減らせます。

目次のデザインを修正する場合は、sty/starter.sty の中で「\l@section」や「\l@subsection」を探して、適宜修正してください。特に目次のページ数が多い場合は、行間を狭めて（「\baselineskip」を小さくして）みてください。

キャプションのデザインを変更

Starter では、ソースコードや表や図のキャプション（説明）を次のように変更しています。

- フォントをゴシック体にする
- 先頭に「▲」や「▼」をつける

これは TechBooster 製テンプレートのデザインを参考にしました。ただし LaTeX マクロの定義はまったく別です[*24]。

[*24] なおこれに関して、「\reviewimagecaption」というコマンドを新たに定義し、「\reviewimage」環境が「\caption」のかわりにそれを使うよう、LATEXBuilder#image_image() にモンキーパッチを適用しています。モンキーパッチは lib/hooks/monkeypatch.rb にあり、review-ext.rb が読み込んでいます。

引用のデザインを変更

引用「//quote{ ... //}」のデザインを変更し、左側に縦棒がつくようにしました。

Re:VIEW では LaTeX のデフォルトデザインのまま（全体が少しインデントされるだけ）なので、引用であることが分かりにくいです。これに対し、Starter では左側に縦棒がつくので、引用であることがより分かりやすくなっています。

また引用中に複数の段落を入れた場合、段落の先頭が 1 文字分インデントされます（Re:VIEW 標準ではインデントされません）。

▼サンプル

```
//quote{
その者蒼き衣を纏いて金色の野に降りたつべし。
失われし大地との絆を結び、ついに人々を清浄の地に導かん。
//}
```

表示結果：

> その者蒼き衣を纏いて金色の野に降りたつべし。失われし大地との絆を結び、ついに人々を清浄の地に導かん。

ページヘッダーを変更

一般の書籍では、ページヘッダーは次のような形式になっています。

- 見開きで左のページのヘッダーには、章タイトルを表示
- 見開きで右のページのヘッダーには、節タイトルを表示

しかし Re:VIEW では、両方のページのヘッダーに章タイトルと節タイトルが表示されています。これはおそらく、タブレットのような見開きがない閲覧環境を想定しているのだと思います。

Starter ではこれを変更し、一般の書籍と同じようなヘッダーにしています。ただしタブレット向けの場合は、Re:VIEW と同じようにしています。

ページ番号のデザインを変更

Re:VIEW のデフォルトでは、ページ番号はたとえば「10」のように表示されるだけです。

Starter では、ページ番号を「– 10 –」のような表示に変更しています。これは、ページ番号であることをより分かりやすくするためです。詳しくは sty/starter.sty を参照し

てください。

箇条書きの行頭記号を変更

LaTeX では、箇条書きの行頭に使われる記号が、第 1 レベルでは小さい黒丸「•」、第 2 レベルではハイフン「–」でした。

▼サンプル

```
* 第1レベル
** 第2レベル
```

表示結果（変更前）：

- 第 1 レベル
  - 第 2 レベル

しかしこれだと、箇条書きの記号ではなくマイナス記号に見えてしまいます。

Starter ではこの第 2 レベルの記号を、小さい白丸「○」に変更しました。これで、より自然な箇条書きになりました。

表示結果（変更後）：

- 第 1 レベル
  - 第 2 レベル

タイトルページと奥付を独立したファイルに

Starter では、タイトルページ（「大扉」といいます）と、本の最終ページにある「奥付」を、それぞれ別ファイルに分離しました。

- sty/mytitlepage.sty … タイトルページを表します。
- sty/mycolophon.sty … 奥付を表します。

タイトルページや奥付のデザインが気に入らない場合は、これらを編集してください。

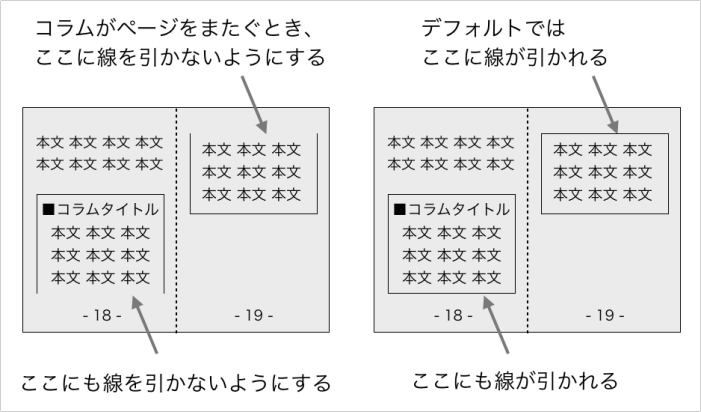
奥付が必ず最終ページになるよう修正

Re:VIEW では、奥付のページは単に改ページされてから作成されます。そのため、場合によっては奥付がいちばん最後のページではなく、最後から 2 番目のページになることがあります（この場合、最後のページは空白ページになります）。

Starter ではこれを改善し、奥付が必ず最後のページになるようにしています。詳しくは sty/starter.sty の「\reviewcolophon」コマンドを参照してください。この LaTeX コマンドは sty/mycolophon.sty から呼び出されています。

コラムがページまたぎする場合は横線を入れない

Starter では、コラムが長くてページをまたいでしまう場合に、横線をいれないようにしています（図 1.11）。こうすると、コラムが続いていることが分かりやすいです。



▲図 1.11: コラムがページをまたぐときに横線を入れない

1.3　LaTeX のコマンドやスタイルファイルに関する機能

スタイルシートを追加

Starter では、次のような独自のスタイルファイルを追加しています。

sty/starter.sty

Starter のサイトで選択したオプションに従って生成されたスタイルファイルです（Starter のバージョンが上がるたび、このファイルもよく変更されます）。デザインを調整したい場合などはこのファイルを編集するか、後述の sty/mystyle.sty で上書きしてください。

sty/mytextsize.sty

本文の幅やページ左右の余白を設定するためのスタイルファイルです。PDF のサイズ（B5 や A5）を変更する場合は、config.yml の「texdocumentclass:」を変更してください。

sty/mystyle.sty
: ユーザ独自の LaTeX マクロ（コマンドや環境）を追加したり、既存のマクロを上書きするためのファイルです。中身は空なので、自由に追加や上書きしてください。

sty/mytitlepage.sty
: タイトルページ（大扉）の内容を表すスタイルファイルです。デザインが気に入らない場合は編集してください。

sty/mycolophon.sty
: 最終ページの「奥付」を表すスタイルファイルです。デザインが気に入らない場合は編集してください。

sty/mytextsize.sty と sty/starter.sty は、どちらも自動生成されます。なので同じファイルにできそうですが、読み込むタイミングが異なるため、別ファイルにしています。

- sty/mytextsize.sty は本文の幅や高さを指定するので、他のスタイルファイルより先に読み込まれます。
- sty/starter.sty は既存の LaTeX マクロ（コマンドや環境）を上書きするので、他のスタイルファイルより後に読み込まれます。

印刷用 PDF と電子用 PDF を切り替える

Starter には、印刷用 PDF と電子用 PDF を切り替えて出力する機能があります[*25]。

印刷用 PDF
: 紙に印刷するための PDF です。色はモノクロで、また A5 の場合はページ左右の余白幅を変更します。

電子用 PDF
: ダウンロードで配布するための PDF です。色はカラーで、ページ左右の余白は同じ幅です。

[*25] ただしタブレット用にプロジェクトを作成した場合は、切り替えは無意味です。

ページ左右の余白幅を変える理由

印刷用 PDF においてページ左右の余白幅を変更するのは、本の読みやすさを保ったまま 1 行あたりの文字数を増やすためです。B5 の場合はたいてい十分な紙幅があるので、ページ左右の余白幅は同じままで大丈夫です。A5 の場合は見開きで内側の余白幅を確保したまま、外側の余白幅を狭めることで、1 行あたりの文字数を増やします。詳しくは「1.2 レイアウトやデザインに関する変更や拡張」内の「奇数ページと偶数ページで左右の余白を変える」(p.36) を参照してください。

当然ですが、このような変更は電子用 PDF では必要ありません。

設定ファイル config-starter.yml の中にある「starter:　target:　*xxx*」の値が「pbook[*26]」だと印刷用 PDF が、「ebook[*27]」だと電子用 PDF が生成されます。初期設定では「pbook」になっているので、デフォルトでは印刷用 PDF が生成されます。

またこの値は環境変数[*28] $STARTER_TARGET で上書きできます。具体的には次のようにすると印刷用と電子用の PDF を切り替えられます。

▼印刷用 PDF と電子用 PDF を切り替える

```
### 印刷用PDFを生成（デフォルト）
$ rake pdf    # または STARTER_TARGET=pbook rake pdf

### 電子用PDFを生成（環境変数を使って設定を上書き）
$ STARTER_TARGET=ebook rake pdf
```

ただしこの機能では、LaTeX のスタイルファイル（sty/starter.sty や sty/mytextsize.sty）の中で行える範囲でしか変更はできません。それ以上のことがしたい場合は、「2.7 その他」内の「印刷用と電子用で設定を少し変えるにはどうするの？」(p.68) を参照してください。

ドラフトモードにして画像読み込みを省略する

Starter では画像の読み込みを省略する「ドラフトモード」を用意しました。ドラフトモードにすると、画像のかわりに枠線が表示されます。こうすると、（LaTeX のコンパイル時間は変わりませんが）DVI ファイルから PDF を生成する時間が短縮されます。

[*26] 「pbook」は printing book の略です。
[*27] 「ebook」は electric book の略です。
[*28] 環境変数とは、コマンドプロセスが参照する外部変数です。環境変数を設定することでコマンドの挙動を一部変更できます。詳しくは「環境変数」でぐぐってください。

この機能は、図やスクリーンショットが多い場合や、印刷用に高解像度の画像を使っている場合は、特に効果が高いです。

ドラフトモードにするには、config-starter.yml で「draft: true」を設定するか、または環境変数「$STARTER_DRAFT」に何らかの値を入れてください。

▼ドラフトモードにして PDF を生成する

```
$ export STARTER_DRAFT=1   # ドラフトモードをonにする
$ rake pdf

$ unset STARTER_DRAFT      # ドラフトモードをoffにする
```

また「ドラフトモードにして PDF 生成時間を短縮したい、でもこの画像は表示して確認したい」という場合は、「//image[][][draft=off]」のように第 3 引数に draft=off を指定すると、その画像はドラフトモードが解除されて PDF に表示されます。

コンパイル時の出力を抑制

LaTeX でコンパイルすると（つまり uplatex コマンドを実行すると）、通常ではたくさんのメッセージが出力されます。これはとても煩わしいので、Starter では出力を抑制するために uplatex コマンドに「-interaction=batchmode」オプションをつけています。

しかしこのオプションをつけると、今度はエラーメッセージが表示されないという問題があります。つまり、こういうことです：

- 出力を抑制したいなら、LaTeX コマンドに「-interaction=batchmode」オプションをつける。
- しかし「-interaction=batchmode」オプションをつけると、エラーメッセージが表示されない。

なんというクソ仕様でしょう！ このクソ仕様を、Starter では次のように回避しています。

1. 「-interaction=batchmode」オプションをつけてコンパイルする。
2. エラーになったら（つまりコマンドの終了ステータスが 0 でなければ）、「-interaction=batchmode」オプションを**つけずに**コンパイルし直すことで、エラーメッセージを表示する。

今のところ、この方法がいちばん妥当でしょう。

なおこの変更は「rake pdf」コマンドでのみ行われます[*29]。「review-pdfmaker con-

[*29] 実装の詳細は lib/tasks/review.rake を参照してください。

fig.yml」を実行した場合はもとの挙動（つまりコンパイルメッセージがたくさん出る）になるので注意してください。

ちなみに、LaTeX のコマンドはエラーメッセージを標準エラー出力 (stderr) に出してくれません。クソかよ。

LaTeX コマンドにオプションを追加

Starter では、LaTeX コマンド（uplatex）に以下のオプションをつけています。

-halt-on-error
: LaTeX のコンパイルエラー時に、インタラクティブモードにせず、そのままコマンドを終了させるオプションです。

-file-line-error
: LaTeX のコンパイルエラー時に、エラー発生箇所の行番号に加えて、ファイル名も出力するようにするオプションです。

指定箇所は config.yml の「`texoptions:`」です。

実行する LaTeX コマンドをオプションつきで出力

Starter では、実行する LaTeX コマンドをオプションつきで出力するように変更しています[*30]。こうすることで、特にエラーが発生した場合にどんなコマンドを実行したかを調べるのに役立ちます。

ただしこれは「`rake pdf`」を実行したときだけであり、コマンドラインから直接「`review-pdfmaker config.yml`」を実行したときは出力されません[*31]ので注意してください。

次が実行例です。uplatex コマンドや dvipdfmx コマンドが、オプションつきで出力されていることが分かります。

▼実行例

```
$ rake pdf
compiling chap00-preface.tex
compiling chap01-starter.tex
compiling chap02-review.tex
compiling chap99-postscript.tex
```

[*30] この変更は、lib/tasks/review.rake で定義されている「`pdf`」タスクを書き換えることで実現しています。

[*31] なぜなら、この変更は「`pdf`」タスクを書き換えることで実現しているので、review-pdfmaker コマンドには影響しないからです。

```
[review-pdfmaker]$ /usr/bin/ruby /tmp/xxx-book/lib/hooks/beforete›
›xcompile.rb /tmp/xxx-book/xxx-book-pdf /tmp/xxx-book

[review-pdfmaker]$ uplatex -halt-on-error -file-line-error -inter›
›action=batchmode samplebook.tex
This is e-upTeX, Version 3.14159265-p3.8.1-u1.23-180226-2.6 (utf8›
›.uptex) (TeX Live 2018) (preloaded format=uplatex)
 restricted \write18 enabled.
entering extended mode

[review-pdfmaker]$ uplatex -halt-on-error -file-line-error -inter›
›action=batchmode samplebook.tex
This is e-upTeX, Version 3.14159265-p3.8.1-u1.23-180226-2.6 (utf8›
›.uptex) (TeX Live 2018) (preloaded format=uplatex)
 restricted \write18 enabled.
entering extended mode

[review-pdfmaker]$ uplatex -halt-on-error -file-line-error -inter›
›action=batchmode samplebook.tex
This is e-upTeX, Version 3.14159265-p3.8.1-u1.23-180226-2.6 (utf8›
›.uptex) (TeX Live 2018) (preloaded format=uplatex)
 restricted \write18 enabled.
entering extended mode

[review-pdfmaker]$ dvipdfmx -d 5 -z 3 book.dvi
book.dvi -> book.pdf
[1][2][3][4][5][6][7][8][9][10][11][12]
386603 bytes written
```

なお実行結果を見ると、LaTeX のコンパイル（つまり uplatex コマンドの実行）が 3 回行われていることが分かります。これはバグではなく、Re:VIEW の仕様です。理由は、ページ数に変更があっても対応できるようにするためと思われます。

PDF 生成を高速化する

DVI ファイルを PDF ファイルに変換する「dvipdfmx」コマンドのオプションを、圧縮率を少し下げるかわりに短時間で終わるようにするよう、設定しました。

具体的には、config.yml の「`dvioptions:`」という項目を、Re:VIEW のデフォルトの「`"-d 5 -z 9"`」から「`"-d 5 -z 3"`」に変更しています。「`-z 9`」は圧縮率を最大にするので時間がかかり、「`-z 3`」は圧縮率を下げるかわりに短時間で済みます。

PDF ファイルのサイズを少しでも減らしたい場合は、「`-z 9`」にしてみてください。

PDF にノンブルをつける

印刷所によっては、PDF にノンブルをつけるのが必須です。たとえば日光企画[*32]さんは、ノンブルをつけないと入稿ができません[*33]。

■ノンブルとは

ノンブルとは、すべてのページにつけられた通し番号です。ページ番号と似ていますが、ページ番号が読者のための番号なのに対し、ノンブルは印刷所の人が間違えずに作業するための番号です。具体的には次のような違いがあります。

- ページ番号は読者のためにつけるので、読者から見えやすい場所につける。ノンブルは印刷所の人が見えればいいので、読者には見えにくい場所につける。
- ページ番号は、まえがきや目次では「i, ii, iii, ...」、本文では「1, 2, 3, ...」と増える。ノンブルは最初から「1, 2, 3, ...」と増える。
- ページ番号は、タイトルページや空白ページではつかないことがある。ノンブルは、すべてのページに必ずつける必要がある。

詳しくは「ノンブル」で Google 検索してください。

Starter では、PDF にノンブルをつけるための rake タスク「pdf:nombre」[*34]を用意しています。

```
$ gem install combine_pdf     # 事前作業（最初の1回だけ）
$ rake pdf:nombre
```

これで、PDF ファイルにノンブルがつきます。

もし pdf:nombre タスクがうまく動作しない場合は、かわりに `https://kauplan.org/pdfoperation/` を使ってください。

rake コマンドのデフォルトタスクを指定する

Re:VIEW では、rake のデフォルトタスクが「epub」になっています。つまり引数なしで rake コマンドを実行すると、epub を生成するタスクが実行されます。

[*32] 技術書典でいちばん多くのサークルがお世話になっている印刷所。電話対応のお姉さんがとても親切。入稿ページの使い方が分かりにくいので、ほとんどの初心者はお姉さんのお世話になるはず。

[*33] `http://www.nikko-pc.com/q&a/yokuaru-shitsumon.html#3-1` より引用：『ノンブルは全ページに必要です。ノンブルが無いものは製本時にページ順に並び替えることが非常に困難な為、落丁・乱丁の原因となります。』

[*34] lib/tasks/starter.rake で定義されています。

これはあまり便利とはいえないし、なにより Ruby と rake をよく知らない人にとっては優しくない仕様です。

そこで Starter では、rake の使い方を表示する「help」タスクを用意し、これをデフォルトタスクにしています。このおかげで、引数なしで rake コマンドを実行すると rake の使い方が表示されます。このほうが、Ruby と rake をよく知らない人にとって優しいでしょう。

▼引数なしで rake コマンドを実行すると、rake の使い方が表示される

```
$ rake
rake -T
rake all             # generate PDF and EPUB file
rake clean           # Remove any temporary products
rake clobber         # Remove any generated files
rake epub            # generate EPUB file
rake help            # + list tasks
rake html            # build html (Usage: rake build re=target.re)
rake html_all        # build all html
rake images          # + convert images (high resolution -> low res
rake images:toggle   # + toggle image directories ('images_{lowres,
rake pdf             # generate PDF file
rake pdf:nombre      # + add nombre (rake pdf:nombre [file=*.pdf] [
rake preproc         # preproc all
rake web             # generate stagic HTML file for web
```

上の表示結果のうち、コマンドの説明文の先頭に「+」がついているのが、Starter で独自に用意したタスクです。

また環境変数「$RAKE_DEFAULT」を設定すると、それがデフォルトタスクになります。たとえば「pdf」タスクをデフォルトにしたい場合は、次のようにします。

▼pdf タスクをデフォルトタスクにする

```
$ export RAKE_DEFAULT=pdf    # デフォルトタスクを変更する。
$ rake                       # 引数がないのにpdfタスクが実行される。
compiling chap00-preface.tex
compiling chap01-starter.tex
compiling chap02-review.tex
compiling chap99-postscript.tex

[review-pdfmaker]$ uplatex -halt-on-error -file-line-error -inter›
›action=batchmode samplebook.tex
....(以下省略)....
```

第2章
Re:VIEW Starter FAQ

残念ながら、Re:VIEW でできないことは、Starter でもたいていできません。
この FAQ では、「何ができないか？」を中心に解説します。

2.1 コメント

範囲コメントはないの？

範囲コメントは、Re:VIEW にはありませんが Starter にはあります。

▼サンプル

```
aaa

#@+++
bbb

ccc
#@---

ddd
```

表示結果：

aaa
ddd

- 「#@+++」から「#@---」までが範囲コメントです。
- 「+」や「-」の数は 3 つです。それ以上でも以下でも範囲コメントとは認識されません。
- 範囲コメントは入れ子にできません。
- 「//embed」の中では使わないでください。
- これは実験的な機能なので、将来は仕様が変更したり機能が削除される可能性が

あります。この機能にあまり依存しないようにし、できれば行コメントを使ってください。一時的なコメントアウトに限定して使うのがいいでしょう。

行コメントを使ったら勝手に段落が分かれたんだけど、なんで？

Re:VIEW の仕様です。

たとえば次のような 5 行は、1 つの段落になります。

▼サンプル

```
これから王国の復活を祝って、諸君にラピュタの力を見せてやろうと思ってね。
見せてあげよう、ラピュタの雷を！
旧約聖書にある、ソドムとゴモラを滅ぼした天の火だよ。
ラーマーヤナではインドラの矢とも伝えているがね。
全世界は再びラピュタのもとにひれ伏すことになるだろう。
```

表示結果：

これから王国の復活を祝って、諸君にラピュタの力を見せてやろうと思ってね。見せてあげよう、ラピュタの雷を！ 旧約聖書にある、ソドムとゴモラを滅ぼした天の火だよ。ラーマーヤナではインドラの矢とも伝えているがね。全世界は再びラピュタのもとにひれ伏すことになるだろう。

ここで途中の行（3 行目）をコメントアウトすると、段落が 2 つに分かれてしまいます。

▼サンプル

```
これから王国の復活を祝って、諸君にラピュタの力を見せてやろうと思ってね。
見せてあげよう、ラピュタの雷を！
#@#旧約聖書にある、ソドムとゴモラを滅ぼした天の火だよ。
ラーマーヤナではインドラの矢とも伝えているがね。
全世界は再びラピュタのもとにひれ伏すことになるだろう。
```

表示結果：

これから王国の復活を祝って、諸君にラピュタの力を見せてやろうと思ってね。見せてあげよう、ラピュタの雷を！

ラーマーヤナではインドラの矢とも伝えているがね。全世界は再びラピュタのもとにひれ伏すことになるだろう。

なぜかというと、コメントアウトされた箇所が空行として扱われるからです、まるでこのように：

▼サンプル

```
これから王国の復活を祝って、諸君にラピュタの力を見せてやろうと思ってね。
見せてあげよう、ラピュタの雷を！

ラーマーヤナではインドラの矢とも伝えているがね。
全世界は再びラピュタのもとにひれ伏すことになるだろう。
```

段落が分かれてしまうのはこのような理由です。

Re:VIEW 開発チームに問い合わせたところ、これが Re:VIEW の仕様であるという回答が返ってきました。しかしこの仕様だと、段落を分けずに途中の行をコメントアウトする方法がありません。この仕様は、仕様バグというべきものでしょう。

そこで Starter では、段落の途中の行をコメントアウトしても段落が分かれないように変更しました。

表示結果：

これから王国の復活を祝って、諸君にラピュタの力を見せてやろうと思ってね。見せてあげよう、ラピュタの雷を！ ラーマーヤナではインドラの矢とも伝えているがね。全世界は再びラピュタのもとにひれ伏すことになるだろう。

こちらのほうが明らかに便利だし、困ることはないと思います。

また「`//list`」や「`//terminal`」でも行コメントが有効（つまり読み飛ばされる）ことに注意してください。

2.2　箇条書き

箇条書きで英単語が勝手に結合するんだけど？

Re:VIEW のバグです[*1]。次のように箇条書きの要素を改行すると、行がすべて連結されてしまいます。

▼サンプル

```
* aa bb
  cc dd
  ee ff
```

表示結果：

[*1] 少なくとも Re:VIEW 3.1 まではこのバグが存在します。

- aa bbcc ddee ff

これは日本語だと特に問題とはなりませんが、英語だと非常に困ります。

そこで Starter では、行を連結しないように修正しています。Starter だと上の例はこのように表示されます。

▼サンプル

```
* aa bb
  cc dd
  ee ff
```

表示結果：

- aa bb cc dd ee ff

順序つき箇条書きに「A.」や「a.」や「(1)」を使いたい

Re:VIEW ではできません。

Re:VIEW では、順序つき箇条書きとしては「1. 」や「2. 」という書き方しかサポートしていません。数字ではなくアルファベットを使おうと「A. 」や「a. 」のようにしても、できません。Re:VIEW の文法を拡張するしかないです。

なので Starter では文法を拡張し、これらの順序つき箇条書きが使えるようにしました。

▼サンプル

```
- 1. 項目1
- 2. 項目2

- A. 項目1
- B. 項目2

- a. 項目1
- b. 項目2
```

表示結果：

1. 項目 1
2. 項目 2

A. 項目 1
B. 項目 2

a. 項目 1
b. 項目 2

「-」の前と後、そして「1.」や「A.」や「a.」のあとにも半角空白が必要です。また半角空白の前の文字列がそのまま出力されるので、「(1)」や「A-1:」などを使えます。

▼サンプル

```
- (1) 項目1
- (2) 項目2

- (A-1) 項目1
- (A-2) 項目2
```

表示結果：

---

(1) 項目 1
(2) 項目 2

(A-1) 項目 1
(A-2) 項目 2

順序つき箇条書きを入れ子にできない？

Re:VIEW ではできません。

Re:VIEW では、順序なし箇条書きは入れ子にできますが、順序つき箇条書きは入れ子にできません。箇条書きの入れ子をインデントで表現するような文法だとよかったのですが、残念ながら Re:VIEW はそのような仕様になっていません。

そこで Starter では、順序つき箇条書きを入れ子にできる文法を用意しました。行の先頭に半角空白が必要なことに注意。

▼サンプル

```
- (A) 大項目
-- (1) 中項目
--- (1-a) 小項目
--- (1-b) 小項目
-- (2) 中項目
```

表示結果：

(A) 大項目
　(1) 中項目
　　(1-a) 小項目
　　(1-b) 小項目
　(2) 中項目

また順序なし箇条書きと順序つき箇条書きを混在できます。繰り返しますが、行の先頭に半角空白が必要なことに注意。

▼サンプル

```
* 大項目
-- a. 中項目
-- b. 中項目
*** 小項目
*** 小項目
```

表示結果：

- 大項目
  a. 中項目
  b. 中項目
     - 小項目
     - 小項目

2.3　ブロック命令

ブロックの中に別のブロックを入れるとエラーになるよ？

Re:VIEW の仕様です。

Re:VIEW では、たとえば「`//note{ ... //}`」の中に「`//list{ ... //}`」を入れると、エラーになります。これはかなり困った仕様です。

そこで Starter ではこれを改良し、ブロック命令の入れ子ができるようになりました。

▼サンプル

```
//note[■ノートの中にソースコード]{

ノートの中にソースコードを入れるサンプル。

//list[][サンプルコード]{
print("Hello, World!")
//}

//}
```

表示結果：

■ノートの中にソースコード

ノートの中にソースコードを入れるサンプル。

▼サンプルコード

```
print("Hello, World!")
```

ただし他のブロック命令を含められる（つまり入れ子の外側になれる）のは、今のところ次のブロック命令だけです。

- //note
- //quote
- //memo

これ以外の命令を入れ子対応にしたい場合は、ハッシュタグ「#reviewstarter」をつけてツイートしてください。

また以下のブロック命令は、その性質上他のブロック命令を含めることはできません。

- //list, //emlist, //listnum, //emlist
- //cmd, //terminal
- //program
- //source

なお Starter では、以前は「====[note] ... ====[/note]」といった記法を使っていました。この記法は今でも使えますが、ブロック命令の入れ子がサポートされた現在では使う必要もないでしょう。

ブロックの中に箇条書きを入れても反映されないよ？

Re:VIEW の仕様です。

Re:VIEW では、たとえば「//note{ ... //}」の中に「 * 項目 1」のような箇条書きを入れても、箇条書きとして解釈されません。これはかなり困った仕様です。

そこで Starter ではこれを改良し、ブロック命令の中に箇条書きが入れられるようになりました。

▼サンプル

```
//note[■ノートの中に箇条書きやソースコードを入れる例]{

 * 項目1
 * 項目2

//}
```

表示結果：

■ノートの中に箇条書きやソースコードを入れる例

- 項目 1
- 項目 2

現在のところ、以下のブロック命令で箇条書きをサポートしています。

- //note
- //quote
- //memo

これ以外の命令を入れ子対応にしたい場合は、ハッシュタグ「#reviewstarter」をつけてツイートしてください。

「//info{ ... //}」のキャプションに「■メモ：」がつくんだけど？

Re:VIEW の仕様です。「//info」だけでなく、他の「//tip」や「//info」や「//warning」や「//important」や「//caution」や「//notice」も、すべて「■メモ：」になります！

▼サンプル

```
//memo[memoサンプル]{
//}
```

表示結果：

■メモ: memo サンプル

▼サンプル

```
//tip[tipサンプル]{
//}
```

表示結果：

■メモ: tip サンプル

▼サンプル

```
//info[infoサンプル]{
//}
```

表示結果：

■メモ: info サンプル

▼サンプル

```
//warning[warningサンプル]{
//}
```

表示結果：

■メモ: warning サンプル

▼サンプル

```
//important[importantサンプル]{
//}
```

表示結果：

■メモ: important サンプル

▼サンプル

```
//caution[cautionサンプル]{
//}
```

表示結果：

■メモ: caution サンプル

▼サンプル

```
//notice[noticeサンプル]{
//}
```

表示結果：

■メモ: notice サンプル

わけがわからないよ。

これらのかわりに、Starter では「//note{ ... //}」を使ってください。詳しくは「1.1 原稿本文を書くための機能」内の「ノート」(p.6) を参照のこと。

2.4　ソースコード

ソースコードの見た目が崩れるんだけど？

恐らく、ソースコードの中にタブ文字があることが原因でしょう。

Re:VIEW では、「//list」などに含まれるタブ文字を半角空白に展開してくれます。しかしこの展開方法に根本的なバグがあるため、正しく展開してくれません。

たとえば次の例では、1 つ目のコメントの前には半角空白を使い、2 つ目のコメントの前にはタブ文字を使っています。

▼サンプル

```
//terminal{
$ printf "Hi\n"          # コメントの前に半角空白
$ printf "Hi\n"          # コメントの前にタブ文字
//}
```

これを Re:VIEW でコンパイルすると、次のようにタブ文字のある行は表示が崩れてしまいます。しかもエラーメッセージが出るわけではないので、なかなか気づきません。

表示結果 (Re:VIEW)

```
$ printf "Hi\n"          # コメントの前に半角空白
$ printf "Hi\n"             # コメントの前にタブ文字
```

Starter ではこの不具合を修正し、タブ文字がある行でも表示が崩れないようにしました。

表示結果 (Starter)

```
$ printf "Hi\n"          # コメントの前に半角空白
$ printf "Hi\n"          # コメントの前に半角空白
```

ただし、タブ文字のある行に「@<b>{}」や「@<del>{}」があると、タブ文字を半角空白に正しく展開できません。これは技術的に修正しようがないので、ソースコードではタブ文字より半角空白を使うようにしてください。

コラム中のソースコードがページまたぎしてくれないよ？

仕様です。ブロックと違い、コラム（「==[column] ... ==[/column]」）の中にはブロックを入れられますが、たとえばソースコードを入れた場合、ページをまたぐことができません。これは LaTeX の framed 環境による制限です。

ソースコードを別ファイルから読み込む方法はないの？

https://github.com/kmuto/review/issues/887 によると、このような方法でできるようです。

▼別ファイルのソースコード (source/fib1.rb) を読み込む方法

```
//list[fib1][フィボナッチ数列]{
@<include>{source/fib1.rb}
//}
```

ただし先のリンクにあるように、この方法は undocumented であり、将来も機能が提供されるかは不明です。「直近の締切りに間に合えばよい」「バージョンアップはしない」という場合のみ、割り切って使いましょう。もし使えなくなったとしても、開発チームに苦情を申し立てないようお願いします。

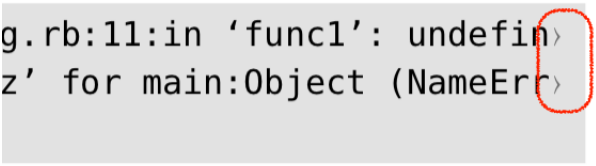
日本語だと長い行での折り返しが効かないの？

Starter では、プログラムやターミナルでの長い行を自動的に折り返してくれます。これは英語でも日本語でも動作します。

しかし折り返し箇所が日本語だと、折り返し記号がつきません。これは LaTeX での制限であり、解決策は調査中です。一時的な対策として、折り返す箇所に「@<foldhere>{}」を埋め込むと、折り返し箇所が日本語でも折り返し記号がつきます。

まだ文字が入りそうなのに折り返しされるのはなんで？

Starter で長い行が自動的に折り返されるとき、右端にはまだ文字が入るだけのスペースがありそうなのに折り返しされることがあります（図 2.1）。



▲図 2.1: 右端にはまだ文字が入るだけのスペースがありそうだが…

このような場合、プログラムやターミナルの表示幅をほんの少し広げるだけで、右端まで文字が埋まるようになります。

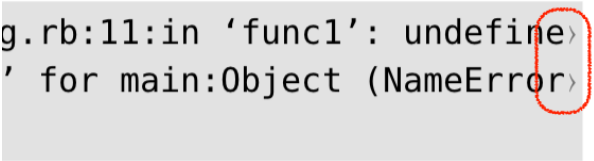
具体的には、ファイル「sty/starter.sty」の該当箇所を次のように変更してください。ここでは「0.3mm」ほど表示幅を広げてますが、この値は各自で調整してください。

▼ファイル「sty/starter.sty」

```
%%% コードブロック（プログラムリストやターミナル）
\newenvironment{starter@codeblock}{%
  ...(省略)...
  \begin{alltt}%
    \setlength{\baselineskip}{0.85\baselineskip}%
    \MakeFramed{%
      \advance\hsize-\width%
      \addtolength{\hsize}{0.3mm}%  ← この行を追加（数値は要調整）
      \advance\hsize-2\starter@codeblock@sidemargin%
      \FrameRestore%
    }%
}{%
```

```
...（省略）...
```

こうすると、プログラムやターミナルの表示幅が少しだけ広がり、文字が右端まで埋まるようになります（図 2.2）。



▲図 2.2: 表示幅をほんの少し広げると、右端まで埋まるようになった

2.5　コンパイル

なんで LaTeX のコンパイルがいつも 3 回実行されるの？

Re:VIEW の仕様です。LaTeX のコンパイル中にページ番号が変わってしまうと、古いページ番号のまま PDF が生成されてしまいます。このような事態を防ぐために、3 回コンパイルしているのだと思われます[*2]。

コンパイルに時間かかりすぎ。もっと速くできない？

たいていの場合、時間がかかるのは LaTeX のコンパイルではなく、そのあとの PDF 生成です。そして PDF 生成は、画像の数やサイズや解像度に比例して時間がかかります。

画像のファイル数は減らせないので、かわりに画像のサイズを減らしたり、執筆中だけダミー画像で置き換えるのがいいでしょう。詳しくは『ワンストップ！ 技術同人誌を書こう』という本の第 8 章を参照してください。

また Starter ではドラフトモードを用意しています。ドラフトモードでは画像が枠線で表示されるだけで読み込まれないため、PDF 生成がとても高速化します。詳しくは「1.3 LaTeX のコマンドやスタイルファイルに関する機能」内の「ドラフトモードにして画像読み込みを省略する」(p.45) を参照してください。

[*2] 本当ならコンパイルの前後で aux ファイルを比較して、ページ番号に差異があればもう一度コンパイルする、という方法が望ましいですが、そこまではしていないようです。

または、config.yml の「dvioptions:」の値を調整してください。「-z 1」だと圧縮率は低いけど速くなり、「-z 9」だと圧縮率は高いけど遅くなります。

2.6　タイトルページ（大扉）

タイトルが長いので、指定した箇所で改行したいんだけど？

長いタイトルをつけると、タイトルページ（「大扉」といいます）でタイトルが変な箇所で改行されてしまいます。

表示例：

週末なにしてますか? 忙しいですか? 金魚すくってもらっていいですか?

この問題に対処するために、Starter ではタイトル名に改行を含められるようになっています。config.yml の「booktitle:　|-」という箇所[*3]に、タイトル名を複数行で指定してください。

▼サンプル

```
booktitle: |-
  週末なにしてますか?
  忙しいですか?
  金魚すくってもらっていいですか?
```

こうすると、タイトルページでも複数行のまま表示されます。

表示例：

週末なにしてますか?
忙しいですか?

[*3] 「|-」は、YAML において複数行を記述する記法の 1 つ（最後の行の改行は捨てられる）。

金魚すくってもらっていいですか?

同様に、サブタイトルも複数行で指定できます。

ただし本の最後のページにある「奥付」では、タイトルもサブタイトルも改行されずに表示されます。

Starter ではなく、素の Re:VIEW や Techbooster のテンプレートを使っている場合は、layouts/layout.tex.erb を変更します。変更するまえに、layouts/layout.tex.erb のバックアップをとっておくといいでしょう。

▼layouts/layout.tex.erb

```
....(省略)....
\thispagestyle{empty}
\begin{center}%
  \mbox{} \vskip5zw
   \reviewtitlefont%
    {\HUGE\bfseries <%= escape_latex(@config.name_of("booktitle")) %> \par}%
    {\HUGE\bfseries 週末なにしてますか?\newline%
                    忙しいですか?\newline%
                    金魚すくってもらっていいですか?\par}%
....(省略)....
```

タイトルページがださい。もっとかっこよくならない？

LaTeX を使ってるかぎりは難しいでしょう。それよりも Photoshop や Keynote を使ってタイトルページを作るほうが簡単です（図 2.3）。

▲図 2.3: Keynote を使って作成したタイトルページの例

タイトルページを Photoshop や Keynote で作る場合は、config.yml で「`titlepage:  false`」を指定し、タイトルページを生成しないようにしましょう。そのあと、別途作成したタイトルページの PDF ファイルと「rake pdf」で生成された PDF ファイルとを結合してください。

なお奥付も Photoshop や Keynote で作る場合は、config.yml に「`colophon: false`」を指定し、奥付を生成しないようにしてください。また config.yml には「`colophon:`」の設定箇所が 2 箇所あるので、ファイルの下のほうにある該当箇所を変更してください。

2.7　その他

設定ファイルをいじったら、動かなくなった！

Re:VIEW の設定ファイルである config.yml や catalog.yml は、「YAML」というフォーマットで記述されています。このフォーマットに違反すると、設定ファイルが読み込めなくなるため、エラーになります。

「YAML」のフォーマットについての詳細は Google 検索してください。ありがちな

ミスを以下に挙げておきます。

- タブ文字を使うと、エラーになります。かわりに半角スペースを使ってください。
- 全角スペースを使うと、エラーになります。かわりに半角スペースを使ってください。
- 「:」のあとに半角スペースが必要です。たとえば
  「titlepage:false」はダメです。
  「titlepage: false」のように書いてください。
- 「,」のあとに半角スペースが必要です。たとえば
  「texstyle: [reviewmacro,starter,mystyle]」はダメです。
  「texstyle: [reviewmacro, starter, mystyle]」のように書いてください。
- インデントが揃ってないと、エラーになります。たとえば catalog.yml が次のようになっていると、インデントが揃ってないのでエラーになります。

▼「CHAPS:」のインデントが揃ってないのでエラー

```
PREDEF:
  - chap00-preface.re

CHAPS:
   - chap01-starter.re
  - chap02-faq.re

APPENDIX:

POSTDEF:
  - chap99-postscript.re
```

- 「-」のあとに半角スペースが必要です。たとえば上の例で
  「- chap01-starter.re」が
  「-chap01-starter.re」となっていると、エラーになります。

印刷用と電子用で設定を少し変えるにはどうするの？

印刷所に入稿するための PDF と、電子用（ダウンロード用）の PDF で、設定を変えたいことがあります。

- 印刷用の PDF は白黒だけど、電子用の PDF はカラーにしたい。
- 印刷用の PDF は外側の余白を詰めるけど、電子用ではしない。

このように印刷用の PDF と電子用の PDF で設定を変えたい場合、Re:VIEW では設定ファイルを継承して別の設定ファイルを作成します。しかしこの機能は設定ファイ

ルを切り替えることしかできないので、使いづらいです。

Starter では別の設定ファイルを用意しなくても、環境変数「$STARTER_TARGET」を切り替えるだけで印刷用と電子用の PDF を切り替えられます。詳しくは「1.3 LaTeX のコマンドやスタイルファイルに関する機能」内の「印刷用 PDF と電子用 PDF を切り替える」(p.44) を参照してください。

```
### 印刷用PDFを生成
$ rake pdf     # または STARTER_TARGET=pbook rake pdf

### 電子用PDFを生成
$ STARTER_TARGET=ebook rake pdf
```

ただしこの機能では、LaTeX のスタイルファイル（sty/starter.sty や sty/mytextsize.sty）の中で行える範囲でしか変更はできません。そのため、たとえば config.yml や catalog.yml や layouts/layout.tex.erb で行うような変更をしたい場合[*4]は、自力で頑張る必要があります。

方針としては、設定ファイルやスタイルファイルを用途に応じて都度生成するといいでしょう。具体的には次のようにします。

(1) まず少し変えたいファイルの名前を変更し、末尾に「.eruby」をつけます。

```
$ mv config.yml          config.yml.eruby
$ mv sty/mytextsize.sty sty/mytextsize.sty.eruby
$ mv sty/starter.sty     sty/starter.sty.eruby
## またはこうでもよい
$ mv config.yml{,.eruby}
$ mv sty/mytextsize.sty{,.eruby}
$ mv sty/starter.sty{,.eruby}
```

(2) 次に、それらのファイルに次のような条件分岐を埋め込みます。

▼config.yml.eruby

```
....(省略)....
<% if buildmode == 'printing'   # 印刷向け %>
texdocumentclass: ["jsbook", "uplatex,papersize,twoside,b5j,10pt,›
›openright"]
<% elsif buildmode == 'tablet'  # タブレット向け %>
texdocumentclass: ["jsbook", "uplatex,papersize,oneside,a5j,10pt,›
›openany"]
```

[*4] たとえば印刷用や電子用とは別にタブレット用を用意し、タブレット用では用紙サイズやフォントサイズを変えるような場合がこれに相当します。以降ではタブレット用を作成する例を紹介しています。

```
<% else abort "error: buildmode=#{buildmode.inspect}" %>
<% end %>
....(省略)....
<% if buildmode == 'printing'   # 印刷向け %>
dvioptions: "-d 5 -z 3"  # 速度優先
<% elsif buildmode == 'tablet'  # タブレット向け %>
dvioptions: "-d 5 -z 9"  # 圧縮率優先
<% else abort "error: buildmode=#{buildmode.inspect}" %>
<% end %>
....(省略)....
```

▼sty/mytextsize.sty.eruby

```
<%
if buildmode == 'printing'   # 印刷向け
  textwidth  = '42zw'
  sidemargin = '1zw'
elsif buildmode == 'tablet'  # タブレット向け
  textwidth  = '40zw'
  sidemargin = '0zw'
else abort "error: buildmode=#{buildmode.inspect}"
end
%>
....(省略)....
\setlength{\textwidth}{<%= textwidth %>}
....(省略)....
\addtolength{\oddsidemargin}{<%= sidemargin %>}
\addtolength{\evensidemargin}{-<%= sidemargin %>}
....(省略)....
```

▼sty/starter.sty.eruby

```
....(省略)....
<% if buildmode == 'printing'   # 印刷向け %>
\definecolor{starter@chaptercolor}{gray}{0.40}  % 0.0: black, 1.0
: white
\definecolor{starter@sectioncolor}{gray}{0.40}
\definecolor{starter@captioncolor}{gray}{0.40}
\definecolor{starter@quotecolor}{gray}{0.40}
<% elsif buildmode == 'tablet'  # タブレット向け %>
\definecolor{starter@chaptercolor}{HTML}{20B2AA} % lightseagreen
\definecolor{starter@sectioncolor}{HTML}{20B2AA} % lightseagreen
\definecolor{starter@captioncolor}{HTML}{FFA500} % orange
\definecolor{starter@quotecolor}{HTML}{E6E6FA}   % lavender
<% else abort "error: buildmode=#{buildmode.inspect}" %>
<% end %>
....(省略)....
```

```
<% if buildmode == 'printing'   # 印刷向け %>
\hypersetup{colorlinks=true,linkcolor=black} % 黒
<% elsif buildmode == 'tablet'  # タブレット向け %>
\hypersetup{colorlinks=true,linkcolor=blue}  % 青
<% else abort "error: buildmode=#{buildmode.inspect}" %>
<% end %>
```

(3) ファイルを生成する Rake タスクを定義します。ここまでが準備です。

▼lib/tasks/mytasks.rake

```
def render_eruby_files(param)   # 要 Ruby >= 2.2
  Dir.glob('**/*.eruby').each do |erubyfile|
    origfile = erubyfile.sub(/\.eruby$/, '')
    sh "erb -T 2 '#{param}' #{erubyfile} > #{origfile}"
  end
end


namespace :setup do

  desc "*印刷用に設定 (B5, 10pt, mono)"
  task :printing do
    render_eruby_files('buildmode=printing')
  end

  desc "*タブレット用に設定 (A5, 10pt, color)"
  task :tablet do
    render_eruby_files('buildmode=tablet')
  end

end
```

(4)「rake setup::printing」または「rake setup::tablet」を実行します。すると、config.yml と sty/mytextsize.sty と sty/starter.sty が生成されます。そのあとで「rake pdf」を実行すれば、用途に応じた PDF が生成されます。

```
$ rake setup::printing  # 印刷用
$ rake pdf
$ mv mybook.pdf mybook_printing.pdf

$ rake setup::tablet    # タブレット用
$ rake pdf
$ mv mybook.pdf mybook_tablet.pdf
```

LaTeX のスタイルファイルから環境変数を読める？

Starter では、名前が「STARTER_」で始まる環境変数を LaTeX のスタイルファイルから参照できます。

たとえば「STARTER_FOO_BAR」という環境変数を設定すると、sty/mystyle.sty や sty/starter.sty では「\STARTER@FOO@BAR」という名前で参照できます。想像がつくと思いますが、環境変数名の「_」は「@」に変換されます。

▼環境変数を設定する例 (macOS, UNIX)

```
$ export STARTER_FOO_BAR="foobar"
```

▼環境変数を参照する例

```
%% ファイル：sty/mystyle.sty
\newcommand\foobar[0]{%                 % 引数なしコマンドを定義
  \@ifundefined{STARTER@FOO@BAR}{%      % 未定義なら
    foobar%                             % デフォルト値を使う
  }{%                                   % 定義済みなら
    \STARTER@FOO@BAR%                   % その値を使う
  }%
}
```

この機能を使うと、出力や挙動を少し変更したい場合に環境変数でコントロールできます。また値の中に「$」や「\」が入っていてもエスケープはしないので注意してください。

あとがき

いかがだったでしょうか。感想や質問は随時受けつけています。

著者紹介

ジョン・スミス
某所で働くエンジニア。

はじめての XXX 入門

概念と基本操作を 2 時間でマスター！

2019 年 9 月 22 日　ver 1.0

著　者　カウプラン機関極東支部
印刷所　〇〇印刷所