

はじめての ReVIEW

書籍制作のノウハウがぎゅっとつまつたガイドブック。
あなたの思いを形にできます。



 **TechBooster**
How to Android by techbooster.org

はじめての ReVIEW

TechBooster 著

2013-12-31 版 mhidaka 発行

はじめに

はじめての ReVIEW を手に取っていただき、ありがとうございます。本書は ReVIEW を実際に使用している執筆経験者により制作されました。書籍制作ツールである ReVIEW を使えば、DTP とよばれる分野、原稿の作成や編集、レイアウトなど制作の多くを効率的に作業できます。同人誌や技術誌を作つてみたい、電子出版に興味がある方はもちろん、書籍制作のワークフローを改善したい方にとってもガイドブックとしてお勧めできる一冊です。

(発行代表 @mhidaka)

本書の内容について

- ・ 入門、開発環境、執筆をはじめるまでのステップを丁寧に解説
- ・ ReVIEW の使い方、スタイルファイル、コマンドなど用法を解説
- ・ 業務で役立つ仕様書の作成方法
- ・ 同人誌における書籍制作ワークフローなど実践の紹介
- ・ 頻出の専門用語やトラブルの Q&A 集

TechBooster とは

TechBooster は Android をはじめとしたモバイルのための技術サークル^{*1}です。オープンソースへの貢献や社会還元を目的にサイトでモバイル技術を解説しています。

お問い合わせ先

本書に関するお問い合わせ <https://plus.google.com/+TechboosterOrg/>

*1 TechBooster の Web サイト <http://techbooster.org/>

第1章

ReVIEW 入門

本章では ReVIEW とは何かについて説明します。ReVIEW 記法の特徴や、それを文書の作成に用いるまでの利点・注意点を紹介していきます。

ReVIEW を用いた原稿の例

ReVIEW（「れびゅー」と読みます）の定義を説明する前にまず雰囲気を感じてもらうため、ReVIEW を用いた実例を見てみましょう。リスト 1.1 は書籍『Effective Android』^{*1} の筆者担当分の冒頭です。紙面の都合で原文にはない改行などが含まれていますが、原稿ほぼそのままです。

リスト 1.1: 『Effective Android』 38 章の冒頭

```
= Google Drive API を使ってファイルをダウンロードする
```

本章では、Python スクリプトから Google Drive に保存されているファイルを取得する方法を紹介します。

Android とは直接関係ありませんが、
同様の API を Android 上で利用する際に参考になるかもしれません。

```
== 収録されている背景
```

『Effective Android』 同人誌版の執筆が行われていた頃、
原稿は Google Drive 上で管理されていました。
一部の技術者の希望により git も用いる運用に途中から切り替えたのですが、
全員が git を利用できるわけではないため、
原則 Google Drive にファイルをアップロードとし、
git は希望者が選択して用いる、という体制になりました^{*2}。

```
//footnote[not_used] [本稿執筆時点ではすでにこの体制は終了しています。]
```

^{*1} <http://tatsu-zine.com/books/effective-android>

このとき、管理者が Google Drive 上のファイルを手動でダウンロードして git プロジェクトに取り込んでいると聞き、私はその作業を自動化できるかも知れないと考えました。

Google Drive API とその SDK は Google から無料で提供されていました。少し調べた結果、今回の目標のためにはそれを使えばよいことが分かりました。

* What Can You Do with the Drive SDK?@
{}

@<href>{<https://developers.google.com/drive/about-sdk>}

これが ReVIEW のコンパイラによって図 1.1 のような PDF に変換されます。

1.1 ReVIEW とは何か

この例により「ReVIEW」には実は二つの異なる側面があることがわかります。「ReVIEW 記法」と「ReVIEW ツール」の側面です。

まず「ReVIEW 記法」について。例で紹介した文章では、いくつか日本語では一般的でない記号が含まれています。それらが、以下のように解釈されています。

- 行頭に「=」を並べると、PDF ではその行は章や節と解釈される。
- 空行で区切ると段落になる。
- //footnote という命令で脚注を作ることが出来る。

人が読む文章の中に、機械が解釈するための命令を含めることで、見出しや段落、フォントサイズといった情報を埋め込んでいます。このような言語は、一般に「軽量マークアップ言語」と呼ばれます。「ReVIEW 記法」はそういった軽量マークアップ言語の一つです^{*3}。

次に「ReVIEW ツール」について。「ReVIEW 記法」で書かれた上記の文章を PDF に変換したツールもまた ReVIEW と呼ばれます。今回は PDF へ変換しましたが、PDF の他にも様々な形式のファイルを出力できます。

- HTML

^{*3} 「軽量」でない「マークアップ言語」もあります。境目は曖昧ですが、例えば後述する HTML や TeX は「軽量」とはあまり呼ばれないようです。人にとって記述しやすく読みやすいと思えるかが一つのポイントのようです。

- EPUB^{*4}
- InDesign XML^{*5}
- (.. その他)

^{*4} EPUB（イーパブ）は電子書籍の標準規格の1つです。

^{*5} Adobe InDesign（あどび いんでざいん）はAdobe社のプロプライエタリなDTP（Desktop · Publishing）ソフトで、InDesign XMLはそれが理解するXMLのフォーマットです。XMLの方はReVIEWでは「IDGXML」という表記をされることもあります。

第38章

Google Drive API を使ってファイルをダウンロードする

本章では、Python スクリプトから Google Drive に保存されているファイルを取得する方法を紹介します。Android とは直接関係ありませんが、同様の API を Android 上で利用する際に参考になるかもしれません。

38.1 収録されている背景

『Effective Android』同人誌版の執筆が行われていた頃、原稿は Google Drive 上で管理されていました。一部の技術者の希望により git も用いる運用に途中から切り替えたのですが、全員が git を利用できるわけではないため、原則 Google Drive にファイルをアップロードとし、git は希望者が選択して用いる、という体制になりました¹。

このとき、管理者が Google Drive 上のファイルを手動でダウンロードして git プロジェクトに取り込んでいると聞き、私はその作業を自動化できるかもしれませんと考えました。

Google Drive API とその SDK は Google から無料で提供されていました。少し調べた結果、今回の目標のためにはそれを使えばよいことが分かりました。

- What Can You Do with the Drive SDK?
<https://developers.google.com/drive/about-sdk>

38.2 プロジェクトの準備

Google Drive API を利用するには、まず Google Cloud Console 上で自分のプロジェクトを登録します。次にそのプロジェクトにおいて Google Drive API を利用できるよう設定し、

¹ 本稿執筆時点ではすでにこの体制は終了しています。

542

第38章 Google Drive API を使ってファイルをダウンロードする

最後に Python スクリプトが用いる OAuth 2.0² の ID とシークレットを取得します。順番に説明していきます。

- Google Cloud Console

図 1.1 38 章の PDF 出力例。章や脚注の処理が自動的に行われている

「ReVIEW」という表現で、軽量マークアップ言語としての「ReVIEW 記法」を指す場合

と、ReVIEW 記法で記述した原稿を処理する「ReVIEW ツール」を指す場合がある点、注意してください。

1.2 ReVIEW の特色

文章を記述するための軽量マークアップ言語は ReVIEW の他にもいくつかあります。例えば、Wikipedia などで使われる Wiki 記法^{*6}も軽量マークアップ言語です。IT 技術者業界では、他にも Markdown^{*7}, reStructuredText^{*8}, Textile^{*9}といった例があります。

他にも選択肢がある中で、ReVIEW を選択するメリットとは何でしょうか。

技術書の執筆・編集を想定して作られている

ReVIEW は特に、技術「書籍」の執筆と編集を想定しており、その現場と相性が良い設計になっています。

ReVIEW はもともと、青木峰郎 (@mineroaoki) 氏^{*10}が自身の著書の執筆のために開発を始めました。

現在では、武藤健志 (@kmuto) 氏^{*11}、高橋征義 (@takahashim) 氏^{*12}、角征典 (@kdmsnr) 氏^{*13}らが開発・保守をしています。この 4 名全員、本業・副業を通じて技術書籍等の執筆・編集に関わっていたか、あるいは現在進行形で関わっており、その業務の中で ReVIEW を利用しています。

^{*6} Wiki (ウィキ) は Web ブラウザから Web ページを編集して皆で共有できるシステムのこと。語源はハワイ語の「速い」で、Web ページの作成や更新が速く出来るという Wiki の特徴を良く示している。(Wikipedia や e-Words より)

^{*7} Markdown の利用例として Stack Overflow が有名。日本では IT 技術ブログとして Qiita などが採用している。文法に「方言」が多く、Stack Overflow も Qiita も微妙に書き方が違う。

^{*8} reStructuredText (ReST、RST など色々書き方がある) は、プログラミング言語 Python の技術ドキュメント記述によく使われる。

^{*9} Textile はプログラミング言語 Ruby 界隈で人気がある。同言語で書かれたプロジェクト管理ソフト『Redmine』でも採用されている。

^{*10} 古くからの Ruby 愛好家。最近の著書に『ふつうのコンパイラを作ろう』『ふつうの Haskell』プログラミングなど。

^{*11} 編集・翻訳・デザイン等を行う株式会社トップスタジオの執行役員。紙の商業出版の編集に ReVIEW を利用してきた実績多数。

^{*12} 株式会社達人出版会代表取締役、日本 Ruby の会代表理事。

^{*13} アジャイル開発手法に早くから注目し、関連書籍の翻訳等を手がけている。訳書に『Team Geek』『Running Lean』『リーダブルコード』など。

開発開始時から現在に至るまで、書籍の編集・出版で実際に使われることを意識して作られてきたわけです。

商業出版の実績も多数あります。武藤氏の所属するトップスタジオでは、すでに50冊程度の書籍の編集でReVIEWを使用しているそうです。また、高橋氏の主宰する達人出版会では、自社で出版しているPDF・EPUBの制作用フォーマットとしてReVIEWを採用しています。

拡張性に優れている

商業出版の現場に対応するため、ReVIEWでは拡張性の高い柔軟な構文を採用しています。

出版の現場では、HTMLのみならず、EPUBやInDesign XMLといった複数種類のフォーマットに向けて出力を行う必要が多々あります。そういった状況に1種類の原稿データで対応したい、それも原稿の書きやすさをなるべく落とさないように……そういう要求にReVIEWは応えられます。

ReVIEWは、出力形式の特長を引き出すための独自構文を追加できる上、こうした追加を行なっても他のケースへの悪影響を抑えることができます。書いた後から出力先が増えた場合にも、元の原稿に新しい出力についての情報を書き足せば良いのです。

一方、他の軽量マークアップ言語では、しばしばアウトプットとしてHTMLフォーマット1種類に特化して作られているため、こういった形で拡張を行うことがしばしば難しくなります。軽量でないマークアップ言語は習熟に時間がかかるため、そもそも原稿を書くのが大変です。

既存のマークアップ言語で原稿を書くとこうなる

軽量マークアップ言語のMarkdownを用いて技術ブログを書いていたとします。この時点では、記事の執筆は快適なはずです。

記事を書きためて人気もそこそこ、というところで、ブログ上の複数の記事をまとめて、一冊の技術者向け同人誌にしようと思い立ったとします。

Markdownは紙の出力を得意としないので、紙媒体に強いTeX^{*14}などを介してPDFを生

*14 読み方は多様らしいですが、筆者は「テフ」と読んでいます。記述は比較的面倒なため、マークアップ言語ながら「軽量」と思う人は稀な気がします。

成することになるでしょう^{*15}。TeX は HTML 出力向けではないため、TeX と Markdown の両方でデータを管理するか、一方からもう一方へ変換することで原稿を管理することになります^{*16}。

この時、少なくとも二つ、回避できない問題が発生します。

- 両方の文法を用いることになる。特に TeX も併用することになるため、Markdown のみを用いていたときほど、快適に執筆出来なくなる。
- 相互の変換時に、レイアウトのニュアンスが壊れるケースが発生する。特に紙面上で TeX が持つ表現力が HTML で失われる。

2種類のマークアップ言語を行き来することは、「表現力」と「書きやすさ」の性質が異なる世界を行き来することを意味します。当然、その過程で片方にはない要素を執筆者が補う手作業が発生します。

さて、フォーマット問題を乗り越えて同人誌を発行した後「電子書籍版も発行して欲しい」という要望が寄せられました。出来れば叶えたいところです。

しかし Markdown も TeX も、電子書籍に適したフォーマットというわけではありません。つまりここでも、紙に印刷するときと同様の困難が発生してしまいます^{*17}。

一方、ReVIEW はもともと HTML、PDF、EPUB といった異なる要求を持つ出力に対応できるだけの柔軟性を持っていますから、このような困難はなくなるか、少なくとも大分軽減されます^{*18}。

現在では、技術者が色々な媒体で情報発信を容易に出来る時代ですから、原稿執筆もスマートにしたいものです。ReVIEW はそれをスマートにサポートしてくれるはずです。

実例としての『Effective Android』

上記の例は、机上の空論ではありません。

*15 某社のワープロソフトを用いてもここで説明する状況は変わらないです。もう少し面倒な事態になるかもしれません。

*16 例えば Markdown から TeX であれば、Pandoc がサポートしているようです。

<http://johnmacfarlane.net/pandoc/>

*17 人気が出て出版社から紙の商業出版のオファーがあったとしましょう。今度は InDesign 対応を協議することになるかもしれません。

*18 特に商業出版では、ReVIEW を用いていても後半の工程で編集者による手作業が必要となることが多いそうです。ReVIEW はそういう現状も見越していて、その工程向けの「指示書」的なフォーマットすらサポートしています。

TechBooster『Effective Android』は、コミケット84^{*19}での紙の同人誌版から始まり、達人出版会による電子書籍版を経て、インプレス社から紙の商業誌として出版されました^{*20}。十数人からスタートした著者数は、版を重ねる毎に増加し、中途でAndroid本体のアップデート(4.4)が挟まって原稿の大修正が必要になったりと、上記の例よりもさらに苛烈な執筆状況でした。

その詳細は他章に譲るとしても、ここではそういう実例においてReVIEWが全面的に採用されたという点を指摘しておきます^{*21}。

オープンソースである

ReVIEWの利点としてもう一点、オープンソースであるという点も指摘したいと思います。

RubyによるReVIEWコンパイラ実装は世の中に公開されており、自由に利用できます^{*22}。バグ報告や機能拡張のリクエストもGitHub上で受け付けており、技術者であればpull request^{*23}を直接行うことも出来ます。『Effective Android』執筆過程で、その執筆者もバグ報告やpull requestを通じてReVIEW開発に貢献しました。

この利点はこれまでと異なり、他のマークアップ言語に対するものではありません。本章で比較されている他の有名なマークアップ言語も、オープンソース実装とセットであることが一般的だからです。ここでこの論点を示すのは、出版業の他のプラットフォームとの違いを示すためです。

特定の企業がツールや仕様を全て握っていて外部のコントリビュータによる修正を受け付けるチャンスを与えない場合、バグが一向に修正されないまま利用者が悩まされる、ということが往々にしてあります。そして、筆者の印象では出版関連ツールはこの手のプロプライエタリのソフトが一般的です。

武藤氏の2012年の発表資料の一節を図1.2に示します。

*19 2013年の夏コミ。

*20 厳密には、本章執筆時点ではまだ編集段階です。

*21 同人誌版で約190ページほど、インプレス版については最終版ではありませんが、執筆時点で手元にあるPDFでは約500~600ページ程度です。どちらにしても「薄い本」でないことは分かるかと思います。

*22 <https://github.com/kmuto/review>

*23 メインの開発者に「こういう実装を作ったので本家に取り込んでください」と伝える方法の一つです。GitHubでよく使います。



図 1.2 オープンでないソフトのバグは直されないと悲惨

ReVIEW のコミュニティはオープンであるため、クローズドなソフトが抱えることのある「バグが放置されて自分で直すことも出来ない」という事態を防ぐことが出来ます。

1.3 ReVIEW の向いている分野、向いていない分野

ReVIEW は、ブログのように一貫したスタイルに基づいて内容が記述されるページ構成をサポートするようにできています。(図 1.3)

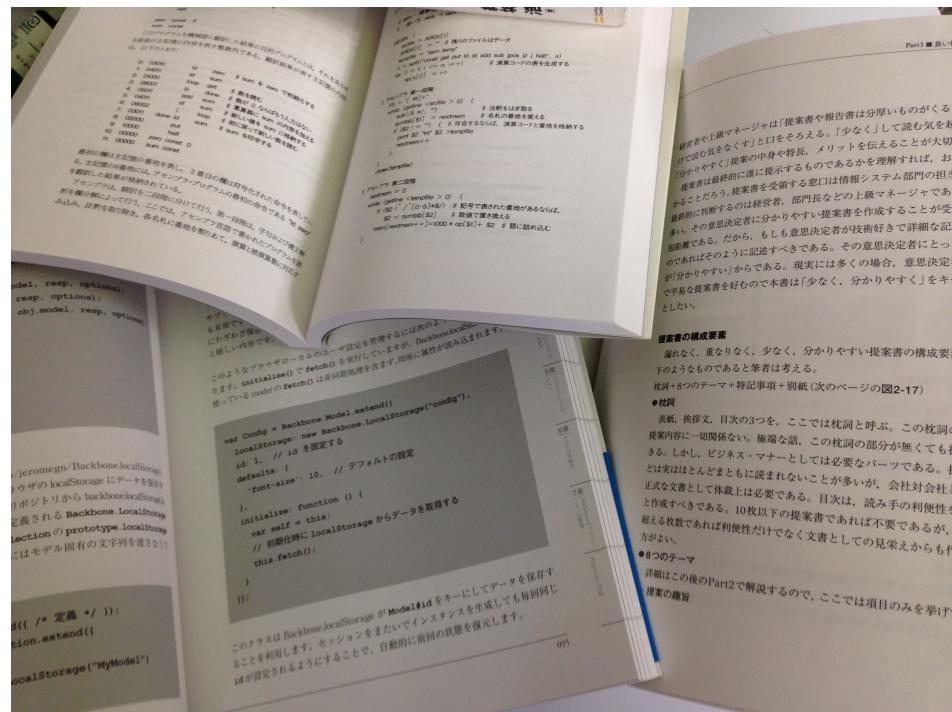


図 1.3 技術書や論文のような一本道の文章には大変強いです

そのため、細切れのストーリーを別のレイアウトで表現したり、見開きを跨いで図画を表示するといった用途にはあまり向いていません。(図 1.4)

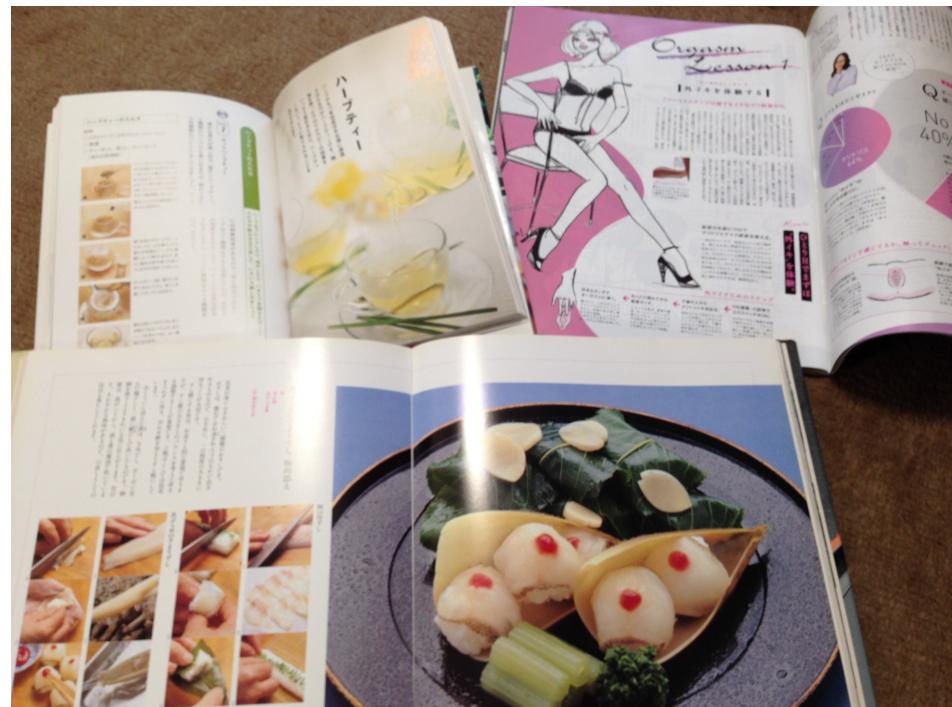


図 1.4 ページをまたぐレイアウトがあったり、コラムが別スタイルで乱舞するものは不得意です

他のツール同様、ReVIEW も全ての表現を実現するためのものではありませんので、利用する上でその点だけ押さえておく必要があります。

1.4 ReVIEW の課題

本書・本稿の目的は、ReVIEW の良さを読者に伝えることです。

しかし公平な視点に立つと、ReVIEW には「筋の良さ」がある一方で、課題もあると筆者は考えます。

読者が更に適切な評価を行えるよう、ReVIEW を利用する上で問題となりそうな点を挙げていきます。

他マークアップ言語と比べるとコミュニティが小さい

まず単純に、ユーザ数がまだまだ少ない点があげられます。

商業出版でも利用実績があることは既に紹介しましたが、現時点では「多数派」というわけではありません。ブログから原稿をアップデートする例を挙げましたが、こういった ReVIEW を採用する事例自体、他のマークアップ言語と比べるとまだまだ少ないので現状です。そもそも、ブログ等で初めから ReVIEW 文法をサポートしているものは現状ではほとんどありません^{*24}。

開発者の数でもまだまだこれからです。Ruby 以外でも ReVIEW 記法を解釈する実装が欲しいことはありますし、WordPress のような代表的なブログソフトにも、ReVIEW フォーマットをそのままブログ表示に適用できるプラグインが欲しい、と筆者は思います^{*25}。

日本人コミュニティ以外での利用例がほとんどないのもまた将来の課題です。ReVIEW の開発時のディスカッション、バグレポートの対応はほぼ日本語で、他言語の利用者・開発者は見当たりません。そもそも、日本の組版事情にあわせて作成されているため、ユーザ層の広がりが国内に留まりがちです。この問題の良い側面は、私達にとって問い合わせの敷居が低いということです。

^{*24} 筆者も当初はマイナーなマークアップ言語を新たに使うことにやや懐疑的だった記憶があります。

^{*25} 第9章「ReVIEW.js で学ぶ ReVIEW 記法のお約束」で紹介する ReVIEW.js は、この課題に対する前向きなアプローチの一例です。

Windows 環境では利用する敷居が特に高い

ReVIEW は、現状では Windows 環境で工夫なしに利用することができません^{*26}。Windows でも利用できる VirtualBox の仮想イメージが武藤氏によって公開されていましたり <http://d.kmuto.jp/20130811.html>、Web 上では Cygwin^{*27}を用いたビルド成功例が紹介されていましたりします^{*28}が、それでも UNIX 系 OS での利用実績のほうが豊富な印象です。

1.5 まとめ

ReVIEW は、書籍の出版に適した軽量マークアップ言語と、ReVIEW 記法で記述した原稿を処理するツールの総称です。

課題もあるものの、ReVIEW は今後非常に伸びしろの多い、期待感を持たせる原稿執筆プラットフォームです。個人での書籍執筆を想定する場合だけでも、その利点を即座に発揮出来るツールとなっています。

本著を通じて読者が使い始めることで、ReVIEW が今後より強力なプラットフォームになっていくことを願います。

最後に、読者のさらなる ReVIEW 理解のため、本章の執筆の上で参考にした URL、記事を紹介します。

- http://d.hatena.ne.jp/hdk_embedded/20130810/1376148749
 - TechBooster の代表日高氏による勉強会のまとめ。下記のリンクを全て含む。
- <https://docs.google.com/file/d/0BymMSOVfiOaWYXNTd0JGNlk3Yzg/edit>
 - 上記勉強会において利用された開発者武藤氏による資料
- <http://kmuto.jp/events/page2012/page2012.pdf>
 - 2012 年 2 月の page2012 にて武藤氏が利用したとされる資料

^{*26} そもそも <https://github.com/kmuto/review/issues/79>

^{*27} Cygwin（シグウィン）は、Windows オペレーティングシステム上で動作するオープンソースな UNIX ライクな環境の一つ（Wikipedia より）。

^{*28} <http://d.hatena.ne.jp/kaorun55/20120710/1341889820>

第2章

環境構築

本章では、ReVIEW で原稿を執筆する環境の構成と、その構築方法を紹介します。

2.1 ReVIEW 環境の構成

第1章でも述べたとおり、本書において「ReVIEW」とは、書籍の出版に適したテキストベースの軽量マークアップ言語「ReVIEW記法」と、ReVIEW記法でマークアップした原稿を処理するツール類「ReVIEWツール」の総称です。

ReVIEWツールは、ReVIEW構文で記述されたファイル（ReVIEWファイル）を、TEXT/HTML/Markdown/EPUB/IDGXML^{*1}やLaTeXなどの形式で出力します。また、組版ソフトウェア「pLaTeX」と組み合わせることで、PDF（Portable Document Format）への出力にも対応します。

本章では、ReVIEWによってPDFの出力ができる環境を「ReVIEW環境」と定義し、各プラットフォームにおけるReVIEWとLaTeXのインストールに焦点を当てて解説します。

ReVIEWツール

ReVIEWツールは、単体のソフトウェアではありません。特定の機能を持つ複数のプログラム（コマンド）セットで提供されています（表2.1）。

オリジナルのReVIEWツールは、プログラミング言語Rubyで実装されており、RubyのパッケージシステムRubyGemsで公開されています。

^{*1} Adobe InDesign CS2以降向けXML形式

表 2.1 主なコマンド一覧

コマンド	解説
review-epubmaker	ReVIEW ファイルから、EPUB を生成する
review-pdfmaker	ReVIEW ファイルから、PDF を生成する
review-compile	ReVIEW ファイルを、指定の形式に変換する
review-vol	ReVIEW ファイルごとのサイズ、文字数、行数、ページ数を一覧で表示する
review-index	ReVIEW ファイル全体の章、節、小節などの区切りごとのサイズ、文字数、行数を一覧で表示する
review-preproc	文字エンコードの変換やタブをスペースの置換処理などを行う

pLaTeX

LaTeX は、レスリー・ランポート (Leslie Lamport) 氏によって開発された組版処理システムです（図 2.1）。



図 2.1 LaTeX ロゴ

ドナルド・クヌース (Donald E. Knuth) 氏が開発した「TeX」に、マクロパッケージを追加する形で構築されています^{*2}。pLaTeX は、LaTeX をさらに日本語に対応するように拡張したもののです。

TeX(pLaTeX) は、TeX の構文でマークアップしたファイルをコンパイルして、DVI 形式^{*3}を出力します。DVI 形式は、PostScript や PDF などに変換できます。

ReVIEW ツールと pLaTeX を組み合わせることで、ReVIEW ファイルを PDF に出力する

^{*2} <http://ja.wikipedia.org/wiki/LaTeX>

^{*3} DeVice-Independent file format: 文書の見た目を画像形式・表示デバイス・プリンタに依存しない形で記録したデータ形式

ことができます。

2.2 Mac での環境構築

MacOS は、本稿執筆時点では最も ReVIEW の導入が容易なプラットフォームです。

なお、MacOS X 10.9(Mavericks) で検証しました。利用している MacOS のバージョンまたは環境によって、細部が異なる場合があるので注意してください。



図 2.2 MacOS X

ReVIEW ツール

MacOS に ReVIEW ツールをインストールするには、二つの方法があります。

一つは RubyGems を使ってインストールする方法。もう一つが、GitHub で公開されているリポジトリからファイルをチェックアウトする方法です。

RubyGems を使ってインストールすると、ReVIEW ツールは RubyGems の管理下にインストールされます。パッケージの更新や、複数バージョンの切り替えには、RubyGems を使いますが、MacOS に標準でインストールされている RubyGems は、gem のインストールに管理者権限が必要になります。

GitHub で公開されているリポジトリを clone すると、常に最新の ReVIEW ツールを使用することができます。インストール先はユーザーの管理下になり、RubyGems のように管理者権限は必要ありません。パッケージの更新や複数バージョンの切り替えには、git のコマン

ドを使います。

gem でインストール

MacOS には、RubyGems が標準で含まれています。

RubyGems を使って ReVIEW ツールのパッケージ (gem) をインストールするには、以下のコマンドを実行します。

```
$ sudo gem install review
```

sudo による管理者パスワードの確認後、管理者権限で ReVIEW ツールがインストールされます。

GitHub からインストール

GitHub の ReVIEW リポジトリ (<https://github.com/kmuto/review>) を clone します。なお、git がインストールされていない場合は、Xcode をインストールするなど準備をする必要があります。

```
$ git clone https://github.com/kmuto/review.git ~/review
Cloning into 'review'...
(省略)
Checking connectivity... done
```

clone が完了したら、review/bin を、環境変数 PATH に追加します。

```
$ echo 'export PATH=~/review/bin:$PATH' >> ~/.bash_profile
$ source ~/.bash_profile
```

また、clone した最新版の ReVIEW を gem としてインストールする場合は、rake install を実行します (gem "bundler"が必要です)。

```
$ cd review  
$ sudo rake install
```

以上で、ReVIEW ツールのインストールは完了です。

MacTeX

MacOS に TeX をインストールする最も容易な方法は、MacTeX のパッケージを使うことです。

MacTeX の公式サイト (<http://tug.org/mactex/>) から、MacTeX をダウンロードします。 MacTeX-2013 のパッケージは 2.3GB あります。標準でインストールすると 4GB を超えるので、ディスクの空き容量には注意してください。

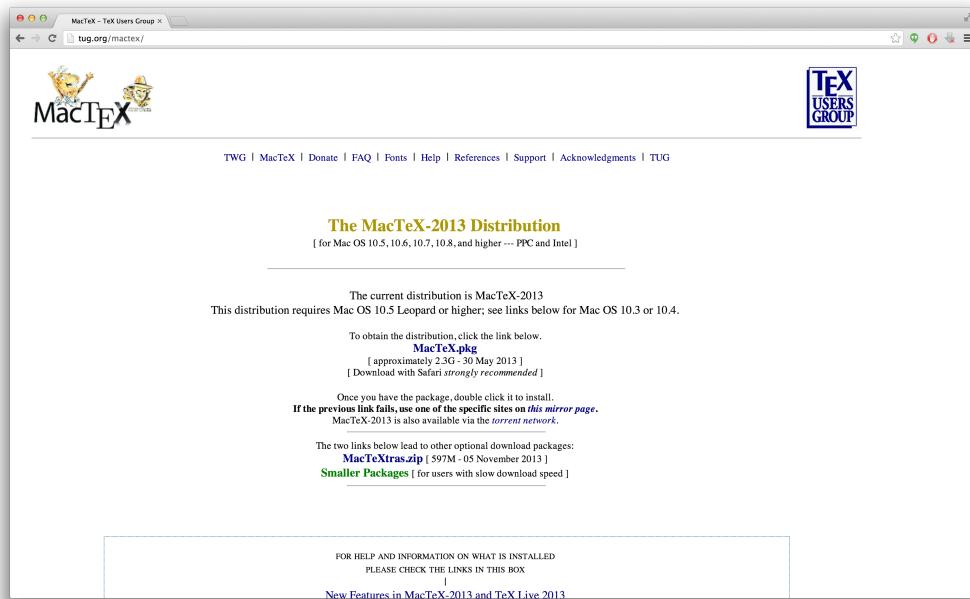


図: MacTeX のダウンロード

インストールを開始すると、利用許諾などの一般的な画面が表示されます。

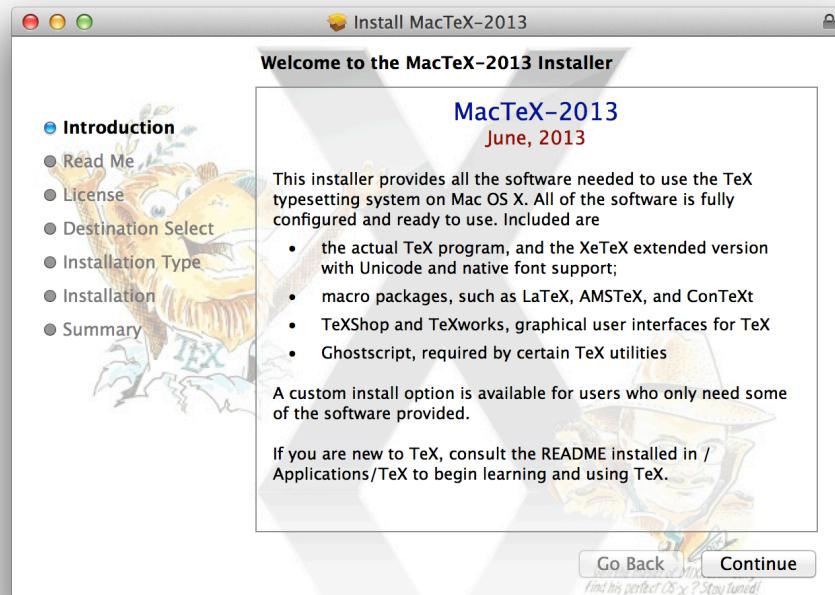


図: MacTeX のインストール

不要なパッケージをインストール installation Type の画面で、Customize を選択します。

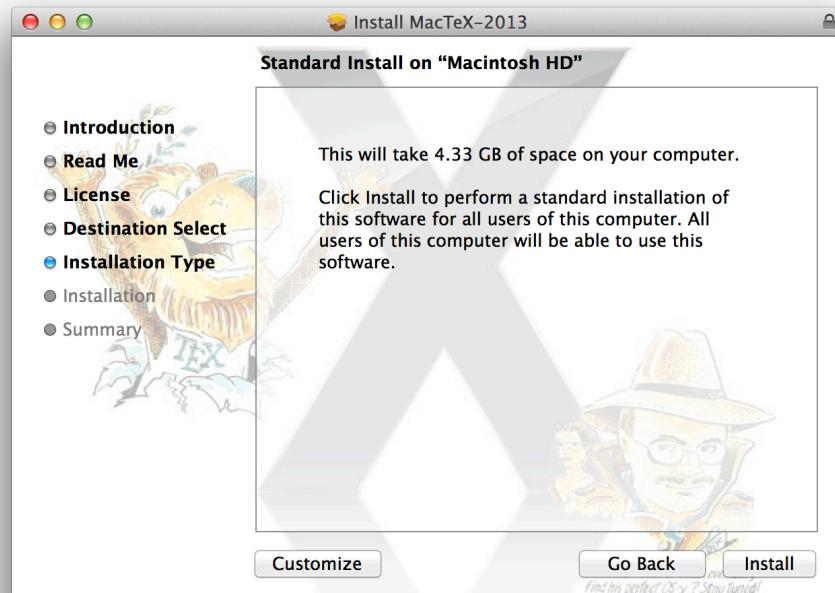
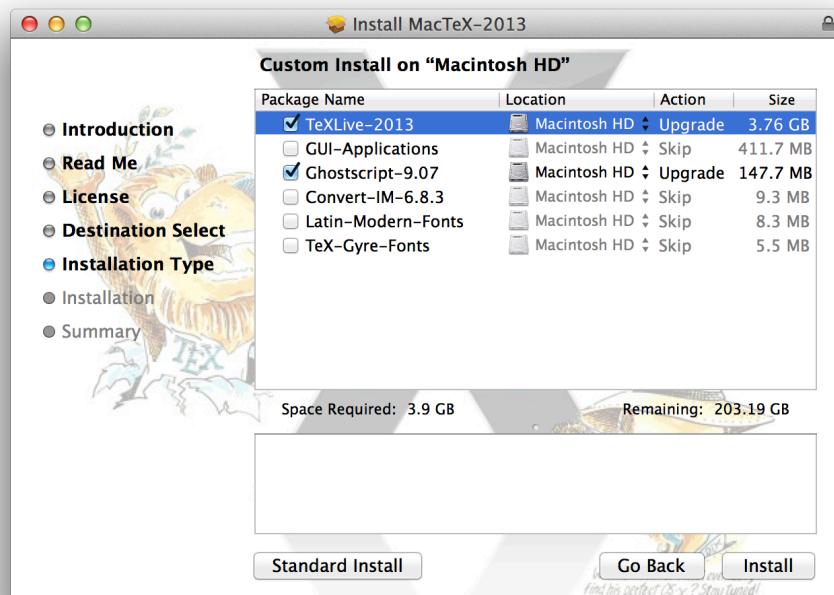


図: 「Customize」を選択する

ReVIEW に必要なパッケージは、TeXLive-2013 と GhostScript-9.07 です。そのほかのパッケージはチェックを外してください。



図：必要なパッケージのみチェックする

Install を押して、MacTeX のインストールを完了します。

PDF の出力

ReVIEW の環境が構築できたら、サンプルを PDF に変換してみましょう。

ReVIEW ツールを RubyGems を使ってインストールした場合は、gem の中にサンプルが含まれているのでコピーして、`review-pdfmaker` を実行します。

```
$ cp -rf /Library/Ruby/Gems/2.0.0/gems/review-1.1.0/test/sample-book ~/
$ cd sample-book/src
$ review-pdfmaker config.yml
```

上は、筆者の環境で ReVIEW をインストールした場合です。RubyGems や ReVIEW のバージョン、システムの設定によってはディレクトリが異なる場合があります。ReVIEW がインストールされているディレクトリを確認するには `gem which review` を実行します。

GitHub の場合は、リポジトリの中にサンプルが含まれているのでコピーして、`review-pdfmaker` を実行します。

```
$ cp -rf ~/review/test/sample-book ~/  
$ cd sample-book/src  
$ review-pdfmaker config.yml
```

2.3 Linux での環境構築

Linux での環境構築法を解説します。

なお、ここで紹介する環境の構築方法は、Ubuntu 12.04 LTS 64bit 版（以下「Ubuntu」といいます）で検証しています。利用しているディストリビューション、バージョンまたは環境によって、細部が異なる場合があるので注意してください。

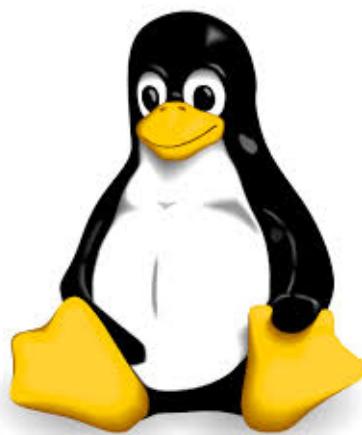


図 2.3 Linux

ReVIEW ツール

Ubuntu に ReVIEW ツールをインストールするには、二つの方法があります。

一つは RubyGems を使ってインストールする方法。もう一つが、GitHub で公開されているリポジトリからファイルをチェックアウトする方法です。

RubyGems を使ってインストールすると、ReVIEW ツールは RubyGems の管理下にインストールされます。パッケージの更新や、複数バージョンの切り替えには、RubyGems を使いますが、gem のインストールに管理者権限が必要になります。

GitHubで公開されているリポジトリをcloneすると、常に最新のReVIEWツールを使用することができます。インストール先はユーザーの管理下になり、RubyGemsのように管理者権限は必要ありません。パッケージの更新や複数バージョンの切り替えには、gitのコマンドを使います。

gemをインストール

まずははじめに、以下のコマンドを実行して、RubyGemsをインストールします。

```
$ sudo apt-get install rubygems

Reading package lists... Done
Building dependency tree
Reading state information... Done

(省略)

The following NEW packages will be installed:
binutils build-essential cpp cpp-4.6 dpkg-dev fakeroot g++ g++-4.6 gcc
gcc-4.6 libalgorithm-diff-perl libalgorithm-diff-xs-perl
libalgorithm-merge-perl libc-dev-bin libc6-dev libdpkg-perl libgomp1 libmpc2
libmpfr4 libquadmath0 libreadline5 libruby1.8 libstdc++6-4.6-dev
linux-libc-dev make manpages-dev ruby1.8 ruby1.8-dev rubygems
0 upgraded, 29 newly installed, 0 to remove and 0 not upgraded.
Need to get 33.6 MB of archives.
After this operation, 96.9 MB of additional disk space will be used.
Do you want to continue [Y/n]? y
```

sudoの管理者パスワード確認の後、RubyGemsをインストールするにあたって必要なパッケージと、インストールの確認が表示されるのでYを入力して実行します。

次に、以下のコマンドを実行して、ReVIEWツールのパッケージ(gem)をインストールします。

```
$ sudo gem install review
Fetching: review-1.1.0.gem (100%)
Successfully installed review-1.1.0
1 gem installed
Installing ri documentation for review-1.1.0...
```

```
Installing RDoc documentation for review-1.1.0...
```

GitHub からインストール

GitHub の ReVIEW リポジトリ (<https://github.com/kmuto/review>) を clone します。なお、git がインストールされていない場合は、apt-get などを使って準備をしておく必要があります。

```
$ git clone https://github.com/kmuto/review.git ~/review  
Cloning into 'review'...  
(省略)  
Checking connectivity... done
```

clone が完了したら、review/bin を、環境変数 PATH に追加します。

```
$ echo 'export PATH=~/review/bin:$PATH' >> ~/.bash_profile  
$ source ~/.bash_profile
```

また、clone した最新版の ReVIEW を gem としてインストールする場合は、rake install を実行します (gem "bundler"が必要です)。

```
$ cd review  
$ sudo rake install
```

以上で、ReVIEW ツールのインストールは完了です。

TeXLive

Ubuntu で pLaTeX は、TeXLive CJK パッケージとして提供されています。

しかし、TeXLive CJK が、Ubuntu のパッケージとしてインストール可能になったのは

Ubuntu のバージョン 12.10 からで、バージョン 12.04 にインストールする場合は、事前に 12.10 用のパッケージをバックポートしたリポジトリを登録する必要があります。

バックポートリポジトリを登録

以下のコマンドを入力して、Ubuntu のパッケージシステムにリポジトリを登録するユーティリティをインストールします。

```
$ sudo apt-get install python-software-properties

Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  python-pycurl
Suggested packages:
  libcurl4-gnutls-dev python-pycurl-dbg
The following NEW packages will be installed:
  python-pycurl python-software-properties
0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
Need to get 73.0 kB of archives.
After this operation, 428 kB of additional disk space will be used.
Do you want to continue [Y/n]? y
```

sudo の管理者パスワード確認の後、インストールの確認が表示されるので、Y を入力して実行します。

次に、以下のコマンドを入力して、TeXLive CJK のバックポートリポジトリを追加します。

```
$ sudo add-apt-repository ppa:texlive-backports

You are about to add the following PPA to your system:
 Backports of the latest TeX Live from Ubuntu 12.10 to Ubuntu 12.04 LTS
 More info: https://launchpad.net/~texlive-backports/+archive/ppa
Press [ENTER] to continue or ctrl-c to cancel adding it
```

ENTER キーを押すと、追加が完了します。

apt-get update で、先ほど追加したバックポートリポジトリを含めて、全てのパッケー

ジを再読み込みます。

```
$ sudo apt-get update
```

TeXLive CJK をインストール

以下のコマンドを実行して、TeXLive CJK をインストールします。

```
$ sudo apt-get install texlive-lang-cjk texlive-fonts-recommended
```

```
Reading package lists... Done  
Building dependency tree  
Reading state information... Done
```

(省略)

```
The following NEW packages will be installed:  
cmap-adobe-japan1 cpp cpp-4.6 doc-base fonts-ipaexfont-gothic  
fonts-ipaexfont-mincho fonts-ipafont-gothic fonts-ipafont-mincho ghostscript  
gs-cjk-resource gsfonds ko.tex-extra-hlfont latex-beamer latex-cjk-all  
latex-cjk-chinese latex-cjk-chinese-aphic-bkai00mp  
latex-cjk-chinese-aphic-bsmi00lp latex-cjk-chinese-aphic-gbsn00lp  
latex-cjk-chinese-aphic-gkai00mp latex-cjk-common latex-cjk-japanese  
latex-cjk-japanese-wadalab latex-cjk-korean latex-cjk-thai  
latex-fonts-thai-tlwg latex-xcolor libcupsimage2 libfile-basedir-perl  
libfile-desktopentry-perl libfile-mimeinfo-perl libfontenc1 libgl1-mesa-dri  
libgl1-mesa-glx libglapi-mesa libgraphite3 libgs9 libgs9-common libijs-0.35  
libjbig2dec0 libkpathsea6 liblcms2-2 libllvm3.0 libmpc2 libmpfr4  
libpaper-utils libpaper1 libpoppler19 libptexenc1 libreadline5 libruby1.8  
libutempter0 libuuid-perl libx11-xcb1 libxaw7 libxcb-glx0 libxcb-shape0  
libxfont1 libxmu6 libxtst6 libxv1 libxf86dga1 libxf86vm1 libyaml-tiny-perl  
lmodern luatex pgf preview-latex-style prosper ps2eps ruby ruby1.8 swath  
tcl8.4 tex-common texlive-base texlive-binaries texlive-common  
texlive-doc-base texlive-doc-zh texlive-extra-utils texlive-font-utils  
texlive-generic-recommended texlive-lang-cjk texlive-latex-base  
texlive-latex-base-doc texlive-latex-extra texlive-latex-extra-doc  
texlive-latex-recommended texlive-latex-recommended-doc texlive-luatex  
texlive-pictures texlive-pictures-doc texlive-pstricks texlive-pstricks-doc  
thailatex tk8.4 x11-utils x11-xserver-utils xbitmaps xdg-utils  
xfonts-encodings xfonts-utils xterm tex-gyre texlive-fonts-recommended
```

```
texlive-fonts-recommended-doc tipa ttf-marvosym  
0 upgraded, 108 newly installed, 0 to remove and 2 not upgraded.  
Need to get 783 MB of archives.  
After this operation, 1,433 MB of additional disk space will be used.  
Do you want to continue [Y/n]? y
```

インストールの確認が表示されるので、Y を入力して実行します。

TeXLive は非常に大きなパッケージなので、インストールには時間がかかります。

以上で、TeXLive のインストールは完了です。

PDF の出力

ReVIEW の環境が構築できたら、サンプルを PDF に変換してみましょう。

ReVIEW ツールを RubyGems を使ってインストールした場合は、gem の中にサンプルが含まれているのでコピーして、review-pdfmaker を実行します。

```
$ cp -rf /var/lib/gems/1.8/gems/review-1.1.0/test/sample-book ~/  
$ cd sample-book/src  
$ review-pdfmaker config.yml
```

上は、筆者の環境で ReVIEW をインストールした場合です。RubyGems や ReVIEW のバージョン、システムの設定によってはディレクトリが異なる場合があります。ReVIEW がインストールされているディレクトリを確認するには gem which review を実行します。

GitHub の場合は、リポジトリの中にサンプルが含まれているのでコピーして、review-pdfmaker を実行します。

```
$ cp -rf ~/review/test/sample-book ~/  
$ cd sample-book/src  
$ review-pdfmaker config.yml
```

2.4 Windows での環境構築

残念なことに、Windows に ReVIEW をインストールするのは容易ではありません。

第 1 章「ReVIEW 入門」でも述べたとおり、Windows 上で ReVIEW を動作させるには Cygwin などの環境を整備する必要があるのです。



図 2.4 Windows 8.1

開発者の武藤氏によって、VirtualBox 用の仮想イメージ（Debian）が公開されているので、そちらを使うのが最も簡単な方法です。

KeN's GNU/Linux Diary <http://d.kmuto.jp/20130811.html>

第3章

執筆を始める

本章では ReVIEW を用いて原稿を書いていく過程を紹介します。^{*1}

3.1 プロジェクトを作成する

まず作業するためのディレクトリ（フォルダ）を作ります。

```
> mkdir new-project  
> cd new-project
```

ここで、ReVIEW 記法で原稿を執筆するため、first-chapter.re を作ることにします。同名のファイルをエディタで開き（新規作成し）、執筆を開始しましょう。

まず章のタイトルを書くところから始めます。（リスト 3.1）

リスト 3.1: 章タイトルを作る

```
= ReVIEW を用いて執筆を始める
```

行の先頭に「=」とついています。これは「章タイトル」を示す ReVIEW の命令です。

命令を記述する際、全角文字と半角文字の違いについては注意してください。ReVIEW では、すべての命令は半角文字で入力します。他のマークアップ言語と同様、ReVIEW は全角の「@」を半角の「@」と同じようには処理してくれません。

タイトルを記述したファイルを HTML 形式にコンパイルしてみましょう。

review-compile は REVIEW 文法で書かれた原稿を指定されたフォーマットへ変換するコ

^{*1} 読者が UNIX 系 OS 上でコマンドやエディタの操作がある程度出来ることを想定しています。

マンドです。--target html と続けて指定することで、HTML 出力をするよう指定します。

```
> review-compile --target html new-project.re
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:ops="http://www.idpf.org/2007/ops"
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <meta http-equiv="Content-Style-Type" content="text/css" />
  <meta name="generator" content="ReVIEW" />
  <title>ReVIEW を用いて執筆を始める</title>
</head>
<body>
<h1><a id="h1"></a>第1章 ReVIEW を用いて執筆を始める</h1>
</body>
</html>
```

標準出力に HTML が出力されています。

HTML を直接読んでもどういう見た目になるのか分かりづらいので、ブラウザで出力結果を見るために test.html に一旦保存します。

リスト 3.2: ファイルにリダイレクトする

```
> review-compile --target=html new-project.re > test.html
```

このファイルを、Google Chrome で開くと、図 3.1 のような出力が出てきます。

第1章 ReVIEW を用いて執筆を始める

図 3.1 最初の出力

次に本文を書いてみましょう。本文は 2 つ段落で構成します。空行をはさむと別の段落になります。

リスト 3.3: 二つの段落を構成する

= ReVIEW を用いて執筆を始める

本章では ReVIEW を用いて原稿を書いていく過程を紹介します。

読者が UNIX 系 OS 上でコマンドやエディタの操作がある程度出来ることを仮定しています。

これをコンパイルすると図 3.2 のようになります。

第1章 ReVIEWを用いて執筆を始める

本章ではReVIEWを用いて原稿を書いていく過程を紹介します。

読者がUNIX系OS上でコマンドやエディタの操作がある程度出来ることを仮定しています。

図 3.2 段落を足す

節のタイトルを==を用いて足してみます。

リスト 3.4: 節を足します

= ReVIEW を用いて執筆を始める

本章では ReVIEW を用いて原稿を書いていく過程を紹介します。

読者が UNIX 系 OS 上でコマンドやエディタの操作がある程度出来ることを仮定しています。

== プロジェクトを作成する

結果は図 3.3 の通りです。

第1章 ReVIEWを用いて執筆を始める

本章ではReVIEWを用いて原稿を書いていく過程を紹介します。

読者がUNIX系OS上でコマンドやエディタの操作がある程度出来ることを仮定しています。

1.1 プロジェクトを作成する

図 3.3 節を足す

原稿を ReVIEW 記法でマークアップしながら、`review-compile` を繰り返し、「HTML でおおまかな見栄えを確認する」を繰り返しおこないます。

随時コンパイルしながら原稿を執筆する

脱線します。この項は無視して頂いて構いません。^{*2}

執筆中、何度もコンパイルしながら内容をブラウザで確認することを考えると、ファイルをブラウザから開くのは面倒です。出来れば、出力したら自動的にブラウザでそれを開いて欲しいですね。筆者の環境（Linux）でこれを達成するためには、例えば以下のようにコマンドを実行します。

リスト 3.5: Linux で Google Chrome を用いて出力を即座に見る例

```
> review-compile --target html new-project.re > test.html; google-chrome test.html
```

または別の方向性として、エディタでセーブするときに HTML を出力させる、ということも思いつきます。例として、vim というエディタで、ファイルを保存するたびに HTML ファイルを生成する、というハックが TechBooster の筆者内で共有されたことがあります。（リスト 3.6）

リスト 3.6: vim で原稿を保存した時に HTML を自動で生成する。`.vimrc` に書く。

^{*2} ちなみに==で「章」「節」に続く次の見出しである「項」を記述できます。この「項」のタイトルは「== 随時コンパイルしながら原稿を執筆する」と記述することで作られています。

```
au BufWritePost *.re silent execute
\ '!review-compile --target html' . expand('<afile>.')
\ . ' > ' . expand('<afile>:p:h') . '/html_m/'
\ . expand('<afile>:t:r') . '.html'
```

これは、例えば /opt/new-project/ ディレクトリで new-project.re というファイルを編集中なら、以下のコマンドをファイルの保存の度に手動で実行することとほぼ同じ効果を持ちます。^{*3}

```
> review-compile --target html new-project.re > /opt/new-project/html_m/new-project
```

こういった最適化にハマると原稿どころではなくなるのでここまでにしますが、ツールに習熟すると良いこともある事実は、覚えておいて良いかもしれません。

3.2 文を修飾する

UNIX 環境についての注意書きは本文ではなく脚注に記載したくなったとします。ここで登場するのが @<fn> と //footnote です。

リスト 3.7: 脚注を導入する

= ReVIEW を用いて執筆を始める
本章では ReVIEW を用いて原稿を書いていく過程を紹介します。 @<fn>{assume_unix_experience}

^{*3} 現在編集しているファイルをセーブした際 (BufWritePost イベント時) に、そのファイルが *.re という正規表現にマッチした場合、自動的に execute 以降に指定された命令を vim が実行します。 '!' が review-compile という文字列の前にあるため、vim は review-compile という文字列をシェル上で実行されるべき外部のコマンドと解釈します。 review-compile に渡される引数ですが、expand() は中に記述された文字列をルールとしてファイル名やらパス名に変換する vim 内の関数です。これによって生成される文字列とその他の文字列を「.」で連結しています。expand('<afile>..') は「現在開いているファイルの名前を実行ディレクトリからの相対パスで表現する」という意味、expand('<afile>:p:h') は「現在開いているファイルのフルパス (:p の機能) から最後のコンポーネントすなわちファイル名自体を削除 (:h) する」という意味で、要はファイルのあるディレクトリを示します。expand('<afile>:t:r') は逆に最後のコンポーネント (:t) の拡張子を削除したもの (:r) になります。

```
//footnote[assume_unix_experience][読者がUNIX系OS上でコマンドやエディタの操作がある程度出来ることを仮定しています。]
```

```
== プロジェクトを作成する
```

`assume_unix_experience`というキーワードを用いて、実際の脚注と、それにユーザを誘導する印との対応関係を記述します。なお、ReVIEW記法全体を通じてこういった「@<"命令">」や「//命令」といった命令が頻繁に登場します。

もう少し書き進めて、コマンドの実行例を紹介したい段階になったとします。コマンド実行例のための命令「//cmd」を使用してみます。

リスト3.8: //cmdを用いる

```
== プロジェクトを作成する  
まず作業するためのディレクトリ（フォルダ）を作ります。  
//cmd{  
  > mkdir new-project  
  > cd new-project  
  //}
```

ファイルの内容を表示するために、//listを用いてみます。`list`というと「箇条書き」のようなものを想像しますが、ここでの`list`は「プログラムリスト」「ソースコード」のようなニュアンスです。

リスト3.9: //listを用いる

```
== プロジェクトを作成する  
まず作業するためのディレクトリ（フォルダ）を作ります。  
//cmd{  
  > mkdir new-project  
  > cd new-project  
  //}
```

ここで、ReVIEW記法で原稿を執筆するため、`first-chapter.re`を作ることにします。同名のファイルをエディタで開き（新規作成し）、執筆を開始しましょう。

まず章のタイトルを書くところから始めます。（@<list>{first-project-content}）

```
//list[first-project-content] [章タイトル]{  
= ReVIEW を用いて執筆を始める  
//}
```

文中に「first-chapter.re」とあるのですが、これはファイル名です。プログラムのソースコードを文中に引用するような、若干異なるフォント（等幅フォント）等で表示して欲しいと考えたとします。ReVIEWでは「@<code>」命令によって、文中にそういった修飾を行うことができるので、今回はこれを用いてみます。

リスト 3.10: @<code>を使う

ここで、ReVIEW記法で原稿を執筆するため、@<code>{first-chapter.re}を作ることにします。同名のファイルをエディタで開き（新規作成し）、執筆を開始しましょう。

既にお気づきかもしれません、本章の例は本章の本文を執筆する際に使ってきたものが元になっています。記法についても同様です。

ただし、解説していない記法もあります。たとえば、本文冒頭は（少なくとも PDF 出力において）他の本文の文面と段落の構成方法が変わっています。いわゆる「リード文」のための ReVIEW 記法を使っているからです。

また、他の章を参照する際、その章の番号を調べて「第〇章」と書いていては、章構成が変わった時に対応できません。そのため、本章の原稿には章番号は直接書かれていません。代わりにファイルの章番号を埋め込む ReVIEW の命令を用いています。

もし興味があれば、第4章を参照するなどして、これらを達成する ReVIEW 記法を調べてみてください。

書きたい内容を決めて、ReVIEW 記法で記述して、HTML で出力を大まかに確認する、というサイクルについて紹介しました。執筆の例についてはここで止め、実際に HTML 以外の書籍データを作成してみましょう。

3.3 PDF に変換する

紙の同人誌として出版する場合に入稿フォーマットとして **PDF** (Portable Document Format) が使われることがあります。そこで、ここまでで記述した原稿から、HTML ではなく

＜PDF出力してみましょう。

PDF出力のためには外部ツールである TeX Live が必要です。インストールされているかを確認する一つの方法として、例えば以下のようなコマンドを実行します。

```
> plateax --version
e-pTeX 3.1415926-p3.3-110825-2.4 (utf8.euc) (TeX Live 2012/Debian)
kpathsea version 6.1.0
ptexenc 1.3.0
Copyright 2012 D.E. Knuth.
There is NO warranty. Redistribution of this software is
covered by the terms of both the e-pTeX copyright and
the Lesser GNU General Public License.
For more information about these matters, see the file
named COPYING and the e-pTeX source.
Primary author of e-pTeX: D.E. Knuth.
```

バージョンが出力されれば準備万端ということはありませんが、エラーが出るようであれば TeX Live の準備が出来ていないかもしれません。第2章を再確認してください。

review-pdfmaker を実行するにはもう少し準備が必要です。PDFで出力するために必要な「本のタイトル」や「筆者名」を記述するための YAML ファイルを用意します。

ここではとりあえず、ReVIEW の Ruby 実装が提供している YAML ファイルをそのまま使って見ることにしましょう。

```
> wget https://raw.github.com/kmuto/review/master/doc/sample.yaml
--1982-06-16 16:59:02--  https://raw.github.com/kmuto/review/master/doc/sample.yaml
raw.github.com (raw.github.com) を DNS に問い合わせています... 103.245.222.133
raw.github.com (raw.github.com)|103.245.222.133|:443 に接続しています... 接続しました。
HTTP による接続要求を送信しました、応答を待っています... 200 OK
長さ: 2864 (2.8K) [text/plain]
`sample.yaml' に保存中

100%[=====] 2,864          ---K/s 時間 0s

2101-12-04 18:19:23 (17.0 MB/s) - `sample.yaml' へ保存完了 [2864/2864]
> cat sample.yaml
# review-epubmaker 向けの設定ファイルの例。
# yaml ファイルを ReVIEW ファイルのある場所に置き、
```

```
# 「review-epubmaker yaml ファイル」を実行すると、<bookname>.epub ファイルが
# 生成されます。
# このファイルは UTF-8 エンコーディングで記述してください。

# ブック名(ファイル名になるもの。ASCII 範囲の文字を使用)
bookname: review-sample
# 書名
booktitle: ReVIEW EPUB サンプル

... (以下省略)
```

得られた sample.re を用いて review-pdfmaker を実行します。

```
> review-pdfmaker sample.yaml
compiling new-project.tex
No such directory - /tmp/new-project/sty
This is e-pTeX, Version 3.1415926-p3.3-110825-2.4 (utf8.euc) (TeX Live 2012/Debian)
 restricted \write18 enabled.
entering extended mode
./book.tex
pLaTeX2e <2006/11/10> (based on LaTeX2e <2011/06/27> patch level 0)
Babel <v3.8m> and hyphenation patterns for english, dumylang, nohyphenation, pi
nyin, thai, loaded.
(/usr/share/texlive/texmf-dist/tex/plateX/jsclasses/jsbook.cls
Document Class: jsbook 2010/03/14 okumura
) (/usr/share/texlive/texmf-dist/tex/plateX/japanese-otf-upTeX/otf.sty
(/usr/share/texlive/texmf-dist/tex/plateX/japanese-otf/ajmacros.sty))
(/usr/share/texlive/texmf-dist/tex/lateX/graphics/color.sty

.. (省略)

./book.aux )
Output written on book.dvi (2 pages, 1436 bytes).
Transcript written on book.log.
book.dvi -> book.pdf
[1] [2]
3250 bytes written
```

ここで大量のログ出力されますが、正常です。内部で TeX Live のコマンド等を実行する際、例え1文字の原稿をコンパイルして PDF にするときにも多くのログが表示されます。大

第3章 執筆を始める

半は TeX 絡みの出力です。

コマンド実行が終了したら PDF が出来たかを確認します。

```
> ls *.pdf  
review-sample.pdf
```

今回は `review-sample.pdf` というファイルが出力されています。なお、使用した YAML ファイル（今回は `sample.yaml`）の、`bookname` という設定に応じてこのファイル名は変わります。

さて、PDF リーダで見てみると……

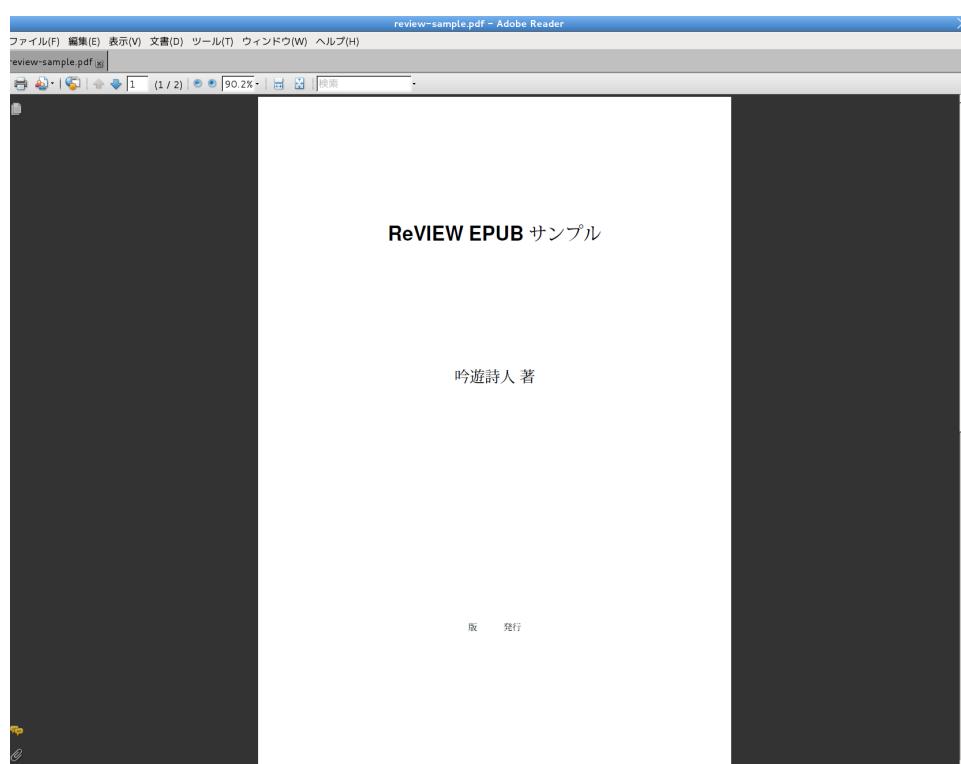


図 3.4 表紙っぽいものは出た……が、表紙と目次の 2 ページしかない

自分の原稿が取り入れられていません(´・ω・`)

catalog.yml を導入する

ここまでで行なってきたことを振り返ってみます。

- new-project/ ディレクトリを作成
- その配下に new-project.re を作成
- review-compile で HTML を出力しつつ原稿を執筆
- sample.yaml を取得
- review-pdfmaker sample.yml を実行
- new-project.re が生成された PDF に含まれていないことを確認

この問題は、new-project.re というファイルが、本を構成するファイルとして認識されないために起こります。例えば仮に second-chapter.re というファイルで「第二章」を作った場合には、どちらが先なのかも指定しなければなりません。

そういういた指定を行うため、catalog.yml というファイルを新たに作成します。そして、そのファイルに、本に収録する ReVIEW 形式のファイルを記述します。

```
> ls
catalog.yml  first-chapter.re  review-sample.pdf  sample.yaml
> cat catalog.yml
CHAPS:
  - first-chapter.re
> rm review-sample.pdf
> review-pdfmaker sample.yaml
(出力省略)
```

これで章として認識されるはずです(`・ω・') 図 3.5



図 3.5 原稿を取り入れた後の PDF の例

現時点ではタイトルがお好みのものではないかと思います。`sample.yaml` を編集して、自著の書籍名に変えるといった作業を行います。

catalog.yml における **PREDEF**、**POSTDEF**、**APPENDIX**、**PART** の解説

`catalog.yml` は本文の章構成を決めるファイルで、ReVIEW では「カタログファイル」と呼びます。また、`catalog.yml` には、CHAPS を含め、書籍を構成するファイルをそれぞれ記述します。

本章で詳しくは紹介しませんが、カタログファイルに記述するものが他にもあるのでここで紹介します。

- PREDEF は前付けに用いる ReVIEW ファイルを列挙します
- POSTDEF は後付けに用いる ReVIEW ファイルを列挙します
- APPENDIX は付録に用いる ReVIEW ファイルを列挙します
- PART は書籍に「部」を導入するときに使われます。

しっかりした書籍を作る際にはこれらを駆使することになりますが、本章を読まれている時点では「そういうものも存在する」くらいの理解で良いと思います。CHAPS のみうまく扱えるようにしましょう。

3.4 EPUB ファイルを作る

次に電子書籍において一般的な EPUB 形式の出力を試します。まず、EPUB 作成に必要な zip コマンドが利用できることを確認します。

```
> zip -L
Copyright (c) 1990-2008 Info-ZIP. All rights reserved.

For the purposes of this copyright and license, "Info-ZIP" is defined as
the following set of individuals:

.. (以下省略。ライセンスが表示されています)
```

YAML ファイルを指定して review-epubmaker コマンドを実行します。

```
> review-epubmaker sample.yaml
adding: mimetype (stored 0%)
adding: META-INF/container.xml (deflated 29%)
adding: OEBPS/review-sample.html (deflated 46%)
adding: OEBPS/review-sample.opf (deflated 54%)
adding: OEBPS/first-chapter.html (deflated 40%)
adding: OEBPS/top.html (deflated 44%)
adding: OEBPS/stylesheets.css (deflated 3%)
adding: OEBPS/review-sample.ncx (deflated 48%)
> ls *.epub
review-sample.epub
```

今回は、Google Chrome の拡張として利用できる電子書籍リーダ「Radium^{*4}」で、出来上がった電子書籍ファイル「review-sample.epub」を開いてみましょう。(図 3.6)

*4 <http://readium.org/>



第1章 ReVIEWを用いて執筆を始める

本章ではReVIEWを用いて原稿を書いていく過程を紹介します。

読者がUNIX系OS上でコマンドやエディタの操作がある程度出来ることを仮定しています。

1.1 プロジェクトを作成する

図 3.6 作成した EPUB ファイルを Radium で開いた例

3.5 おわりに

本章では ReVIEW を用いて原稿を執筆する例を紹介しました。プロジェクト作成から PDF・EPUB の作成まで、出来るようになったかと思います。

実際の執筆作業を続けていく上では、ReVIEW の記法や細かい点について、より詳細に理解する必要が発生します。そのようなとき、本書の他の章が参考になるはずです。

- 記法の詳細は第4章「記法をおぼえよう」にあります。
- 出力の見栄えについては第5章「スタイル解説」で詳述されています。
- ReVIEW コマンドの詳細は第6章「コマンド解説」にあります。
- 他の ReVIEW 利用例を見てみたい場合、第8章「仕様書を作ろう」を参照するのが良いでしょう。
- ReVIEW 記法の技術的理をひたすら深めたい方は第9章「ReVIEW.js で学ぶ ReVIEW 記法のお約束」が参考になるはずです。

各章は独立していますので、必要に応じてトピックを選んで読むのが良いでしょう。
それでは、ReVIEW で素敵な進歩を。

第4章

記法をおぼえよう

本章では、ReVIEWで執筆をする上で最も重要なReVIEW記法について解説します。

4.1 ReVIEW 記法

ReVIEW記法は、執筆した文章をマークアップするための記法であり、ReVIEWの出力結果の最終的な仕上がりを左右する最も重要な要素の一つです。

例えば、技術書では、本文にクラスやメソッドの名前を記述する場合、その部分をイタリックにしたり等幅フォントにすることで、視覚的に差別化することは一般的に行われています。

リスト4.1: 適切な番号の割り当て+キャプションを表示

また、画像やプログラムリストを掲載する場合、視覚的な差別化はもとより、本文から十分な余白をとり、かつ適切な場所に配置することも大切です。

さらに、掲載した画像やリストに本文で言及するために、それぞれ一意な番号を割り当てます（リスト4.1）。

ReVIEW記法でマークアップすることで、文章の構造や見た目を指定できます。マークアップの巧拙が表現力の差となり、高い表現力は、読者にとってわかりやすい文章につながります。

本章では、執筆にあたって適切なマークアップを選択できるように、利用頻度の高いReVIEW記法について解説します。

また本書には、付録としてReVIEW記法のチートシートを収録しています。そちらもあわせて参照してください。

4.2 インライン命令とブロック命令

ReVIEW 記法は、「インライン命令」と「ブロック命令」の2つに分類できます。

インライン命令

インライン命令は、@<命令>{...}のように、@（アットマーク）に続けて括弧<>内に命令名を指定します。続く括弧{}の中が、インライン命令の効果が及ぶ範囲になります。

インライン命令は、文章の装飾やリストや図の参照など表示や内容に影響します。文中に直接記入することができますが、改行をまたぐことはできません。

ブロック命令

ブロック命令は、リスト 4.2 のように、スラッシュ 2つ（//）に続けて命令名を指定します。続く括弧{から、次の//}までが、ブロック命令の効果が及ぶ範囲になります。

リスト 4.2: ブロック命令

```
//命令{  
...  
//}
```

ブロック命令には、複数のオプションを指定できる場合があります。オプションは、命令に続けて括弧[]の中に記述します（リスト 4.3）。

指定可能なオプションの数や省略の可否は、命令によって違います。

リスト 4.3: ブロック命令

```
//命令 [オプション 1] [オプション 2] [オプション 3]{  
...  
//}
```

ブロック命令は、リストや図など本文とは独立した内容を掲載する際に使います。

ブロック命令を文の途中から開始することはできません。ブロック命令の開始と終了は、必

ず行頭に記述する必要があります。行頭に半角スペースなどが入った場合、ReVIEWは、その行をブロック命令として扱いません。

4.3 見出し

見出しが、=から始めて、キャプションを続けます。

=は、必ず行頭から開始する必要があります。また、=の後ろには、必ず半角スペースを入れる必要があります。

=の数によって見出しの深さが変わります。数に応じてそれぞれ章、節、項、段、小段の5段階に対応します（リスト4.4）。

リスト4.4: キャプションと対応

= 章のキャプション

== 節のキャプション

==== 項のキャプション

===== 段のキャプション

===== 小段のキャプション

ReVIEWは、6段階以上の見出しには対応していませんが、一般的な文書では「項」レベルまであれば十分でしょう。

章、節、項それぞれの見出しで含まれる内容のバランスを保つことで、読みやすい構造になります。

参照

他の見出しを参照することができます。

見出しの参照を使わず、見出しのキャプションや番号を記述した場合、見出しの追加や削除、順番の変更が発生するたびに、変更のあった見出しに言及しているファイルを漏れなく探し出し、関係する箇所を全て変更しなくてはなりません。

一方、見出しの参照を使うと、順番を入れ替えるなどして見出しの番号が変わっても、変

更は自動的に反映されます。

見出しの参照は、大きく「章の参照」と「節の参照」の2つに分類できます。

章の参照

章を参照する構文です。

リスト4.5は、3つとも第1章「ReVIEW入門」を参照しています。

リスト4.5: 章の参照

```
@<title>{introduction}  
@<chap>{introduction}  
@<chapref>{introduction}
```

titleは、その章のキャプションになります（ReVIEW入門）。

chapは、その章の見出し番号になります（第1章）。

chaprefは、その章の見出し番号とキャプションになります（第1章「ReVIEW入門」）。

これらの構文の参照先には、ReVIEWの章の識別子を指定します。

章の識別子は、ReVIEWの章のファイル名から拡張子.reを取り除いた名前です。例えば、introduction.reの章を参照する場合、introductionが識別子になります。

節の参照

章より一段深い階層にある「節」を参照する構文です。

リスト4.6: 見出しの参照

```
@<hd>{見出し}  
@<hd>{introduction|ReVIEWの特色}
```

同じ章の節を参照する場合、参照先には、節のキャプションだけを記載します。

異なる章の見出しを参照する場合、章の識別子を指定した上で、縦棒|で区切って節のキャプションを記述します。

章の識別子は、ReVIEW の章のファイル名から拡張子.re を取り除いた名前です。例えば、introduction.re の章を参照する場合、introduction が識別子になります。

見出しラベル

節の参照では、見出しのキャプションをそのまま記述するので、キャプションを変更したときに参照を更新する必要があります。

ReVIEW では、見出しキャプションとは別に、参照用のラベルを指定できます。

リスト 4.7: 見出しの参照用ラベル

```
=={review_markup} ReVIEW 記法
```

リスト 4.7 の括弧{}の中が節の識別子になり、リスト 4.8 のように参照します。

リスト 4.8: 見出しの参照

```
@<hd>{review_markup}
```

通常の節の参照同様、違う章の節を参照することもできます。

4.4 リード文

ひとまとめりの文章の内容を、簡単に説明する短い文章を「リード文」と言います。

リスト 4.9: リード文

```
//lead{  
本章では、ReVIEW で執筆をする上で最も重要な ReVIEW 記法について解説します。  
//}
```

リスト 4.9 のマークアップは、以下のように出力されます。

本章では、ReVIEW で執筆をする上で最も重要な ReVIEW 記法について解説します。

このように、リード文としてマークアップした内容は、本文に比べて左側の余白が大きくなります。

ReVIEWでは、リード文のマークアップは、文章のどこにでも置くことができます。しかし、基本的には見出し、特に章見出しの直下に置くことを想定しています。

4.5 段落と改行

文章と文章の間に空行を挟むと、それぞれが個別の段落として処理されます。

2つ以上空行を入れても段落分けには影響しません。

また、ReVIEWでは、空行を挟まない改行は無視され、出力には影響しません。1つ以上の半角スペースも、出力には影響しません。

@
{}を使うと、強制的に
改行
できます。

リスト 4.10: 段落を含む文章

ReVIEW記法の段落は、空行を挟んで文章を続けます。

2つ以上空行を入れても段落分けには影響しません。

また、ReVIEWでは、
空行を挟まない改行は無視され、出力には影響しません。
影響しません。

1つ以上の半角スペースも、出力には

@
{}を使うと、強制的に@
{}
改行@
{}
できます。

4.6 コメント

行が#@#から始まる場合、その行についてはコメントとして扱われ、最終出力には影響しません。

ReVIEWは、複数行コメントには対応していません。複数行をコメントとして扱うには、

全ての行に#@#を含める必要があります。

リスト 4.11: コメント

コメントの内容は、出力に影響しません。

#@#この行はコメントです。

ReVIEW は、複数行コメントには対応していません。

複数行をコメントとして扱うには、全ての行に#@#を記述する必要があります。

4.7 箇条書き

項目を列挙する際に、箇条書きを利用できます。

ReVIEW は、「番号なし箇条書き」と「番号つき箇条書き」に対応しています。

番号なし箇条書き

*に続けて項目を続けると箇条書きになります。番号なし箇条書きは入れ子に対応しており、*の数が階層になります。

*の前後には、半角スペースが必要です。半角スペースがない場合は、ReVIEW はその行を本文として取り扱います。

リスト 4.12: 番号無し箇条書き

```
* 箇条書き
** 2 層目
*** 3 層目
** 2 層目
* 箇条書き
```

リスト 4.12 のマークアップは、以下のように出力されます。

- 箇条書き
 - 2 層目
 - * 3 層目

- 2層目

- 箇条書き

番号あり箇条書き

1. に続いて項目を続けると箇条書きになります。

先頭とピリオドの後に半角スペースが必要です。番号あり箇条書きは、入れ子に対応していません。

リスト 4.13: 番号付き箇条書き

1. 番号つき箇条書き
2. 番号つき箇条書き
3. 番号つき箇条書き

リスト 4.13 のマークアップは、以下のように出力されます。

1. 番号つき箇条書き
2. 番号つき箇条書き
3. 番号つき箇条書き

また番号部分の連番を誤った場合でも自動的に連番に置き換わります。

リスト 4.14: 番号付き箇条書き

1. 番号つき箇条書き
2. 番号つき箇条書き
2. 番号つき箇条書き

リスト 4.14 のマークアップは、以下のように出力されます。

1. 番号つき箇条書き
2. 番号つき箇条書き
3. 番号つき箇条書き

このように、番号部分は自動的に連番に置き換わります。しかし、行の追加や削除の際は、

可能な限り番号を書き換えることをおすすめします。組版や校正時のミスを予防できます。

4.8 リスト

プログラムコードなど、本文とは分けて掲載する内容を「リスト」と言います。リストの中の改行やスペースはそのまま出力されます。

「4.6 段落と改行」で述べたとおり、ReVIEWでは本文中の複数の半角スペースや改行は、本文には影響しません。しかし、プログラムなどを掲載したい場合、改行やインデントなどはそのまま出力する必要があります。このような場合にリストは有効です。

ReVIEWのリストには、連番付きと連番無しの2種類、行番号の有無を含めると計4種類があります。

表 4.1 リストの区分

	行番号なし	行番号あり
連番付き	list	listnum
連番なし	emlist	emlistnum

連番つきリストにすると、ReVIEWは、リストごとに一意な番号を割り振ります。

リスト 4.15: 連番付きリスト

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
}
```

番号が割り当てられたリストは、リスト 4.16 のように本文から参照できます。

リスト 4.16: リストの参照

番号が割り当てられたリストは、@<list>{list_sample_with_seq}のように本文から参照できます。

一方、連番なしリストは、リストに番号を割り当てません。

連番無しリスト

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
}
```

番号が割り当てられていない場合、本文からは参照できません。

一般的に、ソースコードリストのように本文から参照する場合には、連番付きリスト（list）を推奨します。参照関係が明らかな場合のみ、連番なしリスト（emlist）を利用するといいでしょう。

連番つきリスト

連番付きのリストです。

キャプションには、ソースコードファイル名や処理内容など、リストの示す内容を簡潔に記述します。連番付きリストのキャプションは省略できません。

リスト 4.17: 連番付きリスト

```
//list[identifier] [連番付きリストのキャプション]{  
    ... ソースコード等...  
}
```

リスト 4.19 のマークアップは、以下のように出力されます。

リスト 4.18: 連番付きリストのキャプション

```
... ソースコード等...
```

@<list>{identifier}で、**identifier**（識別子）が示すリストを参照できます。参照は、出力時に「リスト 4.17」のようにリスト番号に置き換わります。

連番なしリスト

連番なしのリストです。

キャプションには、ソースコードファイル名や処理内容など、リストの示す内容を簡潔に記述します。連番なしリストのキャプションは省略できます。

リスト 4.19: 連番なしリスト

```
//emlist[連番なしリストのキャプション（省略可能）]{  
    ... ソースコード等...  
}
```

リスト 4.19 のマークアップは、以下のように出力されます。

連番なしリストのキャプション（省略可能）

```
... ソースコード等...
```

連番無しリストは、本文中からの参照はできません。

連番つきリスト（行番号有り）

連番付きのリストで、かつ、自動的に行番号が表示されます。

キャプションには、ソースコードファイル名や処理内容など、リストの示す内容を簡潔に記述します。連番付きリストのキャプションは省略できません。

リスト 4.20: 行番号有り連番付きリスト

```
//listnum[identifier][行番号有り連番付きリストのキャプション]{  
    ...  
    ソースコード等  
    ...  
}
```

リスト 4.20 のマークアップは、以下のように出力されます。

リスト 4.21: 行番号有り連番付きリストのキャプション

```
1: ...
2: ソースコード等
3: ...
```

@<list>{identifier}で、 **identifier**（識別子）が示すリストを参照できます。参照は、出力時に「リスト 4.20」のようにリスト番号に置き換わります。

連番なしリスト（行番号有り）

連番なしのリストで、かつ、自動的に行番号が表示されます。

キャプションには、ソースコードファイル名や処理内容など、リストの示す内容を簡潔に記述します。連番なしリストのキャプションは省略できます。

リスト 4.22: 行番号有り連番なしリスト

```
//emlistnum[行番号有り連番なしリストのキャプション（省略可能）]{
...
ソースコード等
...
//}
```

リスト 4.22 のマークアップは、以下のように出力されます。

行番号有り連番なしリストのキャプション（省略可能）

```
1: ...
2: ソースコード等
3: ...
```

連番無しリストは、本文中からの参照はできません。

行番号をつける場合は、通常、本文からの参照・説明を意図していることが多く、また、後述する不具合があるので、利用頻度がきわめて低い構文です。

4.9 コマンドライン

コマンドラインの入出力などを示す構文です。

リスト 4.23: 引用

```
//cmd{
$ git add .gitignore
$ git commit -m "first commit"
$ git push
Counting objects: 10, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 25.21 KiB | 0 bytes/s, done.
Total 6 (delta 4), reused 0 (delta 0)
To git@github.com:TechBooster/C85-ReVIEW.git
  261af41..9c23bd9  master -> master
//}
```

リスト 4.23 のマークアップは、以下のように出力されます。

```
$ git add .gitignore
$ git commit -m "first commit"
$ git push

Counting objects: 10, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 25.21 KiB | 0 bytes/s, done.
Total 6 (delta 4), reused 0 (delta 0)
To git@github.com:TechBooster/C85-ReVIEW.git
  261af41..9c23bd9  master -> master
```

ReVIEW は、コマンドラインの中の改行やスペースをそのまま出力します。

また、コマンドラインは、本文中からの参照はできません。

4.10 引用

ウェブサイトや書籍などから引用した文章を示す構文です。

リスト 4.24: コマンドライン

```
//quote{  
文章を記述するための軽量マークアップ言語は ReVIEW の他にもいくつもあります。  
例えば、 Wikipedia などで使われる Wiki 記法も軽量マークアップ言語です。  
IT 技術者業界では、他にも Markdown, reStructuredText, Textile といった例があります。  
//}
```

リスト 4.24 のマークアップは、以下のように出力されます。

文章を記述するための軽量マークアップ言語は ReVIEW の他にもいくつもあります。
例えば、 Wikipedia などで使われる Wiki 記法も軽量マークアップ言語です。 IT 技術者
業界では、他にも Markdown, reStructuredText, Textile といった例があります。

ReVIEW は、引用の中の改行やスペースをそのまま出力します。

また、引用は、本文中からの参照はできません。

4.11 リンク

URL をリンクとしてマークアップする構文です。

リンクには、出力したときに URL を表示する場合と、しない場合の 2 通りの方法があります。

リスト 4.25 は、URL を表示する場合です。

リスト 4.25: URL リンク

```
@<href>{http://techbooster.org}
```

リスト 4.25 を出力した結果は、 "http://techbooster.org" のように URL がそのまま表示されます。

また、リスト 4.26 は、URL を表示しない場合です。

URL に加えて、コンマで区切ってリンクを設定する文字列を続けます。

リスト 4.26: リンク（URL 無し）

```
@<href>{http://techbooster.org,TechBooster}
```

リスト 4.26 を出力した結果は、"TechBooster"のように URL が表示されず、文字列にリンクが設定された状態になります。

当然のことながら、どちらの場合でも紙に印刷するとリンクは使えなくなりますが、URL を表示する方法では印刷後でも URL そのものはわかるので、最終的に印刷する場合は、URL を表示する方法が適しています。

4.12 脚注

本文に別の情報を付記する構文です。「注釈」とも呼ばれます。

リスト 4.27 は、脚注を記述した例です。

リスト 4.27: コマンドライン

```
//footnote[identifier] [脚注の内容は、ページで本文とは別の「脚注の領域」に表示されます]
```

脚注は、記述した場所に関係なく「脚注の領域」に表示されます。

脚注を有効にするには、本文で参照している必要があります。本文で参照されていない脚注は出力されません。

リスト 4.28 では、記述した脚注を参照しています。

リスト 4.28: 脚注への参照

本文から参照すると、脚注番号が付記されます@<fn>{identifier}。

@<fn>{identifier}で、**identifier**（識別子）が示す脚注を参照できます。

本文から参照すると、脚注番号が付記されます^{*1}。

4.13 図

写真や挿絵などを「図」として、本文中に差し込むことができます。

差し込む写真や図面などは別ファイルで保存しておき、本文中に参照を記述します。

画像ファイル

本文に差し込む画像ファイルは、CHAPS があるディレクトリに `images` ディレクトリを作成します。それぞれの画像ファイルは、その画像を差し込む章の名前のディレクトリの下に配置します。

例えば、`markup.re` の章に表示する画像ファイル `enlightened.png` のばあい、`images/markup/enlightened.png` となります。このことから解るとおり、ReVIEW では、異なる章の画像ファイルを直接参照することはできません。画像ファイルは、章ごとに配置する必要があります。

画像ファイルとして利用できるフォーマットは、ReVIEW で出力する形式に依存しますが、PNG、JPEG、**SVG** (Scalable Vector Graphics) など、基本的なフォーマットには対応しています。

^{*1} 脚注の内容は、ページで本文とは別の「脚注の領域」に表示されます



図 4.1 enlightened.png

連番付きの図

キャプションには、ファイル名や処理内容など、内容を簡潔に記述します。キャプションは省略できません。

リスト 4.29: 連番付きの図

```
//image[enlightened] [連番付きの図]{  
... 代替テキスト等...  
//}
```

リスト 4.29 のマークアップは、以下のように出力されます。



図 4.2 連番付きの図

@{identifier}で、本文から **identifier**（識別子）が示す図を参照できます。

参照は、出力時に「図 4.2」のようにリスト番号に置き換わります。

代替テキストは、出力時には無視されます。基本的にはアスキーアートや文章で画像の説明をして、執筆時にどんな画像を意図しているのか確認するために使いますが、何も書かなくても問題ありません。

連番なしの図

キャプションには、ファイル名や処理内容など、内容を簡潔に記述します。キャプションは省略できません。

リスト 4.30: 連番なしの図

```
//indepimage [enlightened] [連番なしの図]
```

リスト 4.30 のマークアップは、以下のように出力されます。



図: 連番なしの図

連番なしの図は、本文中からの参照はできません。また、代替テキストは指定できません。

図のサイズを指定する

スケールオプションで図の表示倍率を指定することができます。

スケールオプションは、`image`、`indepimage` に括弧 [] を追加して、`scale=`に続けて表示倍率を指定します。

表示倍率は、1.0 を等倍、0.0 を下限として指定できます。

リスト 4.31: 連番付きの図

```
//image[enlightened] [表示倍率 50%] [scale=0.5]{  
... 代替テキスト等...  
//}
```



図 4.3 表示倍率 50%

リスト 4.32: 連番なしの図

```
//indepimage [enlightened] [表示倍率 25%] [scale=0.25]
```



図: 表示倍率 25%

スケールオプションを省略した場合、ReVIEW は、自動で適切なサイズを選択します。スケールを指定した結果、図が紙面からはみ出る場合もあるので、注意してください。

4.14 表

罫線で区切られた表をマークアップする構文です。

ReVIEW では、行は改行、セルとセルの間にはタブで区切ります。

空白のセルは、ピリオド (.) を入力します。

セルにピリオドを表示したい、またはセルの内容をピリオドから始めたい場合、ピリオドを2つ続けて (..) 入力します。

また、-----で区切ることで、ヘッダの行を明確に指定できます。

リスト 4.33: 表

```
//table[identifier][表のキャプション]{
.     A      B      C      D      E
-----
1     2012   500    800    1.0    ..15
2     2013   600    .       1.1    ..13
3     2014   900    1200   1.4    ..18
//}
```

リスト 4.33 のマークアップは、表 4.2 のように出力されます。

表 4.2 表のキャプション

	A	B	C	D	E
1	2012	500	800	1.0	.15
2	2013	600		1.1	.13
3	2014	900	1200	1.4	.18

@<table>{identifier}で、 **identifier**（識別子）が示す表を参照できます。参照は、出力時に「表 4.2」のように表番号に置き換わります。

4.15 文字の装飾

ReVIEW 記法では、文字を装飾することができます。

文字の装飾は、基本的には、適用したい範囲を@<修飾子>{...}で指定するだけです。

また、文字の装飾は、本文だけでなく、リストや脚注、表などの中でも有効です。

比較的、利用頻度の高い文字の装飾について表 4.3 にまとめています。

表 4.3 文字の装飾

修飾子	説明	例
@{文字列}	太字（ボールド）フォント	abcdefg あいうえお
@<i>{文字列}	イタリックフォント	<i>abcdefg</i> あいうえお
@<tt>{文字列}	等幅フォント	<tt>abcdefg</tt> あいうえお
@<ttb>{文字列}	太字（ボールド）&等幅フォント	abcdefg あいうえお
@<tti>{文字列}	イタリック&等幅フォント	<i>abcdefg</i> あいうえお
@{文字列}	強調	abcdefg あいうえお
@{文字列}	強調	abcdefg あいうえお
@<ami>{文字列}	網掛け	 abcdefg あいうえお
@<kw>{キーワード, 解説}	キーワードと解説	GNU (GNU is Not Unix)

第5章

スタイル解説

ReVIEW で生成する原稿の見た目を、美しくブラッシュアップしたくはありませんか？

本章では、ReVIEW で生成される各種ファイルの見た目を変更する方法と、実際にカスタマイズする方法を紹介します。

5.1 スタイルファイルの種類

第1章で紹介した通り、ReVIEW は、書籍の出版に適したテキストベースの軽量マークアップ言語「ReVIEW 記法」と、ReVIEW 記法でマークアップした原稿を処理するツール類「ReVIEW ツール」の総称です。

ReVIEW は、ReVIEW 記法でマークアップされたファイル（以下、「ReVIEW ファイル」といいます）を、TEXT/HTML/EPUB/Markdown/IDGXML^{*1} および LaTeX (PDF) の形式で出力します。

本章では、ReVIEW が出力する各種ファイルの形式をはじめ、表示内容に影響する入力項目をまとめて「スタイル」と呼びます。

また、スタイルを設定するファイルをまとめて「スタイルファイル」と呼びます。

ReVIEW のスタイルファイルには、以下のものがあります。

.yml

ReVIEW で最も重要なスタイルファイルが.yml ファイルです。

.yml ファイルは、YAML^{*2} 形式で記述し、書籍のタイトルや表画像、ページの大きさや目次

^{*1} Adobe InDesign CS2 以降向け XML 形式

^{*2} YAML Ain't a Markup Language: 構造化データやオブジェクトを文字列にシリアル化（直列化）するためのデータ形式の一種。

に含める見出しの深さなど、ReVIEW ファイルを変換する際に必要となる設定を記述します。

リスト 5.1 は、本書で使っている.yml ファイルです。

リスト 5.1: config.yml

```
# review-epubmaker 向けの設定ファイルの例。
# yaml ファイルを ReVIEW ファイルのある場所に置き、
# 「review-epubmaker yaml ファイル」を実行すると、<bookname>.epub ファイルが
# 生成されます。
# このファイルは UTF-8 エンコーディングで記述してください。

# ブック名(ファイル名になるもの。ASCII 範囲の文字を使用)
bookname: book
# 書名
booktitle: はじめての ReVIEW
# 著者
aut: TechBooster
prt: 日光企画
prt_url: http://techbooster.org/
edt: mhidaka
dsr: yuya
date: 2013-12-10
rights: |
  (C) 2013 techbooster.org
description: ReVIEW 執筆経験者による執筆ノウハウをまとめました。
# 以下はオプション
#prt: 出版社
#asn: Associated name
#ant: Bibliographic antecedent
#clb: 貢献者
#ill: イラストレータ
#pht: 撮影者
#trl: 翻訳者
#rights: 権利表記
#description: ブックの説明
#
# coverfile: カバーページの body 要素内に挿入する内容を記述したファイル名
coverfile: _cover.html
# coverimage: 表紙用画像ファイル
coverimage: cover.jpg
# 固有 ID に使用するドメイン
urnid: http://techbooster.org/book/Beginners-ReVIEW
# CSS ファイル(yaml ファイルおよび ReVIEW ファイルを置いたディレクトリにあること)
stylesheet: main.css
# LaTeX 用の documentclass を指定する
```

```
texdocumentclass: ["jsbook", "b5j,twoside,openany"]
texstyle: techbooster-doujin
# 目次として抽出するレベル
toclevel: 2
# セクション番号を表示するレベル
secnolevel: 2
# review-compile に渡すパラメータ
params: --stylesheet=main.css
# 目次生成用？
mytoc: true
# 奥付生成用
colophon: true
pubhistory: |
  2013年12月31日 初版発行 v1.0.0
debug: true
```

.yml ファイルは、ReVIEW のコンパイル時に--yaml オプションで指定します。

```
review-compile --target=html --yaml=config.yml style_customize.re
```

.sty

.sty は、LaTeX (PDF) を出力する際に必要な「組版に関する設定」を記述するファイルで、LaTeX のスタイル形式で記述します。

リスト 5.2 は、本誌で採用している.sty ファイルです。

リスト 5.2: sty/techbooster.sty

```
%% サンプルコードを更に小さく
\renewenvironment{reviewemlist}{%
  \medskip\footnotesize\begin{shaded}\setlength{\baselineskip}{1.2zw}\begin{alltt}}{%
  \end{alltt}\end{shaded}}
\renewenvironment{reviewlist}{%
  \begin{shaded}\footnotesize\setlength{\baselineskip}{1.2zw}\begin{alltt}}{%
  \end{alltt}\end{shaded}\par\vspace*{0.5zw}}
```

```
\renewenvironment{reviewcmd}{%
  \color{white}\medskip\footnotesize\begin{shadedb}\setlength{%
    \baselineskip}{1.2zw}\begin{alltt}}{%
  \end{alltt}\end{shadedb}}
```

```
%% from review-pdfmaker
\usepackage{fancyhdr}
\usepackage{ulem}
\pagestyle{fancy}

\fancyhead{}
\fancyhead[LE]{\gtfamily\sffamily\bfseries\upshape \leftmark}
\fancyhead[R0]{\gtfamily\sffamily\bfseries\upshape \rightmark}
\cfoot{\thepage}

\renewcommand{\sectionmark}[1]{\markright{\thesection^#1}{}}
\renewcommand{\chaptermark}[1]{%
  \markboth{\prechaptername\ \thechapter\ \postchaptername^#1}{}}
\renewcommand{\headfont}{\gtfamily\sffamily\bfseries}

\fancypagestyle{plainhead}{%
\fancyhead{}
\fancyfoot{} % clear all header and footer fields
\fancyfoot[CE,C0]{\thepage}
\renewcommand{\headrulewidth}{0pt}
\renewcommand{\footrulewidth}{0pt}}
```

```
\hypersetup{colorlinks=false}
%%Helvetica を使う
\renewcommand{\sfdefault}{phv}
```

```
\sloppy
```

```
\cfoot{\thepage}
```

```
\def\cleardoublepage{%
\clearpage%
\if@twoside%
  \ifodd \c@page \else \hbox{}\thispagestyle{plainhead}\newpage%
    \if@twocolumn\hbox{}\thispagestyle{plainhead}\newpage\fi%
  \fi%
\fi%
}
```

スタイルの指定は、.yml の `texstyle` の項目に、適用するスタイルファイルの名前を指定して行います。

PDF 出力における ReVIEW の役割

PDF を出力するプロセスにおいて ReVIEW は、ReVIEW ファイルを LaTeX のソースファイルに変換する以上のこととはしていません。

実際に、`review-compile` コマンドの出力の形式を指定するオプション`--target` に "PDF" の値を指定できません。

しかし、`review-pdfmaker` コマンドを使った場合、ReVIEW ファイルを PDF として出力しています。`review-pdfmaker` コマンドは、どのような処理をしているのでしょうか。

`review-pdfmaker` コマンドは、`review-compile` コマンドの出力形式に `latex` と指定することで、いったん LaTeX のソースファイルを出力しています。

ここで得られた LaTeX のソースファイルを LaTeX のツールで処理して DVI ファイルに変換。最後に、DVI ファイルを PDF 形式に変換するという一連のプロセスを自動化したものが、`review-pdfmaker` コマンドなのです。

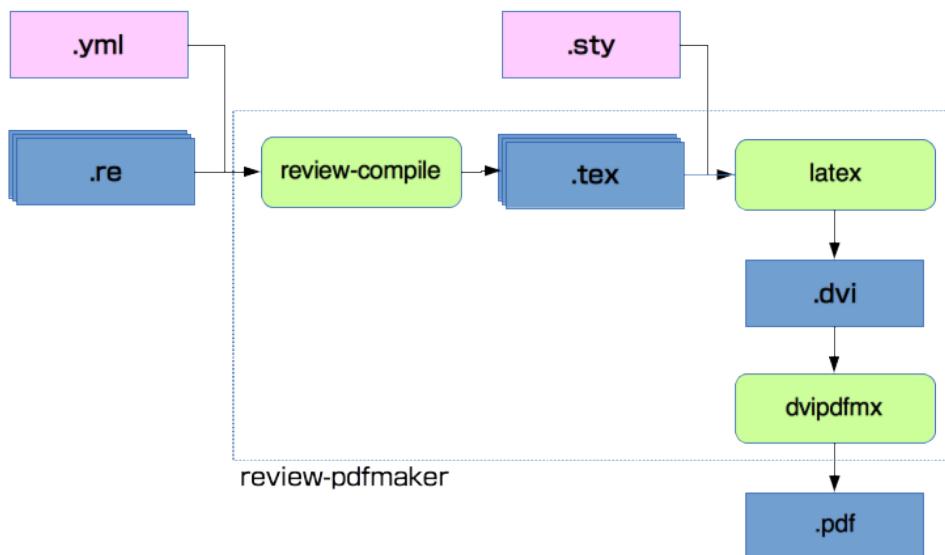


図 5.1 ReVIEW 形式のファイルを PDF に変換

.CSS

.css は、HTML を出力する際に適用するスタイルを記述するファイルで、**CSS** (Cascading Style Sheet) 形式で記述します。

また、review-epubmaker コマンドで出力する EPUB 形式のスタイルにも適用されます。

リスト 5.3: CSS のサンプル

```

@charset "utf-8";
/* Table */
*/
table {
    margin: 0 auto 2em auto;
    border-collapse: collapse;
    padding: 0;
}
table tr th {
    background-color: #eee;
    border: 1px #aaa solid;
}

```

```
    font-size: 0.75em;
    font-weight: normal;
}
table tr td {
    padding: 0.3em;
    border: 1px #aaa solid;
    font-size: 0.75em;
}

p.tablecaption, table caption, div.table p.caption {
    margin: 0;
    text-align: center;
    color: #666;
    font-size: 0.75em;
    font-weight: bold;
    text-indent: 0;
}
/* Quote */
blockquote {
    margin: 2em 0 2em 0;
    padding: 0.3em 1em;
    border: 1px #aaa solid;
}
blockquote p {
    text-indent: 0em;
}

/* Code Block */
div.code, div.caption-code, div.source-code, div.emlist-code, div.emlistnum-code {
    margin: 1em 0 2em 2em;
    padding: 0;
}
div.emlist-code p.caption {
    border-left: 7px solid #06CD23;
}
pre.emlist, pre.source, pre.list {
    margin: 0;
    padding: 5px;
    border: 1px #aaa solid;
    background-color: #eee;
}
div p.caption {
    margin: 0;
    color: #666;
    font-size: 0.75em;
```

```
    font-weight: bold;  
}
```

生成する HTML や EPUB に適用する CSS は、`review-compile` コマンドの`--stylesheet` オプションにファイル名を指定します。

また、ファイルが複数ある場合は、コンマ区切りで区切って列挙します。

```
review-compile --target=html --stylesheet=main.css style_customize.re
```

または、.yml ファイルの `stylesheet:` の項目に適用したい.css ファイルを指定することもできます。こちらもファイルが複数ある場合は、コンマ区切りで区切って列挙します。

`stylesheet` を指定した.yml ファイルは、ReVIEW のコンパイル時に`--yaml` オプションで指定します。

```
review-compile --target=html --yaml=config.yml style_customize.re
```

指定した.css は、HTML の場合、外部のスタイルシートとして参照されます。EPUB の場合は出力されるファイルの内部に取り込まれ、そのまま EPUB の見た目として適用されます。

スタイルファイルまとめ

ここまで、ReVIEW の3つのスタイルファイル (.yml/.sty/.css) を紹介しました。

.yml

ReVIEW ファイルを変換する際に必要となる設定を記述します。内容は、ReVIEW が 出力する全てのファイルに影響します。

.sty

ReVIEW で LaTeX (PDF) を出力する際の「組版に関する設定」を記述します。

.css

ReVIEW で HTML や EPUB 形式で出力する際の見た目を記述します。

次に、これら 3 つのファイルを編集して、ReVIEW が output するファイルの見た目を、実際にカスタマイズする方法を見ていきましょう。

5.2 ページサイズを変更する

PDF 形式で出力するページサイズを指定するには、.yml ファイルの `texdocumentclass:` 項目の 2 番目の値を設定します（リスト 5.4）。

この設定は、LaTeX におけるドキュメントクラスのオプションに該当します。複数のオプションがある場合は、カンマで区切って列挙します。

リスト 5.4: ページサイズを `a4j` に指定している

```
texdocumentclass: ["jsbook", "a4j"]
```

表 5.1 設定できるページサイズ

値	用紙サイズ
<code>a4paper</code>	A4
<code>b5paper</code>	B5
<code>a4j</code>	A4
<code>b5j</code>	B5

表 5.1 は、`jsbook` ドキュメントクラスに設定できる代表的な用紙サイズです。指定するドキュメントクラスによっては、設定できる値が異なる場合があるので注意してください。

A4 や B5 を指す値が重複しているのは、ページサイズに余白の設定を含んでいるためです。

たとえば、`a4paper` は `a4j` に比べ、`b5paper` は `b5j` に比べ余白が大きく、本文領域が狭くなっています。

5.3 目次に表示する項目をカスタマイズする

目次として抽出する章や節の深さを変更するには、.yml ファイルの `toclevel:` 項目を設定します（リスト 5.5）。

リスト 5.5: 抽出レベルを設定

```
toclevel: 2
```

抽出レベルを変更すると、値に応じた深さの見出しを出力します（図5.2および図5.3）。

第5章	ReVIEW スタイル解説	58
5.1	スタイルファイルの種類	58
5.2	ページサイズを変更する	64
5.3	目次に表示する項目をカスタマイズする	65
5.4	余白を調節する	66
5.5	章のページ起こしを指定する	66
5.6	ヘッダとフッタをカスタマイズする	68
5.7	既存のスタイルを変更する	69
5.8	本の構成を変更する	71
5.9	まとめ	74

図5.2 toclevel:2 を指定した場合の目次

第5章	ReVIEW スタイル解説	58
5.1	スタイルファイルの種類	58
	.yml	58
	.sty	60
	.css	62
	スタイルファイルまとめ	64
5.2	ページサイズを変更する	64
5.3	目次に表示する項目をカスタマイズする	65
5.4	余白を調節する	66
5.5	章のページ起こしを指定する	66
5.6	ヘッダとフッタをカスタマイズする	68
5.7	既存のスタイルを変更する	69
5.8	本の構成を変更する	71
	構成の変更事例 - 通しノンブル	72
5.9	まとめ	74

図5.3 toclevel:3 を指定した場合の目次

5.4 余白を調節する

PDFで出力するページの余白を指定するには、.styファイルに\geometryを設定します（リスト5.6）。

リスト5.6: 余白を設定

```
\geometry{top=18mm, bottom=23mm, left=24mm, right=24mm}
```

指定できる単位は、cm, mm の他にも LaTeXでサポートされている in, pt, em/ex, zw/zh, Qなどがあります。

5.5 章のページ起こしを指定する

PDFで出力する章（chapter）ごとのページ起こしを指定できます。

文書内で新しい章が始まるときに、見開きの左右どちらのページから始めるかを「ページ起こし」といいます。

PDF形式で出力するページ起こしを指定するには、.ymlファイルのtextdocumentclass:項目の2番目の値を設定します（リスト5.4）。

この設定は、LaTeXにおけるドキュメントクラスのオプションに該当します。複数のオプションがある場合は、カンマで区切って列挙します。

リスト5.7: 見開き、章は左右どちらのページでも開始するように設定している

```
texdocumentclass: ["jsbook", "a4j,twoside,openany"]
```

表5.2 jsbookで設定できるオプション

値	設定
openright	必ず右側ページから開始する（デフォルト）
openany	章は前章の次のページから開始する

リスト 5.7 では、まず、`twoside` オプションを指定することで、「見開きの文書」であることを設定しています。

次に、`openany` オプションを指定して、章は左右どちらのページでも開始するように設定しています。

`jsbook` ドキュメントクラスは、このオプションを指定しない場合、標準で `oneside` を適用します。その場合、`openright` などの指定が無効になる他、左右のページ位置に応じたスタイルの設定は無視されるので、注意が必要です。

オプションとして指定できる値は、使用するドキュメントクラスに応じて設定できる値が異なるので注意してください。

たとえば、ドキュメントクラス `jsbook` には、`openleft` のオプションはありませんが、`openleft` を指定できるドキュメントクラスも存在します。

■コラム：「ページ起こし」とは

前述の通り、「ページ起こし」とは、文書内で新しい章が始まるときに、見開きの左右どちらのページから始めるかを定めるものです。

章の区切りで、必ず番号の大きいページから始まる場合を「片起こし」と呼びます。`openright` を指定した場合が片起こしとなり、章は必ず、見開きの番号の大きい方のページから始まります（図 5.4）。

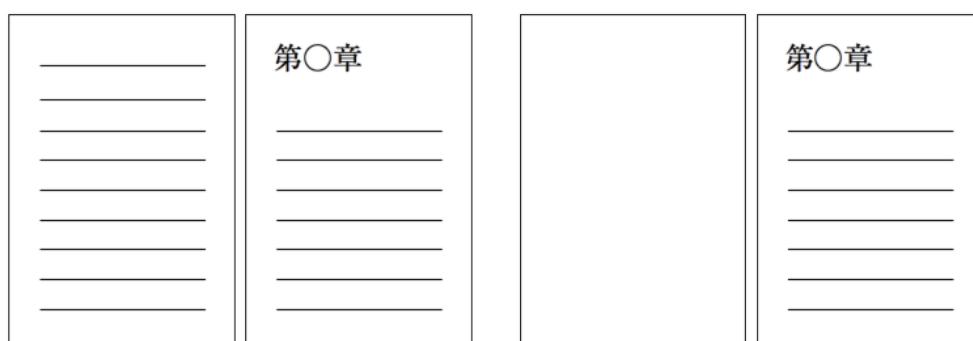


図 5.4 章は必ず見開きの右側のページから始まる

一方、片起こしの反対が「両起こし」と呼ばれます。両起こしは必ず、見開きの番号の小さい方のページから始まります。

両起こしに該当する指定はありません。`openany` を指定した場合、新しい章は、前の章

が終わった次のページから始まります。この場合は、両起こしになる場合があります（図5.5）。



図 5.5 `\openany` の場合、章はどちらのページからも始まる

また、章が左右どちらから始まっているかで、それぞれ「左起こし」「右起こし」と呼ぶこともあります。

5.6 ヘッダとフッタをカスタマイズする

PDF 形式で出力するヘッダとフッタの表示をカスタマイズできます。

表示する位置や内容は、`.sty` ファイルの`\fancyhead` と`\fancyfoot` で設定します。

リスト 5.8: ヘッダとフッタの設定

```
\fancyhead{}  
\fancyhead[LE]{\gtfamily\sffamily\bfseries\upshape \leftmark}  
\fancyhead[RO]{\gtfamily\sffamily\bfseries\upshape \rightmark}  
\cfoot{\thepage}
```

リスト 5.8 では、最初に`\fancyhead` を初期化した後、実際に表示するヘッダの内容を設定しています。

具体的には、ページ番号が偶数かつ見開きの左側ページの場合、ヘッダの左側に`\leftmark`

を表示（図 5.6）。ページ番号が奇数かつ見開きの右側ページの場合、ヘッダの左側に \rightmark を表示するように設定しています（図 5.7）。

第3章 ReVIEW スタイル解説

図 5.6 左ページに表示するヘッダ

3.2 スタイルのカスタマイズ

図 5.7 右ページに表示するヘッダ

なお、\leftmark と \rightmark は、LaTeX のコマンドとして標準で定義されています。

一般的には \leftmark が「章の名前（ReVIEW では = に続く名前）」、\rightmark が「節名（ReVIEW では == に続く名前）」にそれぞれ該当します。

のことから解るとおり、これらのコマンドは、ページ番号の偶数奇数や見開きのページ位置（左右）の組み合わせを、シンボルを指定することで個別に設定できます^{*3}。

たとえば、\fancyhead[LE] は、「偶数ページかつ見開き左側ページのヘッダ」という意味になります。

それぞれのシンボルの意味については表 5.3 を参照してください。

また、ページスタイルに応じて個別に設定を適用することもできます。リスト 5.9 は、ページスタイルが plainhead の場合のヘッダとフッタの内容を設定しています。

リスト 5.9: ヘッダとフッタの設定

```
\fancypagestyle{plainhead}{  
    \fancyhead{}  
    \fancyfoot{}  
    \fancyfoot[CE,CO]{\thepage}  
    \renewcommand{\headrulewidth}{0pt}  
    \renewcommand{\footrulewidth}{0pt}  
}
```

^{*3} ページ番号の奇数・偶数と、見開きのページ位置がそれぞれ独立しているのは、書籍によっては綴じ方向が異なる場合があるためです。

表 5.3 シンボルと意味

シンボル	意味
E	偶数ページ
O	奇数ページ
L	左側
C	中央
R	右側
H	ヘッダ
F	フッタ

5.7 既存のスタイルを変更する

PDF で出力する本文やリスト、図表に適用するスタイルを個別にカスタマイズすることができます。

前述の通り ReVIEW は、ReVIEW 形式のファイルを PDF へ変換する前に、LaTeX のソースファイルを出力します。

例えば、ReVIEW ファイルリスト 5.10 は、LaTeX ソースファイルでは、リスト 5.11 になります。

リスト 5.10: ReVIEW 記法

```
//list[change_pagesize][ページサイズを a4j に指定している]{  
texdocumentclass: ["jsbook", "a4j"]  
//}
```

リスト 5.11: LaTeX ソースファイル

```
\reviewlistcaption{リスト 3.4: ページサイズを a4j に指定している}  
\begin{reviewlist}  
texdocumentclass: ["jsbook", "a4j"]  
\end{reviewlist}
```

\begin{reviewlist}と\end{reviewlist}で囲まれた範囲を Environment（環境）と言
い、環境の中にあらかじめ設定したスタイルを適用できます。

そして、.sty ファイルで環境を再定義することで、これらのスタイルを変更できます。

リスト 5.12 は、renewenvironment によって環境 reviewlist を再定義し、環境中の文
字の大きさを小さめ (\footnotesize) に設定しています。

リスト 5.12: 環境の再定義

```
\renewenvironment{reviewlist}{%
  \begin{shaded}\footnotesize\setlength{\baselineskip}{1.2zw}\begin{alltt}}{%
  \end{alltt}\end{shaded}\par\vspace*{0.5zw}}
```

また、環境だけでなく、コマンドを再定義することもできます。

リスト 5.13 は、renewcommand によってコマンド reviewlistcaption を再定義し、文字
の大きさを小さめ (\footnotesize) に設定しています。

リスト 5.13: コマンドのオーバーライド

```
\renewcommand{\reviewlistcaption}[1]{
  \medskip\noindent{\footnotesize #1}\vspace*{-1.3zw}
}
```

再定義したい環境名やコマンドを探すには、LaTeX のソースファイルを見るのが良いで
しょう。

変換後の LaTeX のソースファイルを確認するには、review-compile するか、
review-pdfmaker を実行するときに指定する.yml ファイルでデバッグ機能を有効に設定
します（リスト 5.14）。

リスト 5.14: デバッグの有効化

```
debug: true
```

\renewenvironment や \renewcommand は、既存の定義を丸ごと変更してしまいます。

したがって、現在適用しているスタイルの一部だけ変更したい場合は、その環境やコマンドの現在の定義を確認してから、一部を変更する作業が必要になります。

環境やコマンドの定義を調べるには、ReVIEW 関係であれば <https://github.com/kmuto/review> の review/lib/review/review.tex.erb にまとめられています^{*4}。

ここに記載のない環境やコマンドは、LaTeX 及び各パッケージで定義されている可能性があります。まずは、どのパッケージで定義されているかを確認するところから始める必要があります。

5.8 本の構成を変更する

印刷所へ入稿する原稿を制作していると、ReVIEW が標準で用意している構成そのものを変更する必要に迫られる時があります。

その場合、.yml や.css を配置しているディレクトリの下に layouts/layout.tex.erb を置くと、ReVIEW が出力する LaTeX ソースファイルの構成を変更できます。

ReVIEW は、通常は <https://github.com/kmuto/review> の review/lib/review/review.tex.erb を、LaTeX のソースファイルのテンプレートとして読み込みますが、layouts/layout.tex.erb がある場合、そちらを優先して適用します。

カスタマイズに当たっては、review/lib/review/review.tex.erb を layouts/layout.tex.erb にコピーして変更すると良いでしょう。

^{*4} 2013年11月時点。最新の情報については GitHub の ReVIEW リポジトリを参照してください。

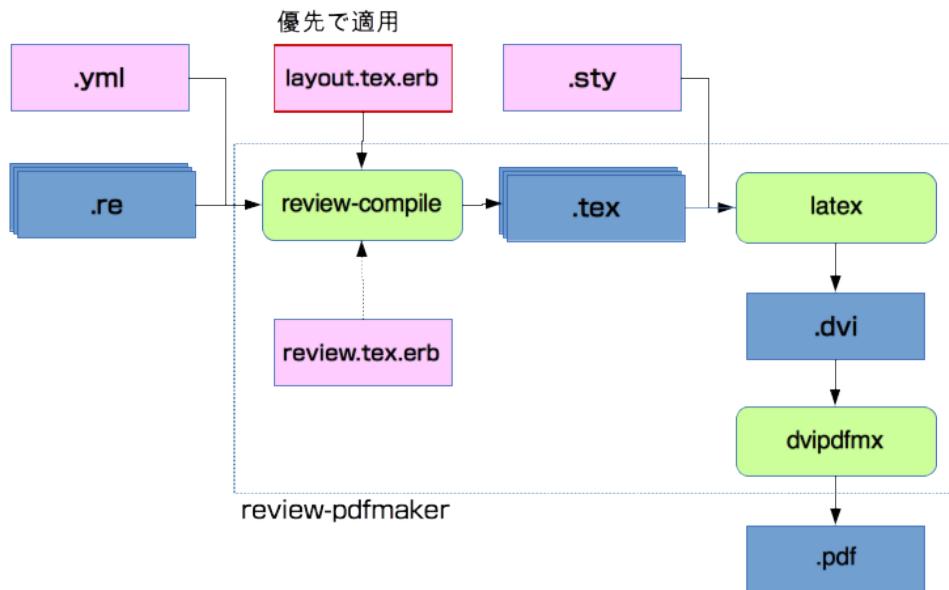


図 5.8 layout.tex.erb の取り扱い

構成の変更事例 - 通しノンブル

たとえば筆者の場合、入稿当日に印刷所より、全てのページに「通しノンブルが必要」との指摘を受け、慌てて構成を変えると言うことがありました。

「通しノンブル」とは、印刷する全てのページに 1 から始まるノンブル（ページ番号）を出力することを言います。印刷、製本の過程でページの抜けや入れ替わりがないことを確認するために必要となり、後々のトラブルを防止するために入稿規程に含めている印刷所も少なくありません⁵。

ReVIEW の標準構成は、前書き (preface) や目次についてはローマ数字 (i, ii, iii...) でのノンブル。本文についてはアラビア数字 (1, 2, 3...) でノンブルを出力します。また、前書きと本文でノンブルがリセットされるので、本文は必ず 1 から始まってしまいます。

さらに、扉や奥付にはノンブルそのものが表示されません。扉や奥付を含む全てのページ

⁵ 製本後に読者の目に触れない場所（綴じ目の奥など）にノンブルを表示する「隠しノンブル」を認めている印刷所もあります

に、通しノンブルが表示されている必要があるとの指摘でした。

この問題に対応するためには、ReVIEW が出力する構成そのものを変更する必要があります。

リスト 5.15 が、実際に `layout.tex.erb` を変更した内容です。

`\frontmatter` の位置をタイトルページの前に移動し、ノンブル表記を `arabic` に指定しています。

次に、`\thispagestyle` でページスタイルをノンブルを表示しない(`empty`)から `plainhead` に変更。同様の変更は奥付にも適用します。

ここで指定しているページスタイル `plainhead` は、`.sty` 側で定義しています（リスト 5.16）。

ヘッダとフッタを全て消去して、フッタの中央にノンブルのみ表示。ヘッダとフッタの横線も `0pt` として非表示にしています。

`\setcounter{page}` でページ番号を 1 と設定しているのは、このままでは表紙もページ数に含まれ、扉が 2 ページ目と表記されてしまうためです。表紙をページ数に含めないため、今回このように指定しました。

前書きや目次から本文に切り替わるタイミングでページ数を引き継いでいるのが `\mainmatter` の次の行にある `\continuenumber` です。

`\continuenumber` は、`\begingroup` 内で定義しているマクロで、直前のページ数を引き継いで `\setcounter{page}` で設定しています。

こうすることで、本来はリセットされるページカウンタを再設定しています。

リスト 5.15: 通しノンブルにする変更

```
diff --git a/article/layouts/layout.tex.erb b/article/layouts/layout.tex.erb
index 4a83d4c..b9e72bd 100644
--- a/article/layouts/layout.tex.erb
+++ b/article/layouts/layout.tex.erb
@@ -176,6 +176,9 @@
 
 \reviewmainfont
+
+\frontmatter
+\pagenumbering{arabic}
+
 <% if values["titlepage"] %>
 <% if custom_titlepage %>
```

```
<%= custom_titlepage %>
@@ -188,7 +191,8 @@
 \end{center}
 \clearpage
 <% end %>
-\thispagestyle{empty}
+\thispagestyle{plainhead}
+\setcounter{page}{1}
 \begin{center}%
 \mbox{} \vskip5zw
 \reviewtitlefont%
@@ -207,9 +211,6 @@
 <% end %>
 <% end %>

-\renewcommand{\chaptermark}[1]{{}}
-\frontmatter
-
%% preface
<%= values["pre_str"] %>

@@ -218,8 +219,18 @@
 \tableofcontents
 <% end %>

-\renewcommand{\chaptermark}[1]{\markboth{\prechaptername\thechapter\postchapter}
+\begingroup
+ \cleardoublepage
+ \edef\continuenum{\endgroup
+   \noexpand\mainmatter
+   \setcounter{page}{\the\value{page}}%
+ }
+
\mainmatter
+\continuenum
+
+\renewcommand{\chaptermark}[1]{\markboth{\prechaptername\thechapter\postchapter}
+
<%= values["chap_str"] %>
\renewcommand{\chaptermark}[1]{\markboth{\appendixname\thechapter^#1}{}}%
\reviewappendix
@@ -228,7 +239,7 @@
 <% if values["colophon"] %>
 %% okuduke
 \reviewcolophon
```

```
- \thispagestyle{empty}
+ \thispagestyle{plainhead}

\vspace*{\fill}
```

リスト 5.16: plainhead ページスタイル

```
\fancypagestyle{plainhead}{
    \fancyhead{}
    \fancyfoot{}
    \fancyfoot[CE, CO]{\thepage}
    \renewcommand{\headrulewidth}{0pt}
    \renewcommand{\footrulewidth}{0pt}
}
```

5.9 まとめ

本章では、ReVIEW が生成する各ファイルの基本的な設定や、見た目（スタイル）に関する変更について解説しました。

スタイルの変更の事例で、LaTeX に関する記述が多かったのは、筆者が主に PDF 出力を目的として ReVIEW を利用していることもあります、カスタマイズの自由度が最も高く、また、カスタマイズが困難であるのも、LaTeX であるからです。

LaTeX のカスタマイズについては、奥村晴彦氏の「LATEX2e 美文書作成入門」^{*6}が参考になりました。

各出力形式における ReVIEW の役割を理解した上で、それぞれのスタイルファイルを正しくカスタマイズすることを心がけてください。

^{*6} 「LATEX2e 美文書作成入門」 <http://www.amazon.co.jp/dp/4774160458> - 奥村晴彦著 技術評論社刊

第6章

コマンド解説

本章では ReVIEW に付属するコマンドを紹介します。

6.1 review-compile

`review-compile` は ReVIEW ファイル 1 つを変換して `--target` オプションで指定するフォーマットに変換します。

最も簡単な例として、HTML を出力する例をリスト 6.1 に示します。入力は第 1 章の冒頭で示した例と同じです。

リスト 6.1: `review-compile` で HTML を出力する例。

```
> review-compile --target=html amedama.re
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:ops="http://www.idpf.org/2007/ops" x-
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <meta http-equiv="Content-Style-Type" content="text/css" />
  <meta name="generator" content="ReVIEW" />
  <title>Google Drive API を使ってファイルをダウンロードする</title>
</head>
<body>
<h1><a id="h38"></a>第 38 章 Google Drive API を使ってファイルをダウンロードする</h1>
<p>本章では、Python スクリプトから Google Drive に保存されているファイルを取得する方法を紹介します。Android とは直接関係ありませんが、同様の API を Android 上で利用する際に参考になるかもしれません。</p>

<h2><a id="h38-1"></a>38.1 収録されている背景</h2>
<p>『Effective Android』同人誌版の執筆が行われていた頃、原稿は Google Drive 上で管理されていました。一部の技術者の希望により git も用いる運用に途中から切り替えたのですが、全員が git を利用できるわけではないため、原則 Google Drive にファイルをアップロードとし、git は希望者が選択して用いる、という体制になりました<a href="#fn-not_used" class="noteref">*1</a>。
```

```
</p>
<div class="footnote"><p class="footnote">[<a id="fn-not_used">*1</a>] 本稿執筆時点  
ではすでにこの体制は終了しています。</p></div>
<p>このとき、管理者が Google Drive 上のファイルを手動でダウンロードして git プロジェクトに取  
り込んでいると聞き、私はその作業を自動化できるかもしれませんと考えました。</p>
<p>Google Drive API とその SDK は Google から無料で提供されていました。少し調べた結果、今回  
の目標のためにはそれを使えばよいことが分かりました。</p>
<ul>
<li>What Can You Do with the Drive SDK?<br /><?dtp tab ?>
<a href="https://developers.google.com/drive/about-sdk"  
class="link">https://developers.google.com/drive/about-sdk</a>
</li>
</ul>
... (以下省略)
```

HTML 向け出力は、執筆時に「大まかにどのような出力になるのか」を確認するためによく使われます。その他にもプレーンテキスト形式にする--target=text、PDF を出力するために必要な TeX 形式のファイルを出力する--target=latex、軽量マークアップ言語である Markdown 形式を出力する--target=markdown などがあります。

読者の環境で--target で指定できるフォーマットを調べるには、ReVIEW 本体のインス
トール先にある lib/review/???builder.rb というファイル名のファイルを確認するのが
早道です。

リスト 6.2: ReVIEW が対応している出力を確認

```
> ls lib/review/*builder.rb
lib/review/builder.rb      lib/review/ewbbuilder.rb    lib/review/idgxmlbuilder.rb
lib/review/latexbuilder.rb  lib/review/textbuilder.rb  lib/review/epubbuilder.rb
lib/review/htmlbuilder.rb   lib/review/inaobuilder.rb  lib/review/markdownbuilder.r
```

もっとも、対応する出力形式の多くは ReVIEW 開発者らが仕事上利用するものであったり
するため、執筆時には--target=html を指定する機会が多いと思われます。

--check を指定すると、結果は出力せず、そのフォーマットで入力したファイルが正しく
変換出来るかを確認できます。例えば、footnote と fn で対応がとれているべき場所でとれ
ていない場合、その旨が表示されます。これもまた執筆中にエラーを前もって避けるために便

利です。¹

リスト 6.3: @<fn>と対応する//footnoteがない場合に発生するエラー

```
> review-compile --check --target=html amedama.re
amedama.re:11: error: ReVIEW::KeyError
```

出力された html を、執筆時のデバッグ用途としてではなく見栄えの良い公開用ファイルにすることもできます。--yaml=(ファイル名) で ReVIEW プロジェクトの設定を読みこませることでスタイルを変更するとよいでしょう。なお、--help を指定すると、ファイルを読み込んで変換する代わりに対応するオプションの一覧が表示されます。本章で説明されていないオプションも多々あるので、必要に応じて参照してください。

注意: review-compile ではそのまま **PDF** と **EPUB** は生成できない

現時点では review-compile からは直接 PDF と EPUB 形式のファイルを生成することはできません。² 後述する review-pdfmaker と review-epubmaker をそれぞれ利用してください。

review-compile は主に単一のファイルに対して操作を行う一方、review-pdfmaker と review-epubmaker は ReVIEW プロジェクト全体を対象とするため、用途も異なります。仮に執筆途中の ReVIEW ファイルのみ、PDF によるフォーマットを確認したい場合、自分で TeX 形式から生成する、あるいは逆に全体を review-pdfmaker で生成したあと、該当する章を pdktk 等のコマンドで切り出してください。

6.2 review-pdfmaker

review-pdfmaker はそのプロジェクトの PDF を生成します。引数として YAML ファイル (config.yml) を一つ指定します。

リスト 6.4: review-pdfmaker の実行例

¹ review-pdfmaker にはこの機能がない一方、前述した通り review-compile では直接は PDF を出力しません。対策として review-compile --check --target=latex とすると、PDF で問題が起こる前に修正できます。出来れば review-pdfmaker を使って対象とする全てのファイルに対してチェックを行える何かが欲しいところです。

² --target=epub は圧縮して EPUB 形式とする前のただの HTML を出力します。

```
> review-pdfmaker config.yml  
(出力省略)
```

YAML ファイルには本のタイトルや筆者名といった本のメタデータとなる設定を記述しておきます。その設定の一つ「bookname」が出力される PDF に対応しています。

bookname で「book」となっている場合、ReVIEW は「book-pdf/」ディレクトリに、途中過程で生成されるファイルを保存した上で、最終的に「book.pdf」を自身のディレクトリに保存します。仮にこれらのファイル・ディレクトリがすでに存在している場合、ReVIEW はコマンドの実行を中止します。

リスト 6.5: すでに PDF を一度作成したあとに再度 review-pdfmaker したときの実行例

```
> review-pdfmaker config.yml  
/opt/review/bin/review-pdfmaker:57:in `mkdir': File exists - ./book-pdf (Errno::EXIS  
    from /opt/review/bin/review-pdfmaker:57:in `main'  
    from /opt/review/bin/review-pdfmaker:219:in `<main>'
```

再度コンパイルする際には、まず手動で PDF (book.pdf) と中間ファイルを収めるディレクトリ (book-pdf/) を削除してから、再度 review-pdfmaker を実行します。

review-pdfmaker は内部で review-compile --target=latex を行ったあとに plateax や dvipdfmx のような TeX 形式のファイルから PDF へ変換する ReVIEW 外部のコマンドを実行します。ReVIEW の外部のコマンドを実行している際には外部のログがそのまま標準出力や標準エラー出力に反映されるため、面食らうかもしれません。

脱線: **Rakefile** で自動化

コンパイル前に book.pdf 等を手動で削除する作業は少々面倒なため、Rakefile^{*3} に詳しい人はしばしば以下のような設定を書いています。

リスト 6.6: Rakefile に book.pdf と book-pdf の削除を行わせてしまう一例

*3 今回のように典型的な作業を自動化するツールの一つです。後述する review-init でも生成されています。

```
task :default => :pdf

desc 'generate PDF file'
task :pdf => "book.pdf"

SRC = FileList['*.re'] + ["config.yml"]
file "book.pdf" => SRC do
  sh "rm -f book.pdf"
  sh "rm -rf book book-pdf"
  sh "review-pdfmaker config.yml"
end
```

このように記述しておくと、rake コマンドは関連する ReVIEW ファイルや config.yml に変更があった際に限って削除と review-pdfmaker の実行を行います。

執筆時の注意点

review-pdfmaker は、その書籍に含まれる ReVIEW ファイルに問題があっても、一見正常に PDF を出力することができます。この場合、コマンド自身は「成功」と報告するのですが、該当する章のデータが勝手に抜け落ちている、という事態につながります。

最も単純な事例は第3章で紹介した CHAPS での指定を忘れることがあります。また、一部の構文エラーについても、そのファイルがなかったこととして、PDF が生成されてしまいます。

これは、複数の ReVIEW ファイルを用いて原稿執筆している際に若干厄介な問題です。特に執筆後半では、見栄えの調整のために review-compile よりも頻繁に review-pdfmaker 等を実行することがあるのですが、review-compile エラーで中止する代わりに、コマンドはエラーが発生した章を無視して PDF を生成します。

対策として、執筆時には review-compile --target=latex --check を実行して執筆中の原稿単体で発生している問題を排除してから review-pdfmaker を実行することをおすすめします。

章がまるまる抜け落ちることを利用してページ数が減っていないか確認することも出来ます。参考まで、Linux の pdftk コマンドでページ数を見る例を示します。

```
> pdftk book.pdf dump_data_utf8 | grep NumberOfPages
```

```
NumberOfPages: 103
```

6.3 review-epubmaker

review-pdfmaker 同様、review-epubmaker はプロジェクトのメタデータとなる YAML ファイルを引数として EPUB ファイルを生成します。

リスト 6.7: review-epubmaker の例

```
> review-epubmaker config.yml  
(出力省略)  
> file book.epub  
book.epub: EPUB ebook data  
(電子書籍リーダに読み込ませることで内容を確認出来る)
```

review-pdfmaker 同様、参照する YAML ファイル内の bookname を元にして出力されるファイル名が決定されます。

また、コンパイルに関わる注意点も review-pdfmaker と同様です。PDF とは異なり EPUB でページ数をチェックすることはできませんが、EPUB は実質 zip 圧縮された HTML ファイルの塊であるという点を利用して、unzip -l 等で自身の原稿ファイルが最終的な EPUB ファイルに含まれているかをチェックすることは出来るでしょう。

6.4 review-preproc

引数に取ったファイル内にある、#@mapfile、#@maprange、#@mapoutput といったタグを認識し、それぞれのタグの要求した出力を元のファイルに埋め込みます。

例として、プログラミング言語 Java のバージョン情報を埋め込む例を見てみます。

リスト 6.8: #@mapoutput を用いた実例

```
//cmd{  
#@mapoutput(java -version 2>&1)  
#@end  
//}
```

この命令だけを含むファイル（sample.reとします）に対して review-preproc を実行すると、例えば以下のとおりになります。

```
> review-preproc sample.re
//cmd{
# @mapoutput(java -version 2>&1)
java version "1.6.0_27"
OpenJDK Runtime Environment (IcedTea6 1.12.6) (6b27-1.12.6-1~deb7u1)
OpenJDK 64-Bit Server VM (build 20.0-b12, mixed mode)
#@end
//}
```

コマンドの実行結果が review-preproc で挿入されています。なお、-r オプションを用いた場合、上述した例のように標準出力に表示される代わりに元のファイルの該当部分を上書きします。

```
> cat sample.re
//cmd{
# @mapoutput(java -version 2>&1)
#@end
//}
> review-preproc -r sample.re
> cat sample.re
//cmd{
# @mapoutput(java -version 2>&1)
java version "1.6.0_27"
OpenJDK Runtime Environment (IcedTea6 1.12.6) (6b27-1.12.6-1~deb7u1)
OpenJDK 64-Bit Server VM (build 20.0-b12, mixed mode)
#@end
//}
```

#@mapfile、#@maprange、#@mapoutput はいずれも、ReVIEW ファイルの外部を参照するための命令です。いずれもその命令開始とセットで#@end を別の行に記述する必要があります。

プログラミング言語の解説等を行なっている場合、サンプルとして提供するソースコードの

表 6.1 生成されるファイルと役割

コマンド名と引数 意味
mapoutput(コマンド) コマンドを実行して、その出力結果を展開する。
mapfile(ファイル名) ファイルの内容全てをその場に展開する。
maprange(ファイル名, 範囲名) ファイル内の範囲をその場に展開する。

一部を本文に挿入し、サンプルのソースコードの修正と併せて本文を動的に変更したいといったことがあります。review-preproc はそういったときに力を発揮するコマンドです。

サンプルと併せてコマンドラインツール自体もアップデートしていく可能性があります。例えば Java のバージョンが手元でアップデートされたとします。通常、手動で埋め込むべきところを、以下のようにアップデートすることができます。

```
> cat sample.re
//cmd{
#@mapoutput(java -version 2>&1)
java version "1.6.0_27"
OpenJDK Runtime Environment (IcedTea6 1.12.6) (6b27-1.12.6-1~deb7u1)
OpenJDK 64-Bit Server VM (build 20.0-b12, mixed mode)
#@end
//}
```

(ここで Java のバージョンをアップデートした)

```
> review-preproc -r sample.re
> cat sample.re
//cmd{
#@mapoutput(java -version 2>&1)
java version "1.7.0_25"
Java(TM) SE Runtime Environment (build 1.7.0_25-b15)
Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode)
#@end
//}
```

mapfile はほぼ自明なので省略します。

maprange は名前の通りファイルの範囲を指定出来ます。ただし、使い方が特殊です。例えば、以下のようなファイルがあるとします

リスト 6.9: 例となるファイル

```
#!/usr/bin/env python
import os
import sys

if __name__ == "__main__":
    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "project.settings")
    from django.core.management import execute_from_command_line
    execute_from_command_line(sys.argv)
```

ここで if から始まる行とそれ以降 3 行を引用したいとします。その場合、`maprange` はこのファイルに対してマーカをセットすることを要求します。

リスト 6.10: `maprange` を使うためにマーカをセット。ファイル名は `test.py`

```
#!/usr/bin/env python
import os
import sys

#@orange_begin(example)
if __name__ == "__main__":
    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "project.settings")
    from django.core.management import execute_from_command_line
    execute_from_command_line(sys.argv)
#@orange_end(example)
```

この状態で、`ReVIEW` のファイルに以下を記述します。

リスト 6.11: マーカを `ReVIEW` のファイルで使う

```
#@maprange(filename.py,example)
#@end
```

そこで `review-preproc` を用いると、マーカ範囲の内容がコピーされます

リスト 6.12: 内容がコピーされる

```
#@maprange(filename.py,example)
if __name__ == "__main__":
```

```
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "project.settings")
from django.core.management import execute_from_command_line
execute_from_command_line(sys.argv)
#@end
```

引用元のファイルを変更するため、状況によっては非常に使いづらい命令だと筆者は考えます。

特にプログラミング言語の著書を書くときには問題になるかもしれません。現在の実装では、この行の先頭は必ず#です。ReVIEW の本体が採用している Ruby 言語ならまだしも、例えば C・C++ といった言語では#は全く別の意味を持ちますから、おそらく引用元のファイルはコンパイルエラーを発生させるでしょう。本末転倒です。

6.5 review-init

`review-init` は、引数に指定された名前のディレクトリを作成し、その配下に新しい ReVIEW プロジェクトを作成します。作成されるディレクトリが存在していた場合エラーになります。このコマンドは新規のプロジェクト作成時に利用できます。^{*4}

```
/tmp> ls new-project
ls: cannot access new-project: No such file or directory
/tmp> review-init new-project
/tmp> ls new-project
CHAPS  POSTDEF PREDEF Rakefile config.yml images layouts new-project.re sty
> review-init new-project
/tmp/new-project already exists.
```

`review-init` で生成されるファイルは表 6.2 の通りです。

原稿サンプルのファイル名のみ `review-init` の引数名に依存します。また、CHAPS には（プロジェクト名）.re が含まれます。このファイルを消す・その名前を変更する場合は CHAPS も同様に修正してください。

^{*4} RubyGems から取得できるバージョン 1.0 でこのコマンドはそもそも存在せず、1.1 では `review-init` が動作しないことが確認されています。git 版の執筆時点での最新版を用いて検証しています

表 6.2 生成されるファイルと役割

ファイル名 役割
CHAPS 本文構成する ReVIEW フォーマットのファイル (.re) を記述する。
PREDEF 本の前付けとなる ReVIEW フォーマットのファイル (.re) を記述する。実例の章参照
POSTDEF 本の後付けとなる ReVIEW フォーマットのファイル (.re) を記述する。実例の章参照
Rakefile コマンド実行の自動化を図るためのファイル。第 11 章の Rakefile の項を参照。
config.yml このプロジェクトのメタデータ。第 5 章を参照。
images/ 本の画像を収めるディレクトリ
layouts/ <chap>{style_customize} を参照。
sty/ <chap>{style_customize} を参照。
(プロジェクト名) .re サンプルの ReVIEW ファイル

review-init を用いない初期化方法

執筆開始時にプロジェクトを作成する方法は何通りかあります。

- review-init を用いる。
- <https://github.com/takahashim/review-sample-book> を用いる。
- 手動で各ファイルを用意する。
- 自分用にカスタマイズ済の既存のプロジェクトから適宜コピーする。

現時点では ReVIEW を構成するファイルは数種類です。それらが複雑に絡み合った依存関係を形成しているわけではありません。手動でプロジェクトを構成しても、特別不都合が発生するわけでもありません。実際、第 3 章「執筆を始める」では ReVIEW の構造を順番に説明出来るように手動の方法を採用しています。

個人個人で最適な方法を用いれば良いのではないかと筆者は考えます。

6.6 その他のコマンド

review-epubmaker-ng

review-epubmaker と同様に EPUB 形式のファイルを生成するコマンドです。武藤氏による新しい実装が試みられている様子がありますが、今後旧実装と統合されてなくなる可能性も

あります。執筆をしたい人が敢えて利用する理由はありません。

review-vol

`review-vol` は ReVIEW で書かれた原稿のファイルサイズや文字数や行数を一覧にして取得します。引数は受け取りません。

リスト 6.13: `review-vol` の出力例。原稿は Effective Android の C84 版を使用

```
(原稿のあるディレクトリで)> review-vol
      1KB    332C    12L    1P preface ..... はじめに
      1KB    403C    15L    1P techbooster_member_introduction サ 一 ク
ル Techbooster 紹介
1.   0KB     0C     0L    0P blank .....
2.   9KB    7240C   288L   9P yanzm ..... はやりのカード UI をちゃちゃ
っと作る
3.  12KB   7345C   270L   12P kassy_kz ..... 引っ張って更新する ListView
を作る (PullToRefresh)
4.  10KB   3655C   126L   10P tommmy ..... デザイナーもエンジニアも幸せ
になる Android アプリデザインの方法
5.  13KB   6121C   170L   13P kazzz ..... Twitter Bootstrap でレスポンシブ Web デザインを試す
6.  13KB   6054C   166L   13P keiji_ariyama ..... AndroidManifest マニアック
ス
7.  17KB   9156C   224L   17P vvakame ..... Eclipse の設定ファイルを科学
する
8.  22KB  12777C   309L   22P zaki50 ..... apt 対応プロジェクトの作り方
9.  15KB   7530C   280L   15P esmasui ..... ContentProvider をアトミック
操作する
10. 15KB   6589C   199L   15P yokmama ..... SQLiteDatabase の悪(フル)を
使う
11.  7KB   3138C   100L    7P gabu ..... 5 分でできるプッシュ通知
12. 14KB   8347C   354L   14P kei-i-t ..... GCM を使用して Android-PHP で
PUSH 通知を実装する
13. 15KB   9092C   330L   15P nkzn ..... 戦いはここにある！ Maps v2 で
戦場を描け
14. 19KB   8614C   285L   19P namasode ..... vCard をちょこっと
15.  5KB   2863C    91L    5P androidsola ..... 残念な NFC タグに反応する
Android を作る
16. 17KB  11992C   376L   17P lychee ..... ヘビーユーザに喜ばれそうなち
よつとした対応
17. 14KB   7185C   235L   14P sys1yagi ..... SharedPreferences に Java オ
ブジェクトを
18. 17KB   7067C   151L   17P muo_jp ..... オーディオ出力のパフォーマン
```

スを改善するための足がかり

19. 17KB 7594C 208L 17P eaglesakura Android の TextureView と SurfaceView の使い分け
20. 11KB 6007C 198L 11P kobashin NavigationDrawer を使ってスライド式のメニューを実装する
21. 9KB 4161C 159L 9P muchiki0226 Fragment で実装するアニメーション付きカード UI
22. 19KB 12624C 458L 19P amedama Google Drive API を使ってファイルを git プロジェクトに取り込む
23. 3KB 1033C 63L 3P contributer コントリビュータ紹介
=====
281KB 156919C 5067L 281P

-P オプションをつけると PART に記述されたパートと PREDEF、POSTDEF 毎に結果を分解します。

リスト 6.14: review-vol -P の出力例 (Effective Android の C84 版原稿を使用)

```
> review-vol -P
    1KB    332C    12L    1P preface ..... はじめに
    1KB    403C    15L    1P techbooster_member_introduction サ ー ク
ル Techbooster 紹介
-----
    2KB    735C    27L    2P

Part 1
 1.  0KB      0C      0L    0P blank .....
 2.  9KB    7240C   288L    9P yanzm ..... はやりのカード UI をちゃちゃ
っと作る
 3. 12KB    7345C   270L   12P kassy_kz ..... 引っ張って更新する ListView
を作る (PullToRefresh)
 4. 10KB    3655C   126L   10P tommmy ..... デザイナーもエンジニアも幸せ
になる Android アプリデザインの方法
 5. 13KB    6121C   170L   13P kazzz ..... Twitter Bootstrap でレスポンシブ Web デザインを試す
 6. 13KB    6054C   166L   13P keiji_ariyama ..... AndroidManifest マニアック
ス
 7. 17KB    9156C   224L   17P vvakame ..... Eclipse の設定ファイルを科学
する
 8. 22KB   12777C   309L   22P zaki50 ..... apt 対応プロジェクトの作り方
 9. 15KB    7530C   280L   15P esmasui ..... ContentProvider をアトミック操作する
10. 15KB   6589C   199L   15P yokmama ..... SQLiteDatabase の悪(ワル)を
```

使う

11. 7KB 3138C 100L 7P gabu 5分でできるプッシュ通知
12. 14KB 8347C 354L 14P kei-i-t GCM を使用して Android-PHP で
PUSH 通知を実装する
13. 15KB 9092C 330L 15P nkzn 戦いはここにある！Maps v2 で
戦場を描け
14. 19KB 8614C 285L 19P namasode vCard をちょこっと
15. 5KB 2863C 91L 5P androidsola 残念な NFC タグに反応する
Android を作る
16. 17KB 11992C 376L 17P lychee ヘビーユーザに喜ばれそうなち
ょっとした対応
17. 14KB 7185C 235L 14P sys1yagi SharedPreferences に Java オ
ブジェクトを
18. 17KB 7067C 151L 17P muo_jp オーディオ出力のパフォーマン
スを改善するための足がかり
19. 17KB 7594C 208L 17P eaglesakura Android の TextureView と
SurfaceView の使い分け
20. 11KB 6007C 198L 11P kobashin NavigationDrawer を使ってス
ライド式のメニューを実装する
21. 9KB 4161C 159L 9P muchiki0226 Fragment で実装するアニメー
ション付きカード UI
22. 19KB 12624C 458L 19P amedama Google Drive API を使ってフ
ァイルを git プロジェクトに取り込む
23. 3KB 1033C 63L 3P contributer コントリビュータ紹介

280KB 156184C 5040L 280P
=====
281KB 156919C 5067L 281P

複数のパートに分かれた著書の管理には便利です。^{*5}

このコマンドが同人誌の執筆等で必要になるケースはないかと思われます。

review-index

現在の ReVIEW ディレクトリのインデックス情報を出力します。主に編集者向けのコマン
ドです。

リスト 6.15: review-index の出力例。全章の情報を出力するため -a も指定している

^{*5} 『Effective Android』商業誌版はパートを分解しているため、review-vol -P の利用価値が上がります。

```
> review-index -a
 1KB    332C   11L  はじめに (preface)
                6L    1 内容について
 1KB    403C   13L  サークル Techbooster 紹介 (techbooster_member_introduction)
                13L  1 サークル Techbooster 紹介
                3L    1 第 11 章 / @kei_i_t
                2L    2 第 19 章 / @kobashinG
                3L    3 第 20 章 / @muchiki0226
                3L    4 編集 / @mhidaka
2.     9KB    7240C  285L  はやりのカード UI をちゃちゃっと作る (yanzm)
                29L    1 カード UI に変更するアプリ
                27L    2 背景色を設定する
                48L    3 カード UI 用のレイアウトを作る
                171L   4 カード UI 用に ListView を調整する
3.    12KB    7345C  292L  引っ張って更新する ListView を作る (PullToRefresh) (kassy_kz)
                130L   1 Twitter 公式アプリ等で使われている更新方法
                5L    1 ライブドアのダウンロード
                68L   2 基本的な使い方
                37L   3 カスタマイズ
                159L   2 Gmail アプリ等で使われているパターン
                6L    1 ライブドアのダウンロード
                27L   2 基本的な使い方
                117L   3 カスタマイズ
.. (以下省略)
```

review-check, review-checkdeps, review-validate

この 3 コマンドについては、本稿執筆時点では動作確認できませんでした。興味を持たれた方はぜひ調べてみてください。

第7章

Sublime Text 2 で ReVIEW を書く

ReVIEW 記法を理解しても、エディタの補助なしで書くのはなかなか辛いものがあります。そこで、この章では Sublime Text 2 というエディタで ReVIEW を書くためのセットアップ方法を紹介します。

Sublime Text 2 はシンプルながら拡張性の高いテキストエディタです^{*1}。<http://www.sublimetext.com/2> からダウンロードすることができます。無料で試すことができ、継続して利用する場合はライセンスを購入します。OS X, Windows, Linux をサポートしていますが、Linux では日本語入力がデフォルトでできないという問題があります。

Sublime Text 2 の大きな特徴の 1 つのが、プラグインを簡単に開発できるという点です。シンタックスハイライトを正規表現で指定できまし、入力補完やビルド時に行う処理などを python で書くことができます。

そこで、ReVIEW 記法に沿ったシンタックスハイライトと入力補完のプラグインを筆者が用意しました。<https://github.com/yanzm/ReVIEW>

7.1 Sublime Text 2 用 ReVIEW プラグインをインストールする

まず、Package Control というプラグインをインストールします。すでにインストールしている場合はこのステップは必要ありません。

[View] - [Show Console] からコンソールを表示し、コンソールにリスト 7.1 のコマンドを入力します。

リスト 7.1: Package Control をインストールするためのコマンド

^{*1} Sublime Text 3 というものもあります。<http://www.sublimetext.com/3>

```
import urllib2,os;
pf='Package Control.sublime-package';
url = 'http://sublime.wbond.net/' +pf.replace(' ','','%20');
ipp = sublime.installed_packages_path();
os.makedirs( ipp ) if not os.path.exists(ipp) else None;
urllib2.install_opener( urllib2.build_opener( urllib2.ProxyHandler( )));
open( os.path.join( ipp, pf), 'wb' ).write( urllib2.urlopen( url ).read());
print( 'Please restart Sublime Text to finish installation');
```

実行したらSublime Text 2を再起動します。

次に、Command + Shift + p (WindowsではCtrl + Shift + p)でCommand Paletteを開き、Add Repositoryを選択します（図7.1）^{*2}。

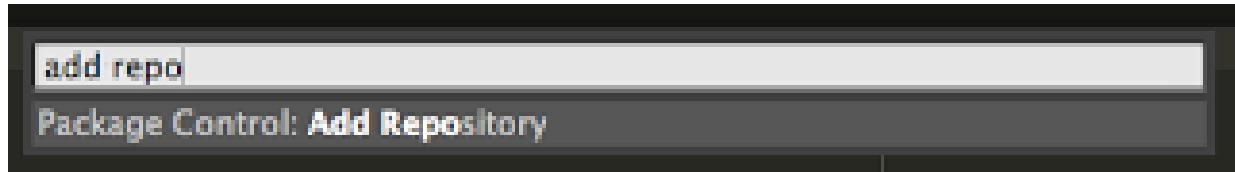


図7.1 Package ControlでAdd Repositoryを選択

下の方にリポジトリのURLを入力するフォームが表示されるので、<https://github.com/yanzm/ReVIEW>と入力してEnterを押します（図7.2）。

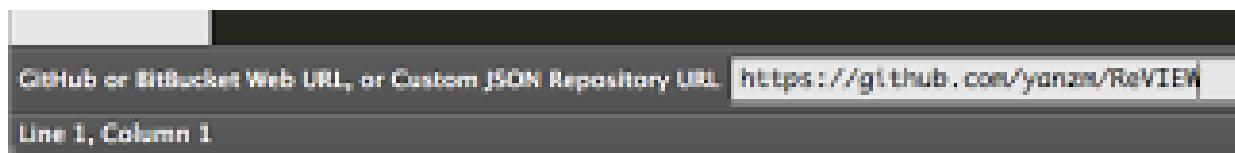


図7.2 リポジトリにReVIEWプラグインのURLを入力

リポジトリを追加したら、Command + Shift + p (WindowsではCtrl + Shift + p)で再度Command Paletteを開き、今度はInstall Packageを選択します（図7.3）。

^{*2} Package ControlにReVIEWリポジトリが登録されていればこのステップは不要なのですが、まだ登録依頼を出していくないです。すいませんでしたあああ。

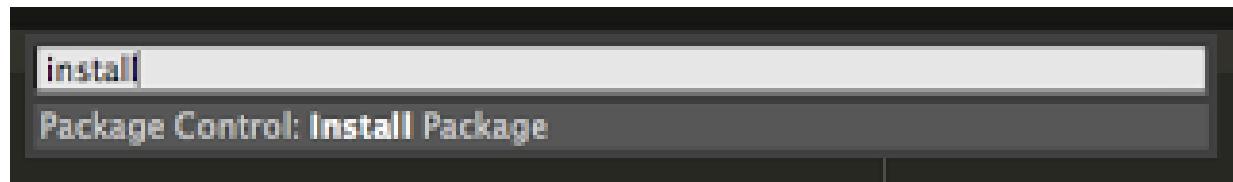


図7.3 Package ControlでInstall Packageを選択

インストールするパッケージを聞かれるので、ReVIEWを選択するとパッケージがダウンロードされインストールされます（図7.4）。

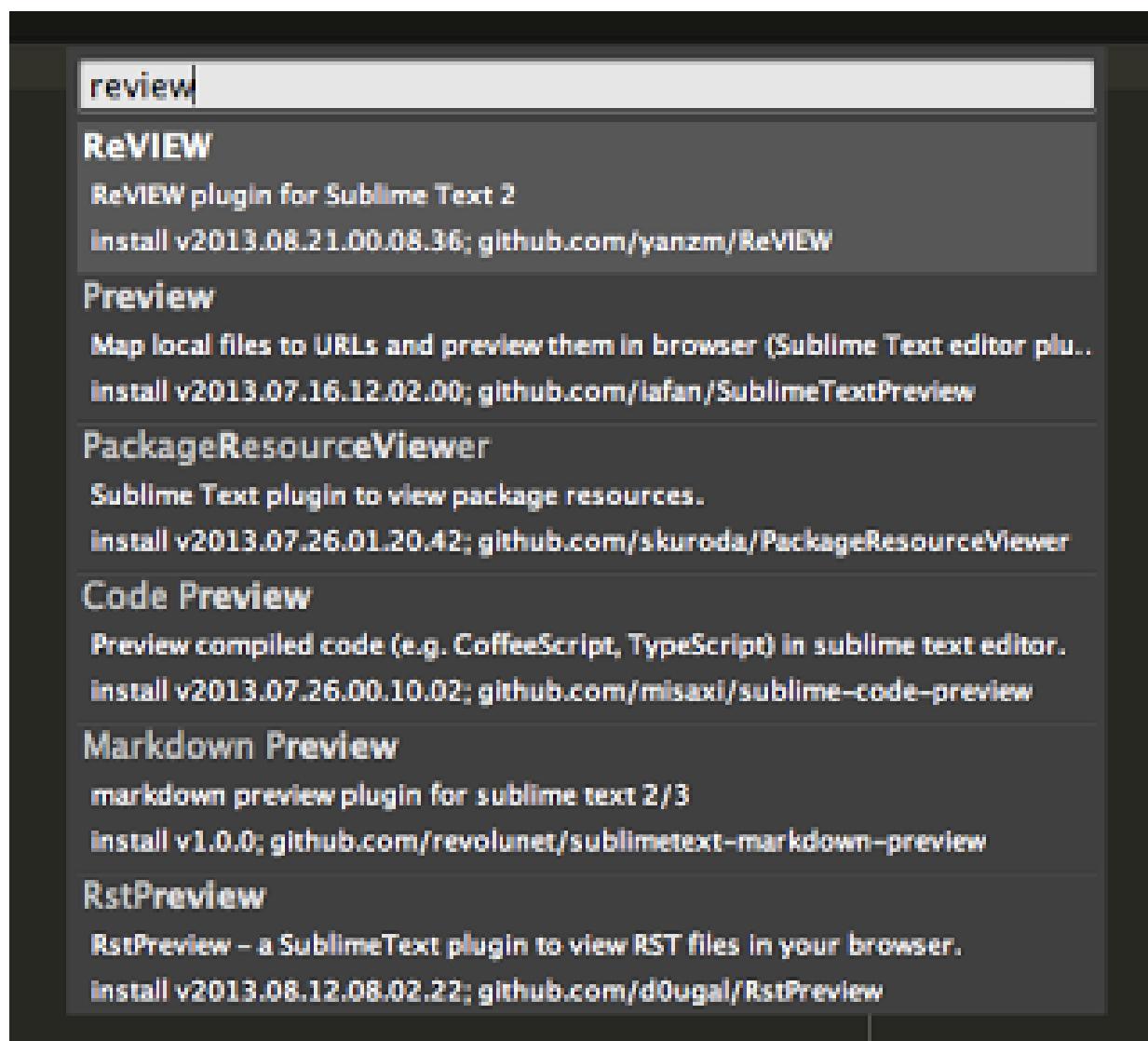


図7.4 インストールパッケージとしてReVIEWを選択

インストールが完了したらSublime Text 2を再起動しておきます。

7.2 ReVIEW プラグインのシンタックスハイライトを使う

ReVIEW プラグインのシンタックスハイライトを利用するには、ファイルの拡張子が .re である必要があります。また、.re のファイルを開いたときに [View] - [Syntax] が Review になっていない場合は、[View] - [Syntax] - [Open all with current extension as...] で Review を選択してください。

```
16
17 == Suica Reader の仕様書を Review で書いてみる
18
19 例として、私が公開している Suica Reader@<fn>{suicareader} という Android アプリの
20 書いてみます。
21 このアプリは、NFC を使って Suica や PASMO などの交通系ICカードの情報を見るものです。
22
23 //footnote[suicareader] [@<href>{https://play.google.com/store/apps/details?:suicareader}]
24
25
26 一口に仕様書と言っても、要求仕様書、機能仕様書、テスト仕様書などいろいろありますね。
27 ここでは要求仕様書を書いていきます。
28 要求仕様書を書くにあたって、Suica Reader に対する要求を明確にしておかないといけません。
29
30 === Suica Reader に対する要求
31
32 === 起動方法
33 1. ランチャーから起動できる
34 1. カードをかざして起動できる
35
```

図 7.5 ReVIEW プラグインによるシンタックスハイライト

7.3 ReVIEW プラグインの入力補完を使う

入力補完を表示するには、Ctrl + Space^{*3}を押します。また、インラインコマンド (@<...>{}) は「@コマンド名」まで入力して Tab を押すと、「@<コマンド名>{}」に変換されます。ブロックコマンド (//...[]{ ... //}) は「//コマンド名」まで入力して Tab を押すと、

^{*3} Mac OS では Ctrl + Space はデフォルトでは Spotlight に割り当てられているため、Sublime Text 2 を利用する場合は、Spotlight を別のキーに割り当てるこをおすすめします。

「//コマンド名 []{...}」に変換されます。

```
1 = ReVIEW で仕様書を作ろう
2
3 ReVIEW で仕様書を書くと
4
5 ami @<ami>{ ... }
6 b @<b>{ ... }
7 bib @<bib>{ ... }
8 ; bibpaper //bibpaper[][]{ ... //}
9 bou @<bou>{ ... }
10 br @<br>{} いますが、
11 chap @<chap>{ ... }
12 chapref @<chapref>{ ... }
13
14
15
16
17 == Suica Reader の仕様書を ReVIEW で書いてみる
18
19 例として、私が公開している Suica Reader@<fn>{suicareader} とい々
20 書いてみます。
```

図 7.6 Ctrl + Space で入力補完を表示した場合

第8章

仕様書を作ろう

ReVIEW で仕様書を書くと

- PDF にするのが簡単
- バージョン管理ツールで変更履歴や差分が見やすい
- 複数人で管理するときにマージが楽

などの利点があります。

仕様書_20131128.doc など見かけることもあるかと思いますが、見つけたら焼き払いましょう。

「敵だー！ 焼き払え！ なぎ払え！」

8.1 Suica Reader の仕様書を ReVIEW で書いてみる

例として、私が公開している Suica Reader^{*1} という Android アプリの仕様書を ReVIEW で書いてみます。このアプリは、NFC を使って Suica や PASMO などの交通系 IC カードの情報を見るものです。

一口に仕様書と言っても、要求仕様書、機能仕様書、テスト仕様書などいろいろありますね。ここでは要求仕様書を書いていきます。要求仕様書を書くにあたって、Suica Reader に対する要求を明確にしておかないといけません。

Suica Reader に対する要求

起動方法

1. ランチャーから起動できる
2. カードをかざして起動できる

*1 <https://play.google.com/store/apps/details?id=yanzm.products.suicareader>

交通 IC カード読み取り方法

1. NFC アンテナのあたりに交通 IC カードをかざして読み取れる
2. アプリを起動していない状態からでも読み取れる

データ表示方法

1. 個々のデータをカード UI で表示したい
2. 新しいデータが上にくるように表示したい

履歴へのデータ登録方法

1. 読み取ったデータを履歴に入れるかどうか選択したい
2. 自動でいつも履歴にいれる設定がほしい

履歴データの表示方法

1. カード毎の履歴を表示したい
2. どのカードの履歴を表示するか選択できるようにしたい

アプリ情報の表示方法

1. アプリのバージョンを確認できるようにしたい

デザイン

1. 2.x でも 4.x と同じ見ためがいい

CSV エクスポート

1. 読み取ったデータを CSV ファイルとして書き出したい（有料オプション）

駅名修正依頼

1. 簡単に駅名修正依頼メールが出せるようにしたい

要求としてはこんな感じです。では、この要求から要求仕様を書いてみましょう。

リスト 8.1: Suica Reader の要求仕様書 suicareader_specification.re

= 前提条件（使用条件）

- * ハードウェア構成
 - ** スマートフォンおよびタブレット
 - ** NFC に対応していること
- * OS
 - ** Android 2.3.3 以上

= 機能的 requirement

== 起動方法

1. ランチャー内のアプリアイコンをタップすることで起動できること
2. 新規起動時はカード読み取り画面であること
3. Felica カードがかざされたことをシステムが検知した場合、起動アプリの選択肢に本アプリが含まれること
4. NFC 検知から本アプリが起動された場合、読み取り画面を表示し、自動的にカードのデータ読み取りを開始すること

== 交通 IC カード読み取り方法

1. 読み取り画面が表示されている場合、カードの検知を優先的に受け取ること
2. システムから Felica カードがかざされたことを通知されたら、データ読み取りを開始すること
3. 読み取り中はプログレスを表示すること
4. 読み取りに失敗した場合は、再度かざすようメッセージを表示すること

== データ表示方法

1. 個々のデータの No.、日付、処理金額、処理内容、残高を表示すること
2. データの各項目をカード上に表示し、カードが縦に並ぶこと
3. No. が大きいデータが上にくるように表示すること
4. スクロールして No. の小さいデータを表示できること

== 履歴へのデータ登録方法

1. 読み取ったデータの表示画面に、履歴に追加するボタンを表示すること
2. ボタンがタップされたら、履歴にデータを追加すること
3. 読み取り時に自動で履歴に入れる設定（ON/OFF）があること
4. 自動履歴追加が ON の場合、履歴に追加するボタンは表示しないこと
5. 自動履歴追加が ON の場合、読み取ったデータを表示するときに履歴にも追加する

== 履歴データの表示方法

1. 履歴に追加されているカードの一覧がリストで表示されること
2. 特定のカードを選択したら、そのカードの履歴を表示すること
3. 履歴データの表示は、読み取りデータの表示と同じであること

== アプリ情報の表示方法

1. 設定に Info 項目を用意すること
2. Info 項目をタップされた場合、バージョンなどのアプリ情報をダイアログで表示すること

== デザイン

1. 2.x でも ActionBar を使ったデザインになること

== CSV エクスポート

1. 読み取ったデータを CSV ファイルとして書きだすボタンを用意する
2. 有料オプションを購入したアカウントのみボタンを表示する
3. ボタンをタップされたら、SD カード内の SuicaReader フォルダに CSV ファイルを書き出す
4. CSV ファイルは、1 行が 1 つのデータに対応し、No.、日付、処理金額、処理内容、残高が含まれること

== 駅名修正依頼

1. 読み取ったデータに対してユーザーが指定することで、駅名修正依頼メールの文章を生成するためのダイアログを表示できること
2. ダイアログでは、正しい駅名を入力できること
3. ダイアログから、メールのタイトル、本文が入力された状態のメールを起動できること

= 非機能的 requirement

== デザインの指定

- * Android のデザインガイドラインに従うこと

== パフォーマンスの指定

- * ANR が起こらないこと

== プログラム・サイズの制限

- * なし

== apk サイズの制限

- * なし

== 操作性に関する規定

* タッチインタラクションがある部分は、Android のデザインガイドラインに沿った大きさを確保すること

== 保守性に関する規定

- * ソースコードを git で管理すること

== 移植性に関する規定

- * NFC に関連する部分はライブラリプロジェクトとしてわけること

= 開発条件

- * 開発環境：Eclipse ADT bundle
- * プログラミング言語：Java (Android 開発向け)

= 納入条件

== 最終納入物

- * リリース用のキーで署名された apk ファイル

== 受け入れ条件

- * テスト仕様書に記述されたテストにすべてパスすること
- * テストにパス出来なかった場合、パス出来るように修正すること

= スケジュール

もうできてるから割愛

= 変更履歴

ここももうできてるから割愛

こんな感じになりました。

第8章 仕様書を作ろう

これを review-pdfmaker で PDF にします。このとき、仕様書向けの設定ファイルが必要になりますよね。

リスト 8.2: 仕様書向けの設定ファイル config.yaml

```
# 要求仕様書むけの設定ファイル例。

bookname: SuicaReader_要求仕様書
# タイトル
booktitle: SuicaReader 要求仕様書
# 著者
aut: あんざいゆき
# 出版
prt: 株式会社ウフィカ
# 版
date: v1.0.0

# LaTeX 用のスタイルファイル (sty ディレクトリ以下に置くこと)
# texstyle: samplemacro
# LaTeX 用の documentclass を指定する
# texdocumentclass: ["jsarticle", "b5paper,oneside"]
# 目次として抽出するレベル
toclevel: 3
# セクション番号を表示するレベル
secnolevel: 2
# 表紙を出力する
titlepage: true
# 目次を出力する
toc: true
# review-compile に渡すパラメータ
# params: --stylesheet=sample.css
```

.re ファイルを CHAPS というファイルに書いておく必要があります。

リスト 8.3: CHAPS

```
suicareader_specification.re
```

ここまでできたら、ついに PDF に変換！

```
$ review-pdfmaker config.yaml
```

2

第2章 機能的の要求

2.1 起動方法

1. ランチャー内のアブリアイコンをタップすることで起動できること
2. 新規起動時はカード読み取り画面であること
3. Felica カードがかざされたことをシステムが検知した場合、起動アプリの選択肢に本アプリが含まれること
4. NFC 検知から本アプリが起動された場合、読み取り画面を表示し、自動的にカードのデータ読み取りを開始すること

2.2 交通 IC カード読み取り方法

1. 読み取り画面が表示されている場合、カードの検知を優先的に受け取ること
2. システムから Felica カードがかざされたことを通知されたら、データ読み取りを開始すること
3. 読み取り中はプログレスを表示すること
4. 読み取りに失敗した場合は、再度かざすようメッセージを表示すること

2.3 データ表示方法

1. 各々のデータの No.、日付、処理金額、処理内容、残高を表示すること
2. データの各項目をカード上に表示し、カードが紙に並ぶこと
3. No. が大きいデータが上にくるように表示すること

図 8.1 生成された PDF 例

いい感じの PDF ができたよ！ 元がプレインテキストなのでソースコードと一緒に git で管理するのがおすすめです。

8.2 まとめ

この章では ReVIEW を使って仕様書を作る方法を紹介しました。実際の仕様書にはもっと多くの項目、情報が入ってくると思います。もちろん画像や表を入れることができますし、本文から画像や表を参照する際に自動採番なので、番号がずれる心配がありません。画像ファイルとプレインテキストに分かれているため、画像の差し替えも簡単です。奥付に履歴を入れることもできます。

このように、仕様書を書くのにも ReVIEW は適していると思います。ReVIEW で快適な開発を！

第9章

ReVIEW.js で学ぶ REVIEW 記法のお約束

9.1 ReVIEW.js ってなに？

筆者は、Ruby による実装であるところの本家 Review を参考に、JavaScript 実装^{*1}である ReVIEW.js を作成しています。現在のバージョンは 0.1.0 です。ReVIEW.js は Node.js とモダンブラウザ上で動作させることを念頭において開発されています。とりあえず試してみたい場合は <http://vakame.github.io/review.js/> で試すことができます^{*2}。

ReVIEW.js は以下の理由により開発がスタートしました。

1. Ruby 版 ReVIEW は原稿のパーサとコンパイラが分割されていないため、構文的に整合性が取りにくい。
2. 出力用のビルダ毎にサポートされている構文に差があり、何も考えずに書くとエラーになる場合がある^{*3}。
3. Ruby 版 ReVIEW を利用するにはコマンドラインを使わねばならないため、導入の敷居が高いため、Web ブラウザ上で動かしたい。
4. 利用可能な構文を処理系から得られず、書籍のために独自構文を追加した時にエディタ側のサポートが得にくい。

現実的な応用として、技術的なブログ記事の執筆に ReVIEW が使いにくいという問題があります。これは、ブログ用アプリケーションが ReVIEW 記法をサポートしていないためです。技術ブログ→書籍化 という流れも多い昨今ですので、ブログ上の記述に ReVIEW がサポートできるとなるとメリットも大きいでしょう^{*4}。そのための策として、ReVIEW.js が有效地に機能するかもしれません。

^{*1} まあ、JavaScript 版といいつつ実装は TypeScript なんですね。TypeScript 可愛いよペロペロ。

^{*2} 筆者は API を美しくすることは得意ですが UI を美しくするのはめっちゃ不得意なので誰か助けてくれてもいいのよ！？

^{*3} 無いなら自分で作ればいいじゃない！ <https://github.com/kmuto/review/pull/179>

^{*4} Qiita さん見てますかー？ チラッチラッ

もっともらしいことを書きましたが、本当はブラウザ上で ReVIEW の原稿の編集が行える @sys1yagi 製ツール PreVIEW^{*5} が裏側で必死に Ruby 版動かしているのを見て、ブラウザ上で全部やっつちまいなよ！ と思ったのが開発スタートの原因です。アトウッドの法則「全ての JavaScript で記述可能なアプリケーションは、ゆくゆくは JavaScript で書き直される」の通りですね :)

よりリッチな入力補完のための工夫

ReVIEW.js では、エディタ上での快適な編集をサポートするために利用可能な構文を処理系から取得することができます。 <http://vvakame.github.io/review.js/> で表示される構文は、全て処理系から得たもので、引数の数などは実際に受理可能なもののみが出力されています（リスト 9.1）。

リスト 9.1: 処理系から得られる出力のサンプル

```
{
  "rev": "1",
  "SyntaxType": {
    "0": "Block",
    "1": "Inline",
    "2": "Other",
    "Block": 0,
    "Inline": 1,
    "Other": 2
  },
  "acceptableSyntaxes": [
    ...,
    {
      "type": 0,
      "symbolName": "image",
      "argsLength": [
        2,
        3
      ],
      "description": "図表を示します。\\n//image[sample] [サンプル] [scale=0.3]{\\n メモ\\n//}\\n という形式で書きます。\\n 章のファイル名が test.re の場合、images/test-sample.jpeg が参照されます。\\n 画像のサイズを調整したい場合、scale で倍率が指定できます。\\n 中に書かれて いるメモは無視されます。"
    }
  ]
}
```

*5 <https://github.com/sys1yagi/PreVIEW>

```
},  
...  
]  
}
```

ビルダ毎のサポートしている構文の統一

ReVIEW.js では、字句解析&構文解析、意味解析、コード生成が切り分けられています。また、入力補完候補を得るために全ての構文の収集が実行時に可能なように設計されているため、利用するビルダ全てでサポートされている構文に不足がないかを事前に検査することができます。そのため、実装が足りないビルダを使おうとするとエラーにすることができるため、ビルダ作成時の効率的なテストとして役立てるることができます。

ReVIEW パーサの作成

ReVIEW 記法に対してパーサを作成しました。peg.js という、PEG 記法を用いたパーサジェネレータを利用しています。<https://github.com/vvakame/review.js/blob/master/src/main/peg/grammar.pegjs> にて、内容が参照できます。PEG 記法を用いたパーサジェネレータ同士でも書き方に互換性がないので何も変えずにそのまま流用、というのは難しいのですが、Ruby プログラムから頑張って移植するよりはパーサ部分の汎用性が高くなります。

本章のこの後の節では、この ReVIEW パーサを題材にして正しい ReVIEW 原稿の書き方を学んでいきましょう^{*6}。

まだまだ Ruby 版には追いつかない

もちろん、長い間幾多のユーザの利用に耐え、開発が継続してきた Ruby 版 ReVIEW には追いつけていません。まずは、この ReVIEW 本の原稿の処理全てを ReVIEW.js で…といいたいところですが、実装されていない機能がそれなりにあるため、それもままなりません。悲しいですね。

^{*6} 免責事項：頑張って PEG 記法に起こして、割りと頑張って Ruby 版が受理できる原稿を同じく受理できるようにしたつもりだけど、頑張りきれていなかったらごめんね！ バグ報告もよろしくね！

実装が追いついていない機能は以下の通りです。

- 本としての処理
 - PDF や epub の生成
- Text と HTML 以外のビルダの作成。特に重要なのが LaTeX のビルダですが全く手付かずです。
 - LaTeX ビルダ
 - idgxml ビルダ
 - markdown ビルダ
- サポートされていない記法の存在
 - LaTeX ビルダ用の記法なので実装されていない
 - * txequation ブロック記法
 - * m インライン記法
 - Ruby 版の仕様がよく理解できないので実装されていない
 - * chap インライン記法
 - * title インライン記法
 - * chapref インライン記法
 - * bibpaper ブロック記法
 - * bib インライン記法
 - 実装方法を考えている最中なので実装されていない
 - * raw ブロック記法
 - * raw インライン記法
 - * graph ブロック記法
 - * table ブロック記法
 - * tsize ブロック記法
 - * table インライン記法
 - * コラム用の記法
- プリプロセッサ
 - #@mapfile など
- その他細々としたもの
 - 気がついていないものがたくさんある気がする…！

table 記法マジラスボス…。というか仕様がでかいです。入れ子が可能な設計にすると単純にタブ文字で分割してやればいいじゃん、というわけにもいかず。正直 table 記法さんだけ特殊すぎる所以何か良い記法を編み出すとかしたほうがよいような気が…。

9.2 ReVIEW 記法を調べよう！

長い前置きでした。ReVIEW.js で使っている PEG 記法をシンタックスダイアグラムに変換し、それを見ていきます。

<http://bottlecaps.de/convert/> で `grammer.pegjs` (ReVIEW.js の構文ファイル) を投げ込み、EBNF に変換し、さらに <http://bottlecaps.de/rr/ui> に投げてシンタックスダイアグラムを得ます。

さて、その結果を見て行きましょう。

肩慣らしに基礎となる簡単なブロックから

Digit

`[0-9]` というのは、`0, 1, 2, 3, 4, 5, 6, 7, 8, 9` のいずれかにマッチします。つまり、数字 1 文字であればなんでもいい…ということになります。正規表現の書き方と同じですので、わかりやすいですね。

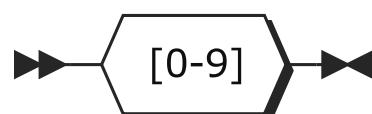


図 9.1 Digit. `0, 1, ..., 9` のいずれか数字 1 文字

Digits

Digit から出た線が、またしても Digit に戻ってくるルートがあります。つまり、Digit (数字 1 文字) が 1 回以上連続しているもの全てにマッチします。

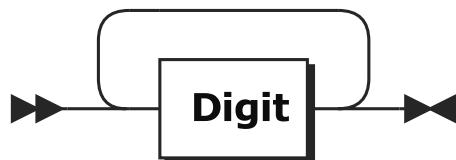


図 9.2 Digits. Digit が 1 回以上連続していること

AZ

Digit は数字 1 文字でしたが、今度はアルファベットの小文字 1 文字にマッチします。AZ という名前だと大文字 1 文字っぽい感じがしますね…。なぜ、こんな定義が必要かというと、Ruby 版 ReVIEW の @<nanika> {} の nanika の部分はアルファベット小文字のみで構成されていないといけない制約があり、それを真似するためです。

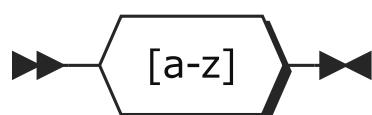


図 9.3 AZ. a, b, … , z のいずれかアルファベット小文字 1 文字

Newline

これは Windows で一般的な CRLF と Unix 系で多く見られる LF を 1 つの改行とみなしてマッチします。PEG 記法では、上に書いてあるものが優先してマッチされます。

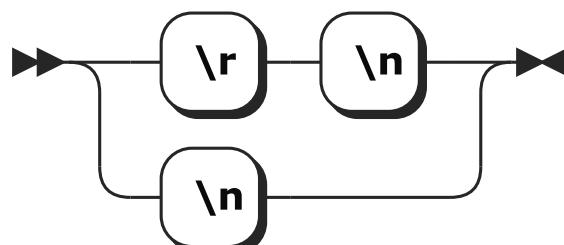


図 9.4 Newline. 改行を表す

Space

空白 1 文字と認識されるものを定義しています。半角スペース、全角スペース、タブが含まれます。

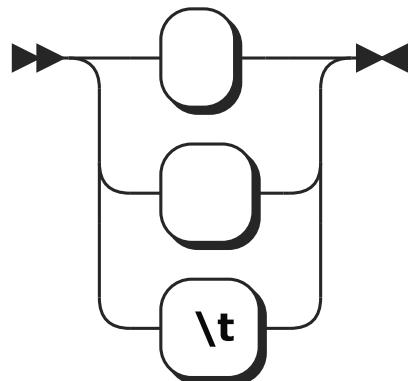


図 9.5 Space. 半角スペースとタブ、改行を含む

_ (スペーサ)

ReVIEW 上、読み捨てできる文字の塊と認識されるものにマッチします。迂回路があることからわかる通り、0 文字にもマッチします。つまり、このパターンは失敗しません。

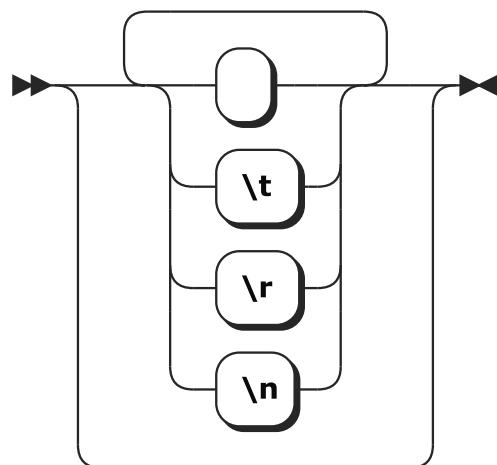


図 9.6 _ . 読み飛ばすべき空白

EOF (End Of File)

これはファイルの最後であることを指します。つまり、もうこの後に何も続いてはならない…！ というパターンにマッチします。



図 9.7 EOF. ファイルの終端

ReVIEW 記法の構造に迫る

いよいよ ReVIEW 記法の詳細に迫っていきましょう。

Start

ReVIEW 文書のスタート地点です。全ての ReVIEW 文章は複数のチャプター（章）を含むことができます。含むことができる、というだけで、実際は複数の章を 1 つの .re ファイルに押し込むのは良いやり方ではありません。処理系を甘やかすためにも、1 ファイル 1 章になるようにしましょう。

最初と最後に _ (無視するべき空白) があります。これは無視してしまうので出力には関係ありません。



図 9.8 Start. ReVIEW 文書は複数のチャプターから成り立つ

Chapters, Chapter

Chapter は、日本語訳すると "章" にあたりますが、ReVIEW.js 内部では 章、節、項、段、小段 の全てを指します。これは、Ruby 版の ReVIEW が章（見出し深さ 1）の下に項（見出し深さ 3）を許す設計になっているためです。これも、可能なだけで実際に深さ 2 を飛ばして

しまうのは悪い作法なので、処理系レベルで制限してしまったほうが良いとは思うのですけどね。

1つの Chapter は Headline (見出し本体) と Contents (コンテンツ) からなります。

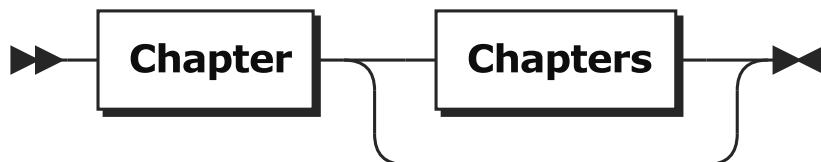


図 9.9 Chapters. 複数のチャプターから成る

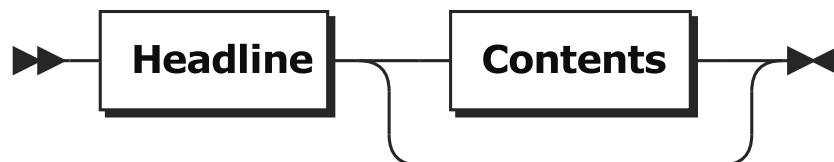


図 9.10 Chapter ヘッドライン (見出し) とコンテンツ (本文) から成る

Headline

ようやっと、生の文字列の指定がでてきました。Headline は、日本語訳すると "表題" になります。文章を書き始める時に = 表題 と書くと、これがそのまま Headline になります。途中で BracketArg, BraceArg (後述) が出てきます。これらはそれぞれ、[] と {} に対応しています。[] は、コラムを書く時に使います。=[column] コラムの表題と書くと、コラムになります。コラムを終える時は次の Chapter を書き始めるか、=[/column] で閉じる必要があります。

[] の中に来るものは、必ず column に限定されますが、ReVIEW.js ではそのような制約を設けていません (実装してる時は知らなかつたんだ！)。さらに、=[/column] でコラムを終えることもできません (実装してる時は知らなかつたんだよ！！)。

= 表題の "表題" にあたる部分は SingleLineContent (後述) として定義されています。このため、ReVIEW.js では表題中でもインライン記法が利用可能になっています。LaTeX などの環境をターゲットにした時にポータブルにならなさそうではあります…。

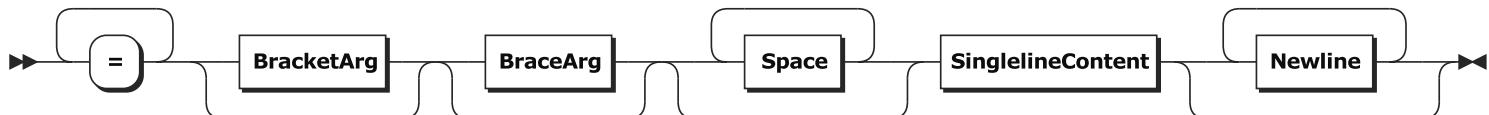


図 9.11 Headline. 見出しを表す

Contents, Content

Chapter の本文の部分にあたります。それぞれ、SinglelineComment（1行コメント）, BlockElement（ブロック記法）, Ulist（箇条書き）, Olist（数字付き箇条書き）, Dlist（用語の定義）, Paragraph（段落）があり、Chapter の本文はこれらの繰り返しからなります。

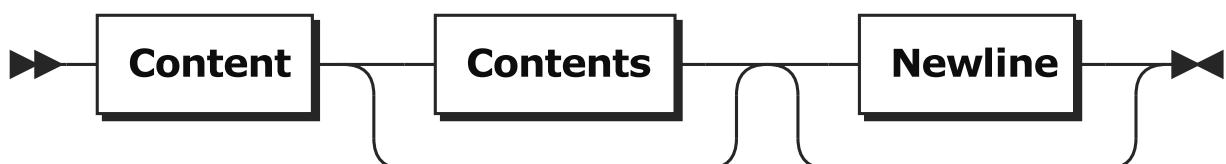


図 9.12 Contents. Content の繰り返しから成る

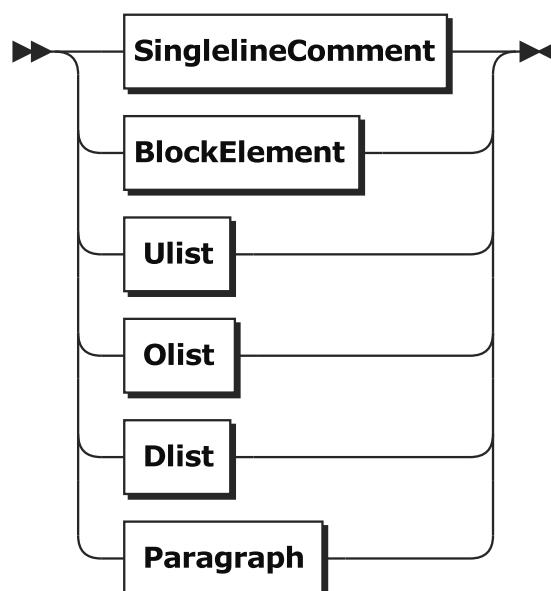


図 9.13 Content. いろいろなものが本文に成りうる

Paragraph

Paragraph は ReVIEW.js が使っているパーサジェネレータの PEG.js の処理の仕様上、ParagraphSubs, ParagraphSub を持ります。ざっくり意図を説明すると、Paragraph は InlineElement (インライン記法), ContentText (地の文, プレーンテキスト) の繰り返しから構成されます。

Paragraph は、行頭に = が来る (新たな Chapter の始まりを意味する) ことのない場合のみにマッチします。ダイアグラム上では省略されていますが、実際の PEG 記法では以下のようになっています。!"=" による、否定先読みが入っていることがわかります。

```
Paragraph "paragraph"
= !"=" c:ParagraphSubs -
;
```



図 9.14 Paragraph. 行頭が = ではない ParagraphSubs から成る

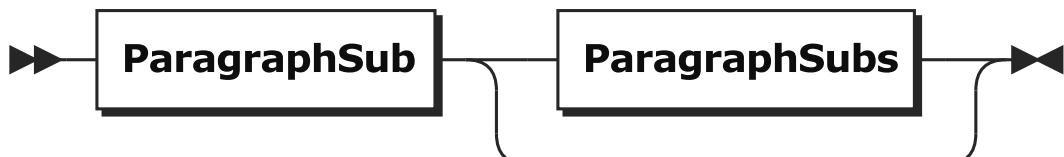


図 9.15 ParagraphSubs.

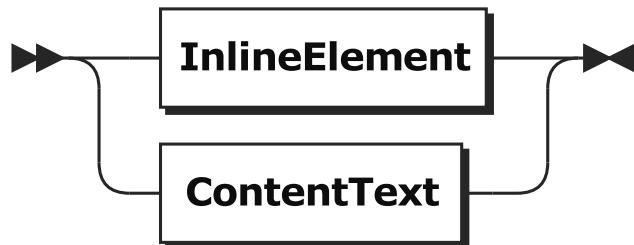


図 9.16 ParagraphSub. インライン記法とプレーンテキストから成る

ContentText

これは地の文、かつ、プレーンテキスト（INLINE記法ではない）です。

ダイアグラム上は現れていませんが、大量の否定先読みが入っています。

先頭が

- 改行ではない
- Headline ではない（新しい Chapter の始まりではない）
- SinglelineComment ではない
- BlockElement ではない（BlockElement は Paragraph ではない）
- Ulist ではない（Ulist は Paragraph ではない）
- Olist ではない（Olist は Paragraph ではない）
- Dlist ではない（Dlist は Paragraph ではない）

かつ

- InlineElement ではない（InlineElement は ContentText ではない）
- 改行ではない

を満たす文字列にマッチします。改行または EOF を見つけたらそこで処理を打ち切り、1つ改行を食べられたら食べます。その後に続くものがさらに ContentText の場合は、ContentText の続きとして処理します。

はあっ！複雑！わかりにくく、辛く長いルールですね。一回日本語でまとめてみましょう。

ContentText は、改行が2つ連続しない限りマッチする(2つ連続でマッチしたら Paragraph の切れ目)。“連續でマッチする文字列”と呼ばれるものは、ブロック記法や新しい Chapter の始まりなど、Paragraph の切れ目に当たるものが来ないようにする。また、Paragraph の中

には ContentText と InlineElement が混在する場合があるため、InlineElementを見つけたらその直前までを1つのContentTextとする。

自然言語の記述力の高さが光ります。これだけの指示でパソコンが全部理解してくれると、パソコンから僕への愛を感じるのですが、残念ながら今まで僕はパソコンさんからの愛を受け取ったことがありません。

```
ContentText "text of content"
  = text:$(` !Newline !Headline !SinglelineComment !BlockElement \ # 本当は一行
    !Ulist !Olist !Dlist ( !InlineElement [^\r\n] )+ Newline? ContentText?
`;
```

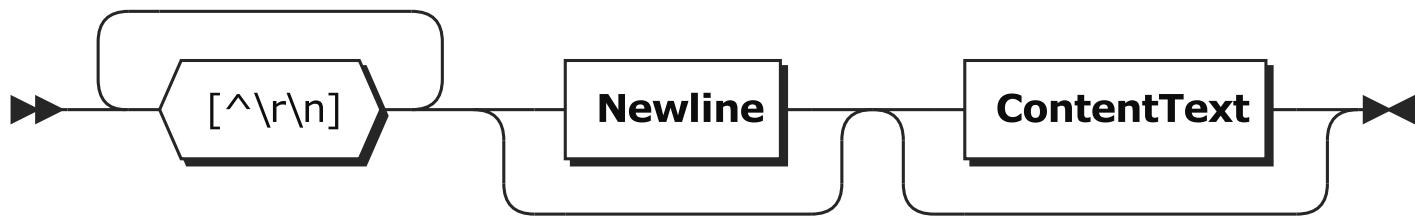


図9.17 ContentText. プレーンテキスト

BlockElement

これはブロック記法を表します。例えば、以下のようなものです。

```
//list[sample][サンプルだよー]{
  console.log("Hello world!");
}
```

ブロック記法は行頭が // で始まり⁷、そこにシンボルが続きます。シンボルは AZ (a から z までのアルファベット小文字) のみ許可されます。わかりやすさのためには、//リスト [サンプル] [サンプルだよー] というように日本語も許したほうが非技術者の人にもわかりやすいと思うのですけどね。ここは、Ruby版実装に倣っています。

⁷ 行頭で始まり、とは書きましたが、それを強制するルールはないので、BlockElementのマッチの判定が始まつた時は、必ず改行文字が食われた直後である、というようにルールを作ることで対応しています。辛い。

続く BracketArg は [] の事です。0 個以上の引数を書くことが可能です。

最後に { と //} を使って対象範囲をくくります。ブロック記法は範囲の指定ナシにする事もできるため、末尾の改行を食べる _ のみが配置されている分岐もあることがわかります。

対象範囲は BlockElementContents として表されます（後述）。

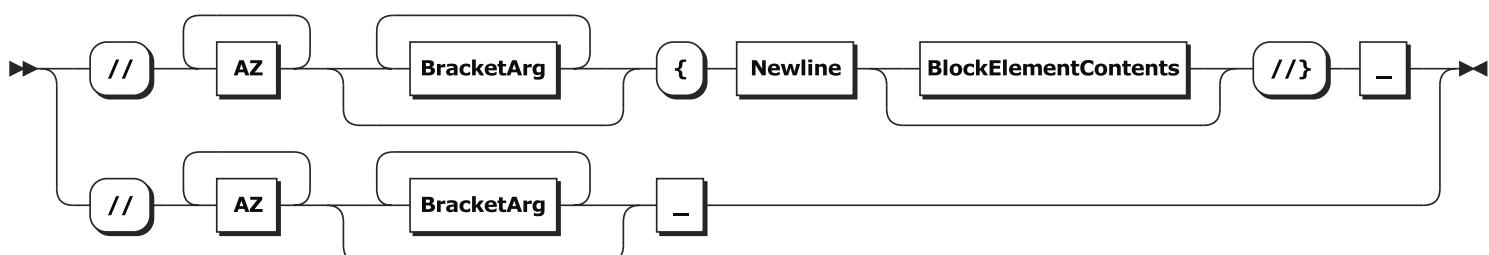


図 9.18 BlockElement. ブロック記法を表す

InlineElement

これはインライン記法を表します。例えば、以下のようなものです。

まずは@<list> {sample}を参照してください。

インライン記法は @< で始まり、そこにシンボルが続きます。さらに畳み掛けるように >{ ! 内容に InlineElementContents (後述) ! おしまいに } と続けます。

さて、ブロック記法と違って、こちらのシンボルはなぜ改行と>以外の全ての文字を許すようになっているんでしょうか。あとで Ruby 版の挙動見て修正しないと…。まあ、文法規則修正するだけで処理系は全く手を付けずに済むので楽なもんですけどね！（強気）

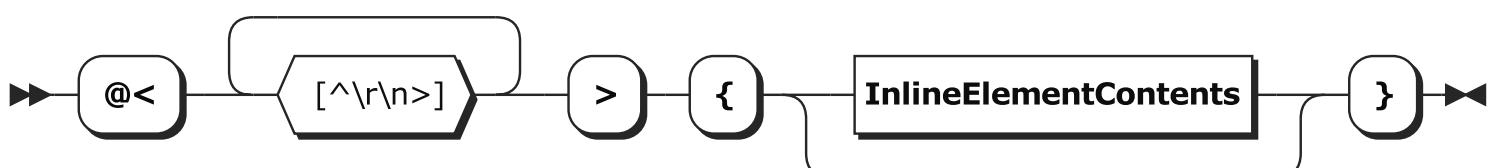


図 9.19 InlineElement. インライン記法を表す

BracketArg

すでに何度か登場していますね。

改行または]以外の文字を引数として取ります。引数にあたる部分を [と] で囲みます。

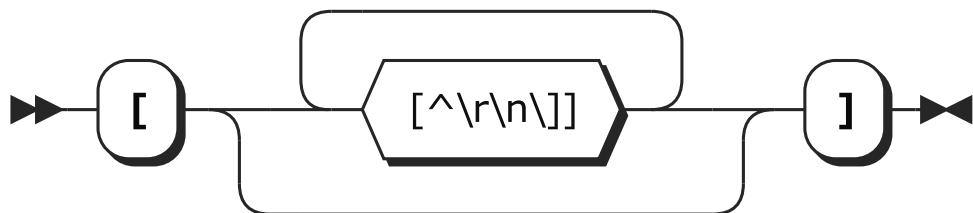


図 9.20 BracketArg. []を使った引数の指定を表す

BraceArg

すでに何度か登場していますね。

改行または}以外の文字を引数として取ります。引数にあたる部分を { と } で囲みます。

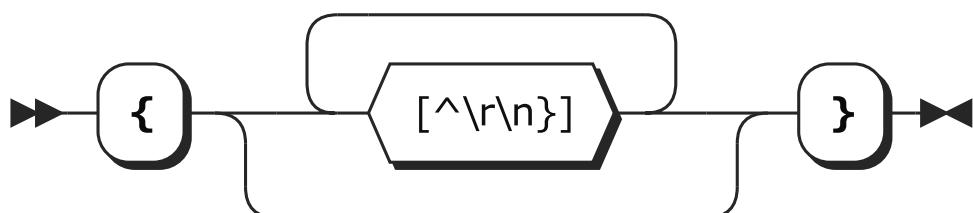


図 9.21 BraceArg. {}を使った引数の指定を表す

BlockElement のコンテンツ部分

ブロック記法の本文の部分です。ブロック記法内ではインライン記法が有効であることと、モノによっては Paragraph などの文脈をサポートするものがあります。例えば、lead ブロック記法ではまるでブロック記法の中ではないかのように、段落やら箇条書きやらを認識してくるので、頑張ってこれも対応する必要があります。そのための BlockElement-Contents, BlockElementContent, BlockElementParagraph, BlockElementParagraphSub, BlockElementContentText です。

既出の内容とほぼ同一なので、全体的に説明をカットしますが、BlockElementContentText

だけ見どころがあるので取り上げます。ContentText と BlockElementContentText を比較してみましょう。

```
BlockElementContentText "text of content in block"
= text:$(`& !"/` !SinglelineComment !BlockElement \ # 本当は一行
    !Ulist !Olist !Dlist ( !InlineElement [^\r\n]+ Newline? )+ )
;
```

```
ContentText "text of content"
= text:$(`!Newline !Headline !SinglelineComment !BlockElement \ # 本当は一行
    !Ulist !Olist !Dlist ( !InlineElement [^\r\n]+ Newline? ContentText?
`;
```

ContentText との一番大きな差異は、先頭の、何を見つけたらマッチングを打ち切るか、という指定です。BlockElementContentText はブロック記法の終端、つまり //} で始まらないという指定です。ContentText は Paragraph、ひいては Chapter の始端である Headline で始まらないという指定です。この差異のおかげで共通化出来ない部分になっちゃってるんですね。処理上は一緒に扱ってるのでまあ PEG 記法上の問題なんですが。

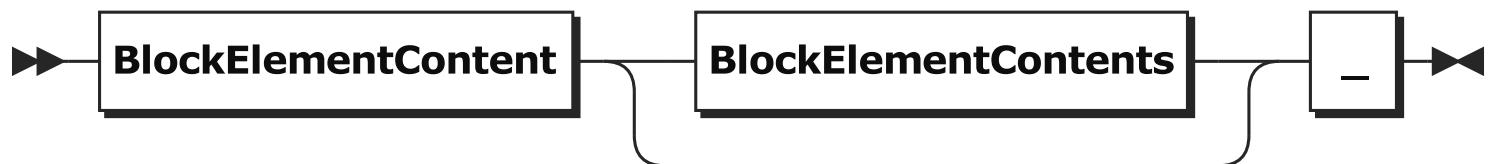


図 9.22 BlockElementContents. Contents 相当

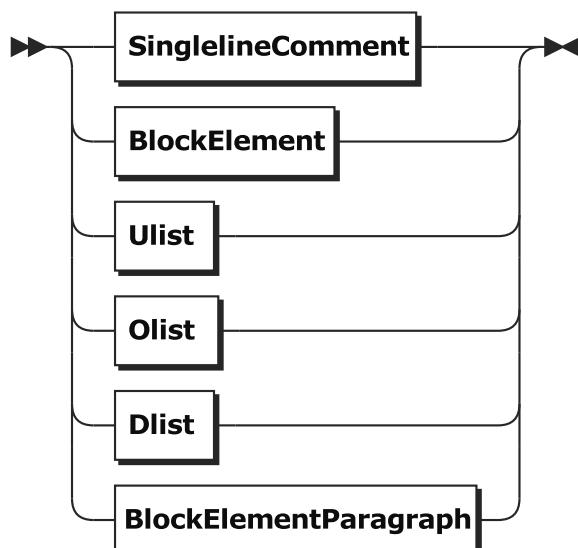


図 9.23 `BlockElementContent`. Content 相当



図 9.24 `BlockElementParagraph`. Paragraph 相当 先頭での = の否定先読みを行わない

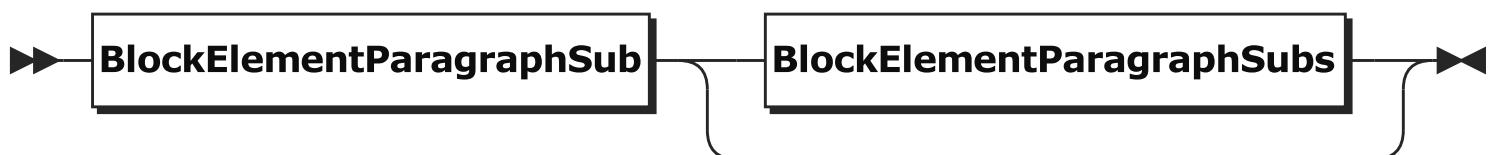


図 9.25 `BlockElementParagraphSubs`. ParagraphSubs 相当

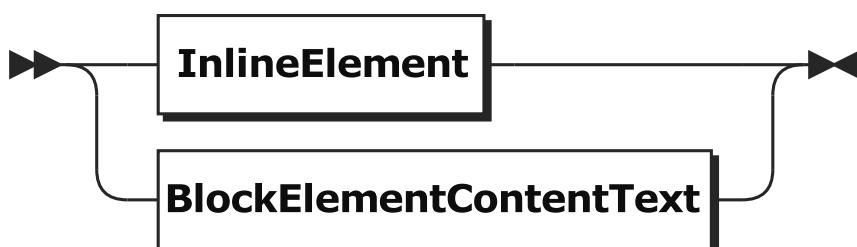


図 9.26 `BlockElementParagraphSub`. ParagraphSub 相当

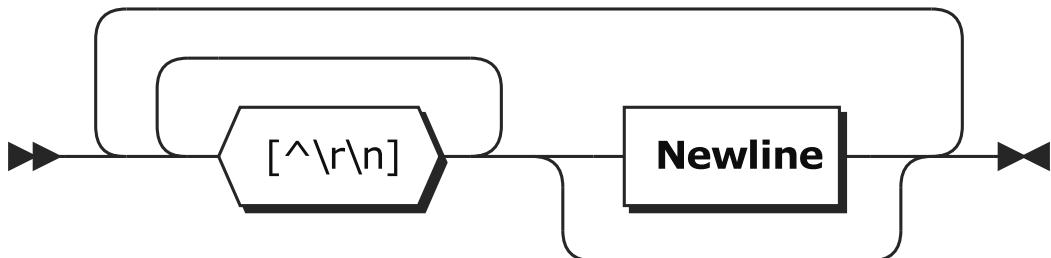


図 9.27 BlockElementContentText. ContentText 相当だが否定先読みが少し増えている

InlineElement のコンテンツ部分

インライン記法の中はできることが限られているため、定義も簡素になっています。

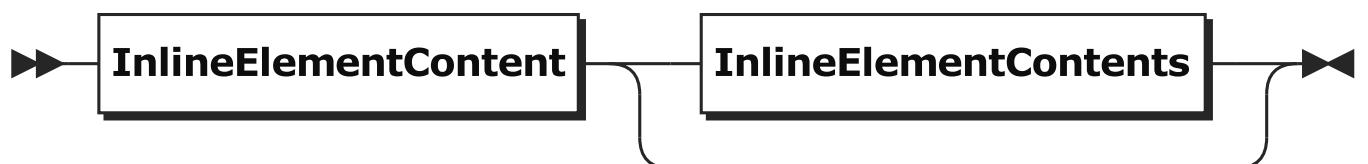


図 9.28 InlineElementContents. Contents に相当

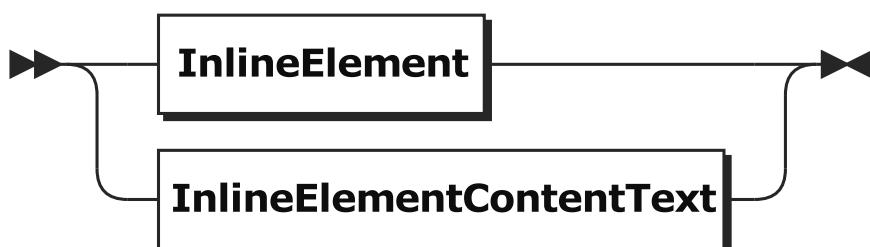


図 9.29 InlineElementContent. Content に相当

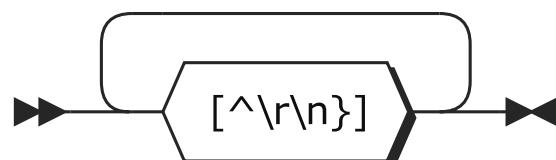


図 9.30 InlineElementContentText. ContentText に相当 簡素！

SinglelineContent

そろそろ解説も終盤戦に近づいてきた信じたいところで SinglelineContent です。その下に ContentInlines, ContentInline, ContentInlineText があります。これも Contents, Content, ContentText と同様の構造です。Headline のように 1 行で書かれる必要のある文を表します。後述の UlistElement, OlistElement, DlistElement でも利用されます。

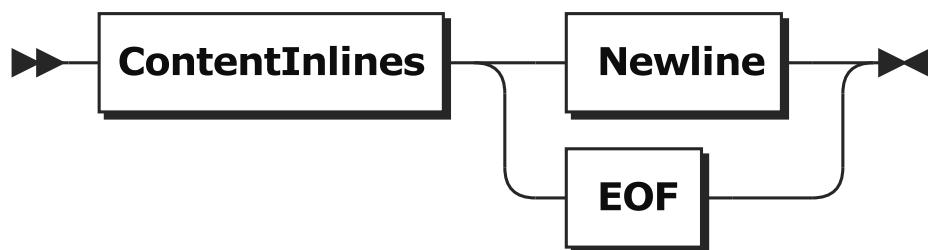


図 9.31 SinglelineContent. 1 行に収まるコンテンツ

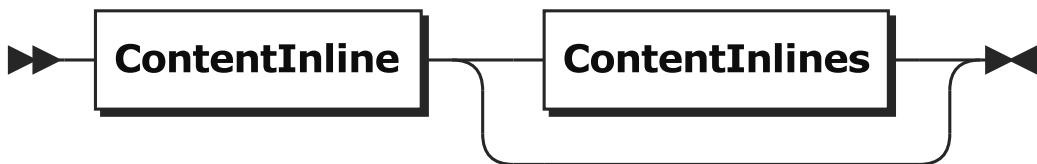


図 9.32 ContentInlines. Contents に相当

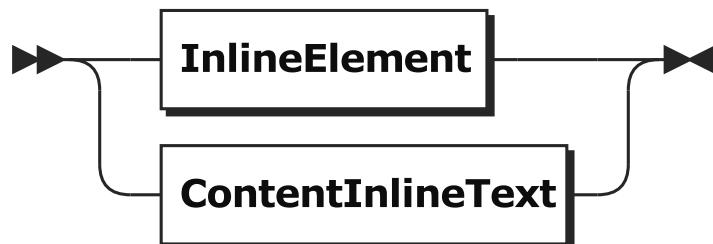


図 9.33 ContentInline. Content に相当

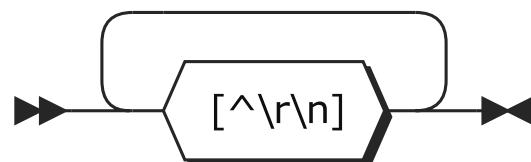


図 9.34 ContentInlineText. ContentText に相当

Ulist

箇条書きを表します。見て分かる通り、行頭には必ず半角スペース 1 つ以上が必要です。また、その後の * が複数回連続させることで深さを変えることができます。筆者は Markdown のように、* 文字の前にスペースを入れて深さを調節する方法より、ReVIEW の方法のほうが簡単に深さの変更ができるため便利なので好きです。

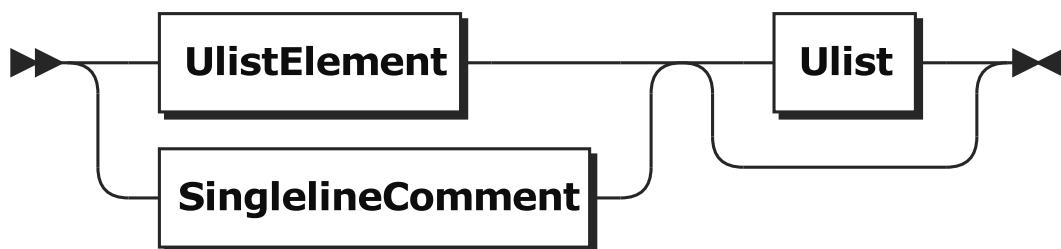


図 9.35 Ulist. 箇条書き

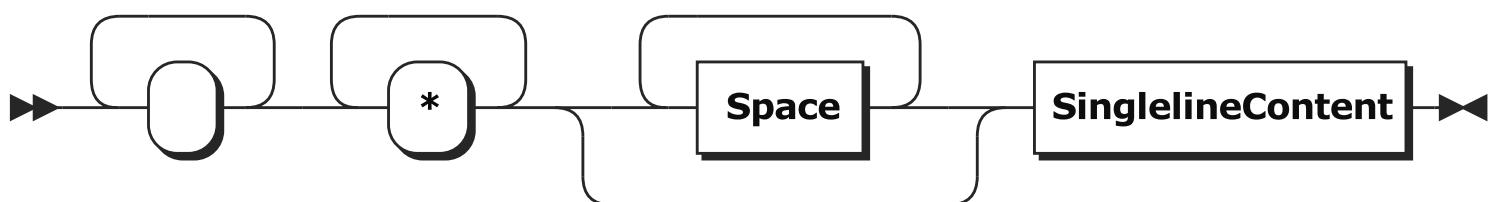


図 9.36 UlistElement. 1 行ごとの記法

Olist

数字付きの箇条書きを表します。これも Ulist と同様に、行頭に半角スペース 1 つ以上が必要です。また、1. のような数字はいくつでも問題ない構造になっていて、Ruby 版 ReVIEW の仕様では自動で採番されなおすことになっていますが、一部ビルダの出力先の都合上、人力で正しい番号どおりに並べたほうがよいそうです。

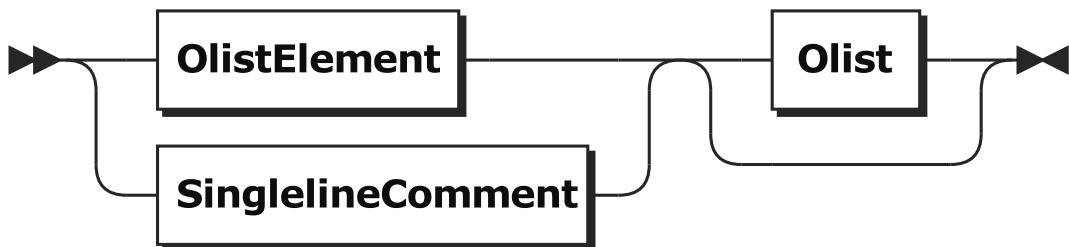


図 9.37 Olist. 数字付き箇条書き

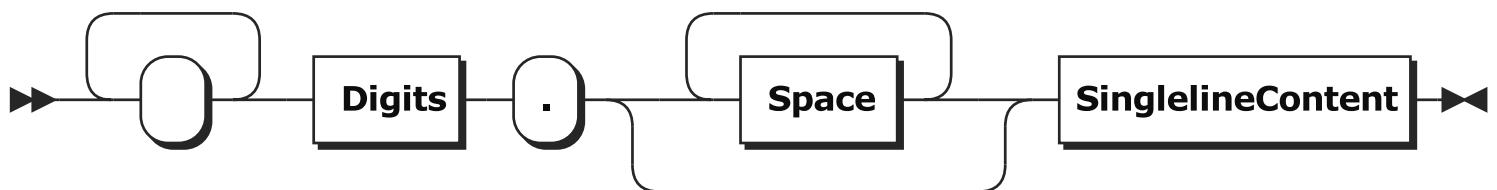


図 9.38 OlistElement. 1 行ごとの記法

Dlist

用語の定義と解説を表します。これはめずらしく行頭に半角スペースがなくても大丈夫ですしかし、解説の行はかならず行頭に半角スペースまたはタブがなければなりません。

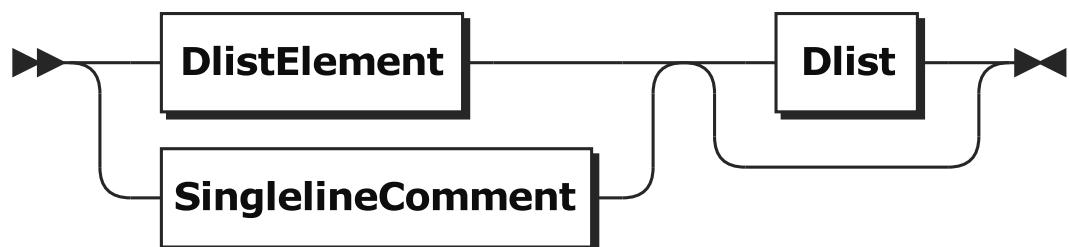


図 9.39 Dlist. 用語の定義と解説を表す

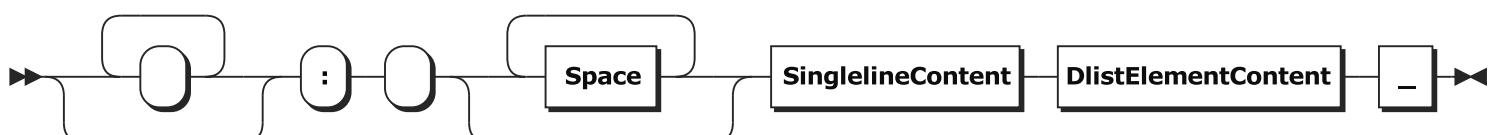


図 9.40 DlistElement. 定義の行

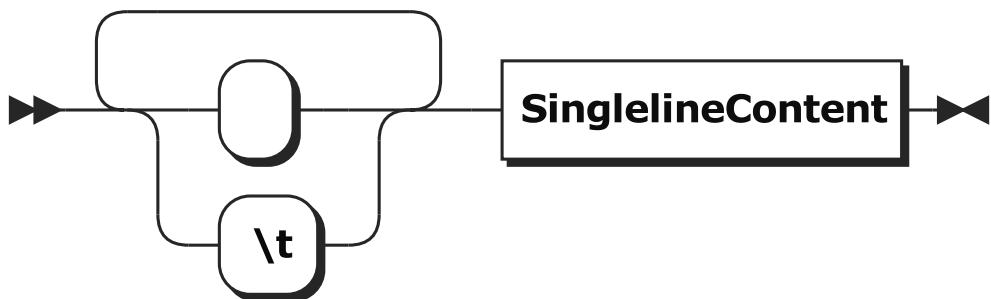


図 9.41 DlistElementContent. 解説の行

SinglelineComment

これは 1 行コメントで、処理上は単に無視されます。

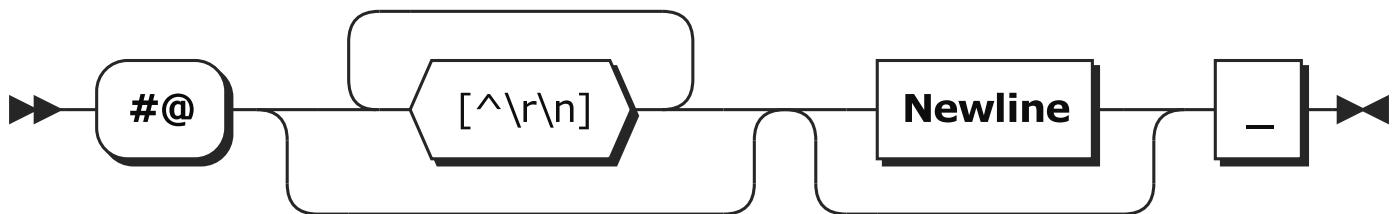


図 9.42 SinglelineComment. 1 行コメント

まとめ

いかがだったでしょうか？ これが ReVIEW の書き方の全てです。字句解析&構文解析上は上記に当てはまるように書けば良いでしょう。

一部、意味解析のフェーズで撥ねられたり、構文木を組み替える処理を行う場合もありますが、概ねこの通り解釈されます。

複雑な構文はほとんど存在せず、タイトルを書いて、ブロック記法、インライン記法、箇条書きと用語の定義の記法さえ覚えれば、ReVIEW の正しい構文の書き方は覚えたと言って良いでしょう。問題となるのはブロック記法、インライン記法の各種シンボルの使い方を覚えるだけです！！

とか書くと、HTML なんか <と> で囲むだけでハイパーシンプルじゃないかふざけんな！ という声も聞こえてきそうですが。それでも、ReVIEW 記法を覚えるのは簡単ですよね！！

各種シンボルの解説については第 4 章「記法をおぼえよう」を参照してください。

では、快適な ReVIEW ライフを！

最後に、ReVIEW.js はバグ報告、pull request、開発に専念するための出資などを隨時受け付けております。よろしくお願いします☆

第10章

同人誌を作ろう

本章は技術的な内容を同人誌にまとめるという、ごくありふれた創作活動の紹介です。必ずこうすべきだ、というものではなく、ReVIEW を使った同人誌ってこうやってつくったんだよ、という体験記です。同人誌というジャンルは広く、多様な表現が可能です。技術誌としての企画、構成、編集、入稿まで制作フローに興味があるとより楽しめると思います。そしてこれを読んで作ってみたいな、とおもった方はガイドラインとしてご活用ください。

また紙面においての例の多くは TechBooster の mhidaka（通称、ひつじ）が経験したことがベースです。具体的にはコミックマーケット 84 で頒布した『Effective Android』の同人誌版～電子版～商業出版へ展開した経緯や、コミックマーケット 85 にむけた同人活動、商業誌への寄稿や単著、共著の執筆活動のノウハウです。

10.1 企画と募集

よろしい、ならば同人誌だ

同人誌を作りたいと思った時がつくり時です。さあ作ろう、作りましょう、作るんだ！ 同人誌を書き出す前に、決めておくとスムーズなことがいくつかあります。

- ・参加イベント
- ・執筆内容と構成案

参加イベントの決定は、もっともスケジュールに影響を与えます。なるはやで決めてしまいましょう。技術書を頒布する^{*1}のに最も適したイベントは、コミックマーケットです。それ以外にもいくつか探してみましたが、技術書がある程度まとまった規模になるイベントは、ほとんどありませんでした。つまり、一択です。

^{*1} 同人用語。有償、無償問わず、イベントで同人誌を提供すること

執筆内容と構成案については言わずもがな、ですね。この段階ではなかなか予測がつきにくいかもしれません、完成時のページ数は、印刷所へ入稿するために重要な情報です。サイズは B5 (JIS) が一般的で、おすすめです。

印刷にかかるお金は大きく異なるため、ページ数と部数は早めに想像しておきましょう。この段階では、きっちり固まっている必要はありません。経験から技術書は厚い薄い本になりやすいです。ちなみに筆者のサークルがコミックマーケット 84 で刊行した『Effective Android』は 100 ページを予定していて、気が付くと 184 ページの書籍になりました。まあそんなもんですよ^{*2}。

あなたが「どうせつくるなら品質も気にしたいな」と思うなら、最初に決めるべき内容が、もうひとつあります。それは、本のグランドデザインです。執筆内容にあわせて本のイラストやメインカラーなどティストを考えておきます。何かしらの 2 次創作であればパロディとして、文体を変えたり、構成を整えるとパロディ的な側面を、より強調できます。また内容に合った表紙は、それだけでうっとりできます。これは間違いありません。自分で描いてもよいですが、なかなかそうもいかないので、知り合い伝いで探してみてください。

そしてイラストを描くには大変な労力が必要です。同人誌は商業ベースではないのですが、ご飯だったり、お金だったり、ありがとうの気持ちを伝えるのを忘れずに。

最後に、執筆内容について考える際、最も肝心な構成案についてです。構成の考え方、出し方は人それぞれだと思いますが、筆者の場合は、その時に旬な技術を取り扱うように心がけてます。SDK の新バージョンや、Firefox OS など時々で紹介したいな、と思う技術トピックを中心に構成します。これは読者視点も考慮した考え方です。

企画の原点は、仕事で困っていたことや不便に感じていること、気になっているんだけど調べられていない分野などテーマを決めて調べることがあります。自分が好きなことを面白く読んでもらうための切り口探し、というのが適切かもしれません。

もっとも、読んで面白いと感じる同人誌は、「ああ、ホントに好きなこと書いたんだろうなあ」と伝わってきますので、深く考えず、技術的興味や、好きなジャンルを選んでください。

バンドメンバー募集、当方ボーカル。

バンドメンバーがボーカルばかりになっても問題ありません。たいていの場合、1 人で書ききることが多い同人誌ですが、サークルも大小あり、友人や知り合いで集まって書くこともま

^{*2} もしあなたがエンジニアならわかるはず

あります。

筆者の場合、最大 35 名で書きました。最少は 5 人です。多いと書く以外の調整業もそれなりにあるので、はじめは仲のいい友人と一緒にとかいいかも。音楽性の違いによる解散も経験のうちです。

ひとりでやると、それはもうすべてが自由なので、とても気軽なのですが、それはそれでサークル参加の事務作業であったり、準備なども漏れてしまうことがあります。そわそわする日々になります。イベント参加に手慣れた人がそばにいれば、これ幸い、相談してください。たいてい知らないルールや手続きのほうが多いです。

さて、一人で書く場合は役割分担も必要ないので本節は読み飛ばしましょう。複数人で執筆する場合は、役割分担を決めておきます。実際、同人誌を作つて頒布しようと思うと意識する項目は結構、多いです（多分、性格にも依存する）。

- 運営と執筆

視点はさまざまですが、基本的には執筆者と編集者、サークル代表などの運営者、という側面があります。とくに執筆と編集は明確でないと、文章が良い悪いでモメちゃうかもしれません。何度かヒヤッとするようになりました。誰がどんな権限を持ってるかはどこかにまとめておきましょう。

よくあるのが他人の文章をどこまで編集していいのか、わからない、といった混乱です。自分の文章ならどんどん直すんですが、やっぱり人の文章だと遠慮しちゃいますよね。

また全体の構成を議論する前に個別の文章について議論することも混乱の元です。戦術（文章）をどれだけ頑張っても、戦略（構成）の失敗は取り返すことはできません。適切な粒度での議論を心がけたいのですが、議論になっている最中は、相手の意見に耳を傾け、最大限、尊重してください。

この場合、編集者（運営者）としての役割を誰かが持っているほうがスムーズです。特定の執筆者が兼任してもうまく機能すると思います。文章表現、国語は時として「絶対に正しい」みたいな価値観に当てはまりません。「ちょっと読みにくいんだけど」「僕はそう思わないよ」みたいな状況がほとんどです。誰の意見を優先したら良いか、という判断基準があるととてもスムーズです。

「Effective Android」のケースでは特定の一人が編集者として立ち回りましたが、それだけでは追いつかなかったです。相互レビューなどで文章の品質を上げて、結果を GitHub の Issue 管理で行っていました。

- 書く内容を固める

執筆を始める前に、ざっくりと頭の中に書く内容を固めておきましょう。この時点で筆が進まないようなら、新刊を落とす恐怖におびえることになります。幸い TechBooster では新刊を落としたことはありません。

具体的にはセクションのレベル 2 ぐらい、つまり章と節のタイトルとリード文があれば安心です。またプログラミングに関する技術書ならサンプルコードが先にあるとさらに余裕です。

技術的検証をやりながらの執筆は、笑えるぐらい遅い、というのが統計上わかっています^{*3}。

10.2 執筆 – しんちょくどーですかー？

肝心なのはスケジュール、つまり締切です。エンジニア諸兄におかれましても、実装が終わりではなく、検証とリリースまで終えてプロジェクト終了というのは周知の事実です。場合によっては保守、不具合対応という名前でリリース後まで引っ張られますね、おつかれさまです。しかし、安心してください。同人誌、商業誌の場合は印刷所に入稿したら実装に戻ることはありません！

執筆の締め切りは思ったより早く到来します。印刷所の入稿前日まで原稿を書くことは避けるべきです。ウッ、なぜか胃が痛くなってきました。

ここでは、執筆の心構えと、文章を読みやすくするためのテクニックを紹介します。基本的なお話をメインですが案外と、奥が深いものです。

文章を書くのに大切なこと

1 つの文章にたくさんの主張を入れると読む人は混乱してしまいます。ですので、なるべく 1 つのコンテキスト（章、節、項で主張の大小はあります）では、言いたいことを 1 つ主張するようにしてください。

読者はあなたの主張を読みたがっています。数ある同人誌のなかから偶然にも興味をもって読んでくれています。章のはじめには、導入を用意して話の流れを先に説明して、読者と記事のマッチングをはかります。リード文というかたちでもいいですし、本文の扱いでも問題ありません（本のスタイルとして統一していればよい）。

まとめでは主張の魅力を最大限伝えてください。そのあとこの本文を読んだ際の理解度が大

^{*3} N=1,mhidaka の経験則

幅に向かって広げます。

また文章を書いていると上下を意識して「以上が」「以下のとおりです」など書きたくなりますが組版の都合で上下が明らかでない場合があります。次のページにいってしまったりするわけですね。それぞれ「前述のとおり」「次のとおりです」で置き換えるとよいでしょう。

また図、表、サンプルコードについては、

- 登場前に内容を簡単に説明する一文を書く
- 登場後に詳しい解説を書く

を念頭に構成してください。読者が読んだときの唐突感や投げっぱなし感がなくなります。

文章を理解してもらうには？

ここでは文法的な、または文章として読みやすくする工夫を紹介します。ちょっとしたことなのですが、すべてを守ろうとおもうと意外に難しいものです。以下に3点のポイントに絞って、まとめました。

- 文章は短く、簡素に書く
- 初出の用語/概念は説明する
- 表現を統一する

長い文章は、主語、述語の対応が取りにくく（筆者の国語力もありますが）主張があいまいになります。経験的に技術文章で3行を超える文章（句点「。」で区切られていない長文）は読みづらく感じます。

まれに長くても面白い文章を書く人がいるのですが、一般的なノウハウというより、国語的な才能の場合があります。技術的な文章であれば、修飾的な表現は割り切って、簡潔に伝えましょう。

執筆者の文体にも依存しますが、意識的に読点「、」を増やすと読みやすくなります。

ひらがながおおめの文章なら読点をなるべくおおめにいれましょう。

ひらがなが、おおめの文章なら読点をなるべく、おおめにいれましょう。

極端な例ですが、商業出版されている技術書でも、同様の傾向がみられます。なるべく国語的な難しさを取り払い、わかりやすさを優先しています。

更に、書き進めているうちに用語解説を忘れることがあります。「文章を書く」という作業は、言い換えるならば、筆者の主張や考えをまとめ、という行為です。

そうして出てきた単語は、いわば筆者の中では、既知のものとなっており、説明不要と思い込んでしまうことがままあります。読者にとって、初めて出てきた単語は未知の物体 X であり、理解の妨げになることを十分留意して下さい。

最後の「表現の統一」は、小さいなことですが読みやすさに最も影響します。国語的なことではありますが筆者が主張を文章に落とし込んだあとは、その文章があなたの主張、意見そのものになります。読みやすさを気にしてしすぎる、ということはありません（もちろん著者の書くモチベーションや勢いも大事なので最後にチェックするといいでしょう）。

同じ概念、事象、構造、ブロック、クラス、メソッドなど、ものごとへの言及は、表現を統一して言い換えないようにしてください。

また商業出版物をよく見ていると、段落も細かくとる書籍が増えているようです。本項「文章を理解してもらうには？」でも段落を細かくとってみました。気付きましたか？ もともと mhidaka の書く文章は、段落が細かいため、読みやすさは感じなかったかもしれません。いずれにせよ、このあたりは流行がありますから、現在の主流として意識しておけばよいでしょう。

ちなみに、このような書き方の習熟は、才能+訓練です。商業紙への寄稿や出版の機会に恵まれると自然と身につきますが、なんにせよ自己分析や努力は必要です。楽しみながらできることが一番ですね。

10.3 編集 – しめきりそこですよ？

「先生！ 原稿まだですか？！」という編集のお仕事は、アニメ「サザエさん」の登場人物、伊佐坂先生に対するノリスケのポジションです。

ただ、本当に大事なことは記載された内容を試してみたり、読んでみたりして、わかりやすく表現を書き換えたり、不足している内容を書き足してもらうようお願いするなど、全体を整える作業こそ、編集者たるノリスケの手腕が問われる瞬間です。

とはいものの同人誌においては執筆者が兼ねる場合がほとんどです。現に「Effective Android」でも執筆と並行してレビュー・修正が行われていました。ことエンジニアの執筆は

趣味や副業的な側面もあるため、自然と時間が限られ、締切との戦いになります。

本節では、技術書として注意したらよいポイントをいくつか列挙していきます。粒度に差はありますが、重要な項目からの紹介です。

文中の文言、装飾について

これまでの同人活動を通じて、執筆者は装飾を使わないほうがよい、と感じています。装飾には、いわゆる太字、斜体などフォントの見た目を変えて読みやすくする意図があります。しかし、利用基準を定めて守るほうがよっぽど難しく、一貫性を保つのが作業量的にしんどい側面もあります。

大事な内容であれば、文章中で注意喚起したほうがよっぽど良いでしょう、という個人的主張のもと、装飾が必要なときは編集担当者がチェックできる基準でのみ実施すべきです。

文章中の記号は全角が基本（特にカッコ）

こちらも見た目に影響する話です。半角記号は基本的に英字に合わせてフォントが作られています。文章中で表現として使う場合、記号は全角を利用して下さい。メソッド名、プログラムのソースコードからの引用はその限りではありません。

文章中の英字で空きは詰める

英単語を文章中に入れる際に次のような書き方があります。

2014年1月17日にEffective Androidは、インプレスさんから発売されます！

こちらは多分Webの文化か何かだとおもうのですが、紙の書籍では無意味な半角スペースの混入扱いされます。やっぱり見栄えが良くないためですね。

2014年1月17日にEffective Androidは、インプレスさんから発売されます！

編集段階で取るのは結構大変です。事前にルールとして共有したいところですね。ちなみに全角スペースであっても NG です。

漢字の開きを統一する

漢字の開き方、は聞きなれない用語かもしれません。漢字をつかうべきか（閉じる）、ひらがなで書くべきか（漢字を開く）、という基準が一定、存在しています。出版社および編集部ごとに細かい作法は違うため、同人誌を書くにあたって一般的な用法を紹介します（表 10.1、表 10.2）。

表 10.1 推奨する漢字の開き方

推奨の表現	非推奨の表現
こと	～する事、した事がある
とき	～した時、～の時、
できる	～出来る
ただし	但し
したがって	従って
のように	～の様に
ない	～が無い
いえる	言える、言う
ひとこと	一言
最も	もっとも
さまざま	様々な表現がある
かかわらず	関わらず
むやみ	無闇
ほうが	方(ホウ)が
ほうが良い	ほうがいい
大掛かり	おおがかり
隠ぺい	隠蔽
さらに	更に
持つ	もつ

経験的に技術書で多い表現は「～出来る」「～様に」「～する事です」などです。このあたりはどの出版社も、すべて漢字を開いてくる方針で運用していますので、同人誌を作る際も合わせたほうが、よいでしょう。特に電子書籍や商業出版を次の展開として経験してみると「最初からやつといたほうがよかったでござる！」感がものすごくありました。

表 10.2 推奨する漢字の開き方(続)

とおり	通り
基づい	基い
見ていく	みていく
使う	つかう
勧め	すすめ、薦め
他	ほか
すでに	既に
付き	つき
分かる	わかる
気をつけ	気を付け
気づく	気付く
すべて	全て
たとえ	例え、例えば
のち	後(ノチ)
たくさん	沢山
中	～のなか、なか
特に	とくに
きれい	綺麗
たいてい	大抵
すぎない	過ぎない、過ぎる
振る舞い	振舞い、ふるまい
相性が良い	相性がよい

文章表現に関するエトセトラ

文体の統一

文体は著者の味となるため、過剰な編集は好みませんが、それでも以下のような文章は編集段階で変更しています

- ネガティブな表現を利用しない
- 体言止め、話し言葉は利用しない
- 「ですます」調と「である」調を混在しない

ネガティブな表現は読者的心証を悪くします。気持ちの問題なのですが、期待して本を読んでくれている読者にデメリットだけでなくメリットも感じてもらえるように表現をポジティブに改めます。ネガティブな表現が同人誌そのものに感染して、本をネガティブに捉えられても、もったいないですからね。

体言止め、話し言葉については、技術書であれば平易な表現を心がけて読者の理解に努める方針のもと、著者の文体を壊さない程度に調整しています。

「ですます」調と「である」調の直接の混在はあまりありませんが、著者ごとに癖はあるものです。読んでいてリズムを崩しそうなら修正することがあります。「Effective Android」では「～だと思います」という文章であれば「です」で編集したり、「～することができます」という表現であれば簡潔に「～できます」と縮めています。これも気が付いた範囲でしか実施できないですが、編集時には想定読者になりきって読みやすい文章を追及しましょう。

サンプルコードの統一

意外に忘れるのがサンプルコードのフォーマットチェックです。紙面においてはサンプルコード（リスト）も本文の一部として扱います。小さな工夫としては「Effective Android」では紙面を節約するため、タブインデントを 2 スペースとしています。

表紙、目次、画像のチェック

文章以外での確認項目は次の通りです。

- 書籍の「はじめに」など執筆者の作業でないところを作る
- 図表の参照が本文中にあるか確認する
- 本文、図表が印刷範囲からはみ出でないか確認する

基本的にすべてのページに目を通し、印刷にあたって問題となる個所をすべて修正してまいります。最終入稿に近い形で検証するため、フォーマットは PDF です。ちなみに商業出版だと最後は紙に印刷して確認します。同人誌（技術書）ではデジタル入稿が一般的になっているため、PDF でチェックします。

「本文、図表が印刷範囲からはみ出でないか確認する」という項目は、折り返しが難しく、自動で対応できない箇所をさします。技術書でよくあるミスとしては長いメソッド名や大きなテーブルによる紙面からのはみ出します。ReVIEW のコンパイル時ログと合わせて検証していきましょう（Too wide... と警告が出ているはずです）。

- 著者紹介を作る
- 目次を見て構成や名前を調整する
- ノンブル（ページ番号）が全頁入ってるか確認

このあたりは調整に時間がかかるものがほとんどです。時間をたっぷり用意しておきたいですね（用意できたためしはありませんが）。「はじめに」と目次は読者が最初に目に入る部分です。1 ページほどでわかりやすい構成を心掛けるといいででしょう。

10.4 入稿と頒布

入稿 ー よろしくおねがいしまあああす！！

入稿とは印刷所へ最終的な印刷用データを渡すことです。印刷方法はオンデマンドやオフセットなどさまざまな方法があります。紙の種類も多数あるため詳しい内容については印刷所ごとに、きちんと相談することをおすすめします。ちなみに TechBooster では日光企画さんに依頼しています。

同人誌においては基本的に新刊が最も売れます。在庫は既刊と呼ばれ、新刊ほどの勢いはありません。鉄則は頒布する必要最低限だけ印刷して在庫を持たないことです。サークルごと部数の指標は違いますが、TechBooster がコミックマーケット 84 に参加したときは新刊 2 冊で 450 部を用意しました。

オフセット印刷の場合、本文は PDF データでデジタル入稿します。表紙のみ指定フォーマットに合わせて用意しますが、基本的にイラストを用意したら、表紙用の装丁をおこない、入稿用データを作成します（図 10.1、図 10.2）。

--[[path = (not exist)]]--

表紙例

入稿には、表紙 1（表）と表紙 4（裏）に本の厚みも考慮した背を足したサイズが必要です。断ち切りのための遊びがどれくらい必要か、なども印刷所に確認しておきます。

```
--[[path = (not exist)]]--
```

入稿データ

頒布－新刊でました！

もし頒布イベントがコミックマーケットであれば、新刊は印刷所からの直接搬入が主流です。サークル参加者でも当日に初めて手に取ることになるわけです。ドキドキですね。新刊が入った段ボールの中には、落丁や乱丁に備えて、いくらか余分に冊数がはいっていることがあります（余部）。

経験上、ポスター や テーブルクロス、頒布価格を書いたポップがあると華やかです。頒布価格は 1 ページ 10 円で総ページ数から決めることが多く、さらにそこから切りのよい数字（100 円単位など）で合わせます。TechBooster では厚さのあまり、主力価格帯は 1000 円になってしましました…。ワンコイン（500 円）同人誌も需要が高く、部数が伸びる傾向にあります。またサークル配置は基本的に同じジャンルで集まっています。あなたが欲しいと思う同人誌がそばにあるかもしれません。

頒布の段階では、ReVIEW の特徴であるワンソース、マルチプラットフォームを活かして電子書籍を付録につける、Amazon や Google Play での販売も視野に入ります。ReVIEW を使っている段階で、電子書籍の制作ハードルは非常に低く、再編集の手間が大幅に軽減されます。

創作活動によせて－先生の次回作にご期待ください

本章では TechBooster での経験をベースに同人誌活動を紹介しました。

開発者による技術書作成を推奨したい、と思ったきっかけは、商業出版などの制作リードタイムの問題です。近年、インターネットの普及に伴って商業的な技術誌は発行部数を下げ続けています。これは鮮度の問題で、ニュース性が高い話題はネットメディアで集めるようになってしまったためです。

さらに商業出版される技術書籍についても、制作期間の長さから読者の手に届くときには、価値の低下が始まってしまいます。書籍の良さのひとつに体系的な技術、情報の蓄積がありますが、技術の陳腐化（というより開発環境やバージョンの更新）が早いわけです。このような

時代の移り変わりに、対抗する手段として開発者による情報発信、創作活動があると考えています。

本章の内容は数ある同人サークルのひとつのケースですが、技術書を作るために ReVIEW を使い倒しています。得られたノウハウや知見が、読者へのインスピレーションになれば幸いです。では、次回のコミケでお会いしましょう！

第11章

同人誌的 ReVIEW 関連用語開発

主に筆者らによる偏見です。

Bundler

「ばんどらー」と読む。プロジェクトで利用する RubyGem から拾ってくるライブラリのバージョンを指定して利用できる機能を持つ。第12章の章で言及があるが、ReVIEWを使う時は gem 版を使うか master/HEAD を使うか、という話題がある。とりあえず、同じ本を書いている人の間では同じバージョンの ReVIEW 使ったほうがいいよね、ということでわかめが布教している。だがしかし、わかめ含めみんな Ruby 文化圏の人間ではないのでいまいち流行ってない。もういっその事、Rakefile 中で Bundler のインストールまで勝手にやってしまうべきか… さすがにそれはみんなびっくりするかな…という葛藤が生じていると俺の中で話題沸騰中。

CSS

「しーえすえす」と読む。こいつがプログラマブルじゃないせいでどれだけの血と汗と涙が流れたのか。イマドキは SCSS とか LESS を使う（気がする）

Emacs とか Vim とか

「いーまっくすとかびむとか」と読む。両方共 UNIX 系 OS だと特に良く見る。

EPUB

「いーぱぶ」と読む。HTML を雑に zip で固めた物体だと思っているけど雑なのはお前の理解だー！ ふはははは。

eRuby (erb ファイル絡み)

「いーるびい」と読む。embedded Ruby の略。拡張子は .erb。主に HTML へ Ruby スクリプトを埋め込む技術。ReVIEW でも HTML や EPUB の出力時に使う。Ruby スクリプトを埋め込む技術。Ruby スクリプトを埋め込む技術。繰り返す。「Ruby」スクリプトを埋め込む技術。

format.rdoc

「ふおーまっとどっとあーるどっく」と読む。わかめのブラウザでは、アドレスバーに `format` と入力すると自動的に <https://github.com/kmuto/review/blob/master/doc/format.rdoc> が第一位にサジェストされる程度によく利用される。

git

「ぎっと」と読む。git が使えないエンジニアは声が出せないに等しい。しかしあるエンジニアの外に出てデザイナさんに普及しようとしてもオシャレ波とアンチ KUROIGAMEN 勢力によってメガメガしてしまうのだ。かなしい。

HTML

「へてむる」と読まない。これについての解説がないな！ と思ったけど、解説いらないだろこれ(投棄)。

InDesign XML

「いんでざいんえっくすえむえる」と読む。InDesign という、出版界ではデファクトスタンダードとして使われている Adobe 社の DTP ツールのための入力形式。しかし所詮は XML だ。

master を使え！

「ますたーをつかえ！」と読む。gem 版にある ReVIEW を使うとハマる場合が多いので GitHub の master を隨時使うべき！ という主張。広く使ってほしいツールのくせに gem 版がおすすめできないとは何事だ！ と主張するわかめとは相反するためまた喧嘩になる。なかよくしろ。あめ玉は「そこまで ReVIEW のリリースに期待はできないだろ」とこの本の趣旨を台無しにする発言を度々している。

PDF

「ぴーでいーえふ」と読む。Adobe 社が策定したフォーマットのくせに、至る所で使われ、版管理も難しいくせに ECMAScript の規格書も PDF 版が正式だしこんなの絶対おかしいよ！ 全世界の仕様書が ReVIEW で書かれるようになって簡単に git で diff が取れるようになーあれ♪ ISO 32000-1 として規格化されているそうな。規格書開いて Effective Android (512P) より多かったのでそっ閉じ。

pull request

「ぱるりくえすと」と読む。主に GitHub 上で行われる文通のようなもので、お手紙を書く代わりにコードを送りつける。バグの原因を説明して直してもらうよりバグ直したコード送りつけたほうが早い！ という発想に基づく省力コミュニケーションの次世代形。

Rakefile

「れいくふあいる」と読む。Makefile の Ruby 版。わかめ（第 9 章の筆者）は Rakefile はよくわからないので Gruntfile.js のほうがいいよ！ といってぶつくさいっている。

review-preproc

「れびゅーぱりぱろっく」と読む。外部コマンドを実行して、その出力を .re に埋め込む、みたいなことができる。第 6 章の執筆者である @amedama さんは、僕らのために ReVIEW 原稿 CI サーバを作ってくれています。で、原稿執筆の途中で review-preproc の存在を知って、「このコマンド使えるようにしたら CI サーバで何でも好きなコマンド実行できるじゃないですか！ やだーー！！」と叫び始めて大変可哀想でした。でも、review-ext.rb もあるし、どのみち制限かけるのは無理だと思～うよ☆ 新婚さんに慈悲はない！ CI サーバ殺すべし！ イヤーッ！

Sublime Text 2

「さぶらいむてきすとつー」と読む。Emacs とか Vim とか怖くて使えないよ！ という現代っ子のための、ReVIEW 原稿ライティングツール。yanzm 先生ありがとう！

TeX

「てふ」もしくは「てっく」と読む。TechBooster では割と「てふ」だがむとーしんは「てっく」だった気がする。純粹な `tex` コマンドを使ったことがあるものだけが、この項目を解説しなさい。

TeX Live

「てふらいぶ」と読む。TeX が関わる日本語書籍執筆なら今はこれ。ただしストレージを専有する領域も神レベル。具体的にいうと、Windows 95 のインストールがフロッピー 21 枚だったらしいので、1.44MB として計算すると Mac 用の TeX Live である MacTeX の配布容量は実に 76 Windows 95 に達する。GUI を備えた Operating System 76 個分の容量って、一体何が入っているんだ。希望か、絶望か。

YAML

「やむる」と読む。プログラムの中のデータを保存したりプログラムに取り込んだりするのに便利。XML のタグタグしたところに疲れ果てた人の一部の間で XML の軽量化の機運が高まり、そこに Perl のシリアルライズフォーマットが合体して生まれた謎の記法（後日 JSON とも合体した）。ReVIEW では書籍のファイル名や執筆者やらの情報を保存している。詳細は第 5 章を参照。

この章一体何なんだ

カオスって魅力だと思うんだ。

コミックマーケット

盆休みと年末を差す

宗教上の理由で使えない

読んで字の如く、宗教上の理由で使えない場合に言い訳として用いる。あるプログラムやデータ構造を minify してワンライナー形式に変換できない仕組みや構造であるものには可能な限りかかわらずに生きていきたい。CoffeeScript ! ? Python ! ? YAML ! ? グワーッ ! 宗教上の理由でーッ !!

進捗どうですか？

昨今の進捗聞かれっぷりは異常である。TechBooster 界隈では、Unicode Emoji の HAPPY PERSON RAISING ONE HAND (U+1F64B) の絵文字が、とあるチャットクライアントで「＼進捗どうですかー？／」と聞いているポーズに見えることから、進捗さんと呼ばれて親しまれている。たった 1 文字で原稿が進んでいない相手を煽るので非常に便利だが、スナック感覚で煽り合いに発展するので原稿を早めに勧めて相手より優位に立たねばならない。

むとーしん

@kmuto さんのこと。武藤神。なんで神と呼ばれているかは実はわかぬはわかっていない。みんなは果たして知っているのだろうか。編集注釈：ムトウ神が正しい。Debian メーリングリストにおいて初心者対応の丁寧さが神のごとくであると 2 ちゃんねる民の間で話題となったことが起源。なお某 Android の会メーリングリストで同じ現象が発生する可能性は高くないことをここに書き添えておく。

猫

かわいい。

第12章

トラブル集

12.1 review-init が RubyGems 版で壊れているとか古いとか

本著執筆時に筆者間で議論をしていたのですが、review-init に話が及んだ際、そもそも動作する環境としない環境があるということで議論が紛糾しました。

その時点の RubyGems 版（1.1.0）では review-init が壊れているということが分かりました。

RubyGems 版は開発者の方が時間があるときに git の変更を送るそうですが、しばしば忙しいということでリリースが遅れるようです。

その関係もあって RubyGems 版がしばしば最新の変化を取り落としているので、git の最新版をなるべく利用するべき、という意見と、バージョンは本来作業中には固定であるべきで、unstable の最新を追うべきではないのだから、変更するよう頼み続けるべきだ、という議論がありました。

読者が ReVIEW を利用する場合には、いずれにしても腹を括らねばなりません。

RubyGems 版を利用しているならば最初にバージョンを確認しておきましょう。なにかのトラブルに巻き込まれた時のため、あるいは巻き込まれた後でも差し支えありません。

```
> gem list review
```

大事なのはトラブルを解決しようという気力です^{*1}。

^{*1} どうにもならない場合、Twitter で@mhidaka にメンションを送って相談すると解決の糸口が見つかるかもしれません

12.2 「====タイトル」と原稿に書いたら ReVIEW のコンパイルが止まらなくなった

リスト 12.1: コンパイルが止まらなくなった例

```
====Test
```

お分かり頂けるだろうか……？^{*2}

世の中の書籍執筆の例に漏れず TechBooster でも「継続的インテグレーション」的なサーバというのはありますて、そこでコミットがある度にビルドをしていたのですが、あるときからビルドが終了しない事案が発生。

見てみたらサーバのいくつかのプロセスの CPU 利用率が 100% に……

直接の原因は「====」と「Test」の間に半角スペースがないことなのですが、根本原因是 ReVIEW の正規表現に難がありました。このくらいなら受け付けてくれるか、少なくとも文法エラーで止まるかして欲しいところですが、がっつり無限ループとあいなりました。

現在の git 版最新版では直っています。この一つ前の節も見ましょう。

12.3 TeX Live をインストールしたのにフォントがない！

特に Debian/Ubuntu でありがちなミスという気がします。

第 2 章で紹介されている方法以外に、Debian/Ubuntu では apt-get コマンドで TeX Live をインストールする方法があり、そちらのほうが手頃です。ただ、パッケージが複数に分かれているため、自分の環境に適したパッケージを選ぶ必要があります。

Debian/Ubuntu 系であれば texlive-lang-cjk や texlive-fonts-recommended が入っていることを確認してください。

TechBooster の関連書籍を筆者の Debian wheezy 7.1 の環境下でビルドする際に入っている TeX Live 関連のパッケージを念の為添付しますので、ヒントになれば幸いです。

^{*2} <https://github.com/kmuto/review/issues/213>

```
> dpkg -l | grep texlive | awk '{print $2}'  
texlive texlive-base texlive-binaries texlive-common texlive-doc-base  
texlive-doc-zh texlive-extra-utils texlive-font-utils texlive-fonts-recommended  
texlive-fonts-recommended-doc texlive-generic-recommended texlive-lang-cjk  
texlive-latex-base texlive-latex-base-doc texlive-latex-extra  
texlive-latex-extra-doc texlive-latex-recommended texlive-latex-recommended-doc  
texlive-luatex texlive-pictures texlive-pictures-doc texlive-pstricks  
texlive-pstricks-doc
```

TeX Live 本家を利用してインストールする場合にフォントがない、という問題を聞くことはほぼありません。

この辺りのチョイスは良し悪しです。TeX Live を本家からインストールすると数 GB のストレージを食いますし、アップデートは OS ディストリビューションとは独立に行うことになります。

12.4 半角カナが全角カナに！

『Effective Android』の同人誌版から電子書籍版にアップデートする過程で、文章中の半角カナが勝手に全角カナに直ってしまうという事態が発見されました^{*3}。

根本的な原因は TeX が標準で半角カナを利用できないことにあったのですが、ある種の拡張に頼ることを前提として ReVIEW のレイヤでうまく処理してもらうようになりました。TeX はネイティブで UTF-8 に対応していないので、これ以外にも種々の問題があるようです。

ReVIEW のレイヤからすると、下位の問題を個別にうまく対応する必要があるわけですから、大変は大変ですね。

12.5 せんせい、打ち消し線を使いたいです

要は HTML で言う のことです。^{*4}

ReVIEW では出力先毎に対応を変更「可能」と言いつつ、全て実装されているわけではありません。del 命令もその一つでした。

そもそも TeX では標準で打ち消し線を行う実装が存在しないため、外部マクロを用いるこ

^{*3} <https://github.com/kmuto/review/issues/182>

^{*4} <https://github.com/kmuto/review/issues/178>

と、外部マクロでも複数選択肢があること、といった理由から、ReVIEW の PDF 出力でも打ち消し線の標準的なサポートはないようです。

TechBooster では review-ext.rb に以下のようなコードを記述することで対応しました。

リスト 12.2: TeXにおいて打ち消し線を用いる一例

```
module ReVIEW
  class LATEXBuilder
    def inline_del(str)
      macro('sout', escape(str))
    end
  end
end
```

12.6 あるはずのファイルがないと言う ReVIEW

あるはずのファイルがない時に発生するトラブルがないはずなのにあります^{*5}。

章構成を作る CHAPS について第 3 章で紹介しました。

復習すると「review-pdfmaker や review-epubmaker を使う際には YAML ファイルを作る必要があり、ついでに CHAPS も作る必要がある」という内容でした。

この際、複数の章を別のファイルで作るため、first-chapter.re の他に second-chapter.re を作成して文章を書き始める例を書いていたときのことです。

```
> ls
CHAPS first-chapter.re sample.yaml second-chapter.re
> cat second-chapter.re
= ReVIEW コマンド解説
> review-compile --target html second-chapter.re
review-compile: error: no such file: second-chapter.re
```

ああっと！

調べてみると、CHAPS ファイルがそのディレクトリに収録されているその瞬間から発生するようです。修正方法は「CHAPS にそのファイル名を加える」です。

*5 <https://github.com/kmuto/review/issues/214>

```
> cat CHAPS
first-chapter.re
> echo "second-chapter.re" >> CHAPS
> cat CHAPS
first-chapter.re
second-chapter.re
> review-compile --target html second-chapter.re
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:ops="http://www.idpf.org/2007/ops"
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <meta http-equiv="Content-Style-Type" content="text/css" />
  <meta name="generator" content="ReVIEW" />
  <title>ReVIEW コマンド解説</title>
</head>
<body>
<h1><a id="h2"></a>第 2 章 ReVIEW コマンド解説</h1>
</body>
</html>
```

というわけで、一度 CHAPS を作ったら、以降 ReVIEW ファイルの名前は忘れず追加しておきましょう……という話なんですが、それにしても「ファイルがない」はミスリーディングだなあと。

CHAPS ファイルがある場合、review-compile は同一ディレクトリ内の他の ReVIEW を見て章番号や他の章への参照をうまく処理しようとします。よく見ると、上で出力された HTML の中で「第 2 章」と正しく章番号が設定されていますね。

ただ、執筆時点ではこの部分の実装にバグがあったようで、CHAPS に記述されていないファイルに対しては、何故か、そのファイルがディレクトリにも存在しないかのようなエラーを表示していました。

自分で直す (Pull Request) 余裕もなかったため、最低限レポートをしたところ、翌日には直りました。今は"No such chapter in your book. Check if the catalog files contain the chapter."と出るはずです。今度は「catalog files って何」って思いますが、要は CHAPS, PREDEF, POSTDEF, PART のことです。

12.7 ReVIEW で自分自身は記述できないの？

本書の多くの章で ReVIEW の構文紹介があったかと思います。

しかし ReVIEW には基本的に自分自身の構文を無視する機能がありません。紹介しようとすると、それ自体が修飾するべき構文とみなされて読者の読む出力からは消滅してしまうからです。

対策としていくつも案が挙がりましたが、決定打は今のところありません。

- 全角@使う
- //list 記法の中では先頭にスペースを入れる
- 構文解釈を失敗しそうなポイントに\をピンポイントで差し込む
- ReVIEW の処理系を書き換えてしまう

12.8 みんなで使おう！

より重要なのは、ここで記載されている事項のほとんどは、数日以内に修正されてたりすることなんですね。

第 1 章の課題でも書かれているかと思いますが、「ユーザ数が少ない」ことがダイレクトに本章のトラブルに結びついている、ということが圧倒的に多いと言えます。そして、開発者の対応自体は大変早いので、結果として致命的な問題は起こらず現状でも執筆出来ています。

というわけで本章を読んだ後でも結論はあまり変わらず「みなさん ReVIEW 使いましょう」ということになります。

なるよね。

はじめての ReVIEW

2013 年 12 月 31 日 初版発行 v1.0.0

著 者 TechBooster

デザイン yuya

編 集 mhidaka

発行所 mhidaka

(C) 2013 techbooster.org