

技術書のための：

Pythonで

Markdownから

EPUBをつくる本

py(md) => epub: Advanced



Masakazu Yanai

技術書を作るための：  
Pythonで  
Markdownから  
EPUBをつくる本

---

py(md) -> epub: Advanced

## 目次

前書き	7
挨拶	7
本書の技術的な内容	8
スクリーンショット	9
第1章 EPUBファイル	10
出力するEPUBファイルの構造	10
固定ファイルの内容	12
item/content.opfの内容	13
第2章 EPUB、KPFの注意事項	15
XHTMLのタグ	15
style属性	15
使用可能なタグとスタイル	16
等幅フォント	18
スペースの幅	18
実機での確認	20
第3章 EPUBのCSS	21
縦書き	21
基本のCSSファイル	23
第4章 原稿のファイル構成	26
原稿のファイル構成	26
テンプレート ファイル	27
原稿ファイル	28
第5章 config.yaml	30
YAMLについて	30
config.yamlの全体	31
出力先ディレクトリー、出力名	34
カバー画像	34
複製ディレクトリ、追加複製ディレクトリ/ファイル	35
テンプレートHTML名	36
電書情報	36
ユニークID	37
自動生成目次用	38
XHTML化するファイル	39
第6章 開発の準備	40

VSCodeの導入	40
Pythonの導入	40
VSCodeのPython拡張の導入	41
利用するPython外部パッケージの導入	42
JavaとEPUBCheckの導入	43
電子書籍確認用ソフトの導入	45
第7章 プログラムの構成	47
プログラムのファイル構成	47
ターミナルについて	48
VSCodeのデバッグ機能	49
第8章 プログラムの開始	52
プログラムの準備	52
コマンドの実行	54
プログラムの開始	55
第9章 設定の読み込み	57
ファイル操作の道具	57
設定の読み込み	59
第10章 ファイル複製	64
処理の流れ	64
基本データの複製	65
カバー画像関連の構築	67
ディレクトリー/ファイル複製	69
第11章 Markdown変換	71
原稿ページの構築	71
Markdown変換の全体	74
Markdown変換の初期化	75
Markdown変換の実行	77
第12章 本文構築 周辺処理	79
目次用タイトルの取得	79
拡張置換	82
変換途中のMarkdownを出力	83
第13章 目次構築	85
toc.xhtmlの構築	85
content.opfの構築	87
確認用toc情報の作成	93

第14章 圧縮してEPUB生成	95
圧縮してEPUB生成	95
作成したEPUBの確認処理	100
第15章 拡張置換について	102
拡張置換について	102
プログラムの読み込みや装飾	104
置換の利用	105
関数の利用	106
第16章 拡張置換用のファイル	107
拡張置換用の設定ファイル	107
拡張置換用のCSSファイル	108
第17章 拡張置換の処理	111
拡張置換の入り口	111
正規表現の置換	112
関数の置換	113
サブセット フォント作成	114
第18章 コードの変換	118
コードの全体	118
インポート部分	122
事前置換と事後置換	122
置換	122
テキスト ファイル読み込み	123
コード ブロック置換	124
コード置換	125
コード ライン置換	125
後書き	127
宣伝	128
奥付	130

## コード部分を、等幅のコード用フォントで表示する方法

---

この本をKindleで読む場合は、フォントの設定を「**Publisher Font**」に変えてください。デフォルトでは「明朝」になっています。

## サンプルファイル

---

本書のZIP版ではサンプルファイルを同梱しています。EPUBなどで入手した人は、後書きのところでサンプルファイルをダウンロードできるようにしています。

## トラブルシューティング

---

この本の表示などで何か問題があった場合は、著者にお知らせください。原因が分かれば、次の版で反映したいと思います。

## 前書き

### 挨拶

2025年の2月に『PythonでMarkdownからEPUBをつくろう』という電子書籍を書きました。初級者向けに、Pythonの解説も入れながらEPUBファイルを出力するまでを解説した本です。

この本を書いている時に、上級者向けの本も書こうと思っていました。なぜならば、ソースコードを掲載した電子書籍を作るには、もう少し細々としたテクニックが必要だからです。そうした細かな部分は、初級者には難易度が高いと考えてカットしました。

今回の本は、そうした技術書に特化した部分を多く入れて、初心者向けのPythonの解説を削った内容にしています。

世の中には、テキスト原稿（Markdown原稿）から電子書籍を出力するシステムがいくつかあります。こうしたものを自分で作るのは楽しいです。また細かなカスタマイズができるという利点があります。中身を知っていれば、さまざまな自動化にも応用できます。

私の場合は、Amazon向けにEPUBを出力したり、同じ原稿からPDFファイルを出力したりしています。

この本ではPythonを使い、リフロー型の電子書籍を出力するプログラムを開発していきます。本書はプログラマー向けに、コードを中心に解説していきます。この本自身も、本書で説明するプログラムで、テキストファイルから生成しています。

2025年5月 柳井政和

## 本書の技術的な内容

この本は、PythonでMarkdownファイルからEPUBファイルを作るプログラミングの本です。

### 想定している使い方

想定している使い方は、次の通りです。

- **Markdown記法（拡張子 `.md`）のテキストファイルで原稿を書く**
  - 記号を使い、見出しや強調表示、リスト作成や画像配置などをおこないます。
  - 章ごとにファイルを分けます。
- **YAML形式（拡張子 `.yaml`）で設定を書く**
  - 出力するEPUBの設定やパスを書きます。
  - 章ごとに分けたファイルの名前を設定に書きます。
- **Pythonのプログラムで、原稿をEPUBファイル（拡張子 `.epub`）に変換**
  - 原稿を全てXHTMLに変換します。
  - 目次を自動生成します。
  - EPUファイルとして1つにまとめます。
- **Kindle用にKPFファイル（拡張子 `.kpf`）に変換**
  - Kindle Previewerのウィンドウに、EPUBファイルをドロップします。
  - Kindle Previewerで見た目を確認したあと、メニューからエクスポートします。
  - KPFファイルが得られます。

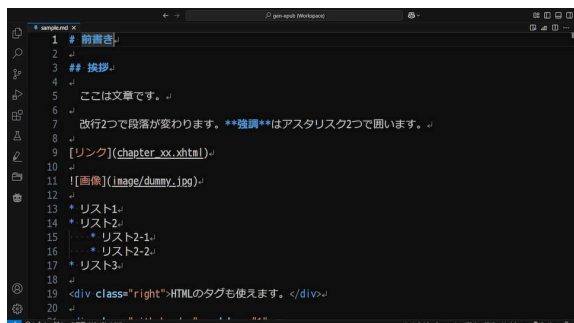
設定のYAMLファイルは、テンプレートのいくつかの場所（書籍名や著者名、各章のファイル名など）を書き換えて使用します。



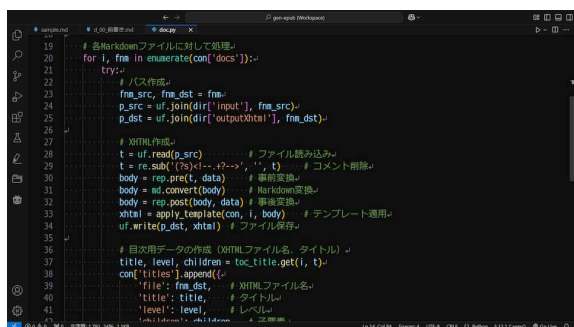
# スクリーンショット

この本で書いた原稿やプログラム、出力したファイルのスクリーンショットです。本の雰囲気把握するために掲載しておきます。

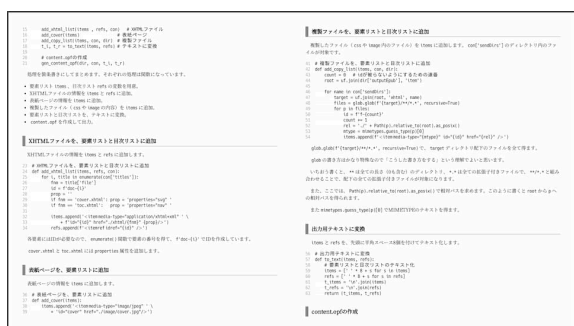
## ▶ Markdownの原稿



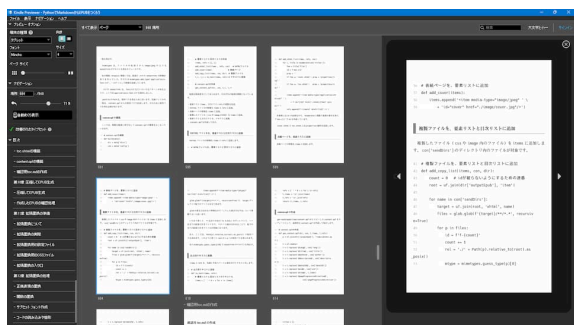
## ▶ プログラムの一部



## ▶ 生成したEPUBを、EPUBビューワで閲覧



## ▶ 生成したEPUBを、Kindle Previewerで閲覧



# 第1章 EPUBファイル

## 章の説明

ここでは作成するEPUBファイルについて説明します。仕様を全て把握するのは大変なので、実用の範囲内で収めます。この章では次の話をします。

- 出力するEPUBファイルの構造
- 固定ファイルの内容
- item/content.opfの内容

## 出力するEPUBファイルの構造

出力するEPUBファイルの構造を示します。作り方は人によって異なります。ここでは、この本のプログラムで出力する構成を掲載します。

### ▶ 出力するEPUBのファイル構造

- epub/
  - item/
    - image/
      - cover.jpg …… 表紙画像
    - xhtml/ …… 原稿の出力先
      - css/
        - main.css …… 主要なCSSファイル
        - add.css …… 書籍固有の追加CSSファイル
        - code.css …… コード表示に特化したCSSファイル
      - image/
        - 画像ファイル
      - cover.xhtml …… 表紙ページ
      - toc.xhtml …… 目次ページ
      - d\_00.xhtml …… 原稿ページ
      - ⋮
      - d\_99.xhtml …… 原稿ページ
    - content.opf …… 収録内容情報
  - META-INF/

- `container.xml` …… メタ情報
- `mimetype` …… ファイル情報

`mimetype` と `META-INF/container.xml` は固定です。プログラムで書き換えることはありません。

プログラムで生成するのは `epub/item/xhtml` の中身です。また、`content.opf` も生成します。

EPUBファイルを作る際は、このような構造のディレクトリーを作ったあと、`epub/` の中身をZIPで圧縮します。

その際、注意する点が2つあります。1つは、ZIPの先頭に `mimetype` を無圧縮で格納することです。他のファイルは圧縮して格納します（圧縮しなくてもよいですが、ふつうは圧縮します）。

もう1つは、`epub` ディレクトリーを圧縮するのではなく、`epub` ディレクトリーの中身だけをZIPに格納することです。

間違ったZIPの中身と、正しいZIPの中身の例を示します。ZIP内に `epub` ディレクトリーがないことが分かると思います。

#### • 間違ったZIPの中身

- `epub/mimetype`
- `epub/META-INF/container.xml`
- `epub/item/content.opf`
- `epub/item/image/cover.jpg`
- `epub/item/image/xhtml/css/main.css`
- ……

#### • 正しいZIPの中身

- `mimetype`
- `META-INF/container.xml`
- `item/content.opf`
- `item/image/cover.jpg`
- `item/image/xhtml/css/main.css`
- ……

この2つの注意点があるため、通常の圧縮ソフトで単純にZIP化しても、正しいEPUBファイルにはなりません。プログラムを使ってZIPにするか、少し面倒な手順を踏む必要があります。

## 固定ファイルの内容

いくつかの固定ファイルについて中身を見ていきます。

### mimetype

まずは `mimetype` です。EPUBファイルの先頭に無圧縮で格納します。中身は次のテキストです。

#### ▶ mimetype

```
1 application/epub+zip
```

MIMETYPEは、マimeTypeと読みます。MIMEは「Multipurpose Internet Mail Extensions」の略で、メディア種別とも呼ばれます。MIMETYPEは、プログラムに対してどのようなファイルなのかを伝えるためのものです。

MIMETYPEは、「タイプ/サブタイプ」の書式で書きます。ここでは、`application`（アプリケーション）であり、`epub+zip`であることを示しています。

この情報をファイルの先頭に付けることで、EPUBリーダーがファイルを読み込んだ際に、ZIP形式で圧縮されたEPUBファイルであることを認識できます。

### META-INF/container.xml

次は `META-INF/container.xml` です。

#### ▶ META-INF/container.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <container xmlns="urn:oasis:names:tc:opendocument:xmlns:container" version="1.0">
3 <rootfiles>
4 <rootfile full-path="item/content.opf" media-type="application/oebps-package+xml"/>
5 </rootfiles>
6 </container>
```

`<rootfile>` タグの `full-path` 属性を見てください。 `item/content.opf` と書いてあります。このパスがEPUBのルートになります。

EPUBリーダーはファイルを読み込んだあと、 `item/content.opf` を読みに行きます。

## item/content.opfの内容

次は item/content.opf です。META-INF/container.xml で示したルート ファイルです。

OPFは、Open Package Formatの略です。中身はテキスト ファイル（XML形式）です。このファイルはボリュームが大きいので、ある程度省略して掲載します。

### ▶ item/content.opf

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <package xmlns="http://www.idpf.org/2007/opf" version="3.0"
3   xml:lang="ja" unique-identifier="unique-id"
4   prefix="rendition: http://www.idpf.org/vocab/rendition/# ebpaj: http://www.ebpaj.jp/"
">
5   <metadata xmlns:dc="http://purl.org/dc/elements/1.1/">
6     <dc:title id="title">本のタイトル</dc:title>
7     <dc:creator id="creator01">著者名</dc:creator>
8     <dc:language>ja</dc:language>
9     <dc:description>本の説明</dc:description>
10    <meta property="dcterms:modified">2025-01-31T00:00:00Z</meta>
11    <meta content="cover" name="cover" />
12    <dc:identifier id="unique-id">XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX</dc:identif
ier>
13  </metadata>
14  <manifest>
15    <item media-type="application/xhtml+xml" id="doc-0" href="/xhtml/cover.xhtml"
properties="svg" />
16    <item media-type="application/xhtml+xml" id="doc-1" href="/xhtml/toc.xhtml" p
roperties="nav" />
17    <item media-type="application/xhtml+xml" id="doc-2" href="/xhtml/d_000.xhtml"
/>
18    :
19    <item media-type="application/xhtml+xml" id="doc-9" href="/xhtml/d_900.xhtml"
/>
20    <item media-type="image/jpeg" id="cover" href="/image/cover.jpg"/>
21    <item media-type="text/css" id="f-0" href="/xhtml/css/add.css" />
22    <item media-type="text/css" id="f-1" href="/xhtml/css/code.css" />
23    :
24    <item media-type="image/jpeg" id="f-4" href="/xhtml/image/dummy.jpg" />
25  </manifest>
26  <spine page-progression-direction="ltr">
27    <itemref idref="doc-0" />
28    <itemref idref="doc-1" />
```

```
29 ..... <itemref idref="doc-2" />
30 ..... :
31 ..... <itemref idref="doc-9" />
32 .... </spine>
33 </package>
```

このファイルは3つのブロックに分かれています。

## metadata

`<metadata>～</metadata>` の部分は書籍情報です。

## manifest

`<manifest>～</manifest>` の部分は収録ファイルの一覧です。

各項目は `<item>` で記載します。それぞれの項目にはメディアの種類である `media-type`、一意の名前である `id`、`content.opf` からの相対パスである `href` の情報を書きます。

`href` の値は、絶対パスで書いてはいけません。必ず `content.opf` からの相対パスにします。また、`properties="svg"` や `properties="nav"` といった設定を追加することもあります。

`svg` は、SVG形式での記載を意味します。`nav` は、ナビゲーションに利用することを意味します。

## spine

`<spine>～</spine>` の部分には本のページの順番を書きます。ここでは `page-progression-direction="ltr"` のようにページ送りの方向を設定できます。

各項目は `<itemref>` で記載します。`<itemref idref="doc-0" />` という項目なら、`<manifest>` の `id="doc-0"` のファイルを参照するという意味になります。

## 第2章 EPUB、KPFの注意事項

### 章の説明

ここでは作成するEPUBファイルやKPFファイルの注意事項について説明します。この章では次の話をします。

- XHTMLのタグ
- style属性
- 使用可能なタグとスタイル
- 等幅フォント
- スペースの幅
- 実機での確認

### XHTMLのタグ

電子書籍の本文はXHTML形式のファイルで作ります。これはMarkdown形式から変換して作ります。このファイルを作るときに、いくつか注意すべきことがあります。

XHTML形式はHTML形式とは全く同じではありません。 `<br>` や `<hr>` や `<img src="">` といったタグ（閉じタグ `</～>` がないHTMLタグ）は、そのままでは使えません。 `<br />` や `<hr />` や `<img src="" />` のように書く必要があります。

幸いなことに、PythonのMarkdownパッケージは、デフォルト設定でこの形式のタグに変換してくれます。

しかし、Markdown形式の原稿に直接書いたHTMLタグは変換してくれません。自分で直接タグを書く場合は注意が必要です。

### style属性

次にKPFファイルについての注意です。

KPFは、AmazonのKindle PreviewerでEPUBから変換して作るファイルです。Kindle端末やKindleアプリで表示する形式と思えばよいです。このKPFファイルは、Kindle特有の制約があります。

まず、HTMLタグに直接 `style` 属性を書けません。 `style="text-align: right;"` のような属性を書くとエラーになります。

見た目を調整する場合には `class="right"` のようにクラスを書き、CSSファイル内で `.right {text-align: right;}` のようにスタイルを書く必要があります。

PythonのMarkdownパッケージでは、テーブルの変換で `style` 属性を使います。これらをプログラム内で、クラスを使った形式に書き換える必要があります。

## 使用可能なタグとスタイル

Kindle端末では、一部のタグやスタイルは使えません。また、Kindle端末やKindleアプリに依存して、同じファイルでも表示結果が異なります。

そのため複雑なことをしようとせず、シンプルなレイアウトを心掛けた方がよいです。複雑なことをしようとすると高確率で破綻します。

Kindleは魔窟です。いくつかKindleの特徴を挙げます。

- 内部動作は公開されていません。
- **環境によって挙動が異なります。**
- CSSの指定どおり動かないことも多いです。
- **CSSの指定を無視して表示を最適化しようとします。**
- 利用可能と書いてあるタグやCSSが使えないことがあります。

こうした特徴があるため、XHTMLもCSSも必要最小限の使用に留める方がよいです。Amazon公式の参考情報を挙げます。

Kindle パブリッシング・ガイドライン

[https://kdp.amazon.co.jp/ja\\_JP/help/topic/GU72M65VRFP43L6](https://kdp.amazon.co.jp/ja_JP/help/topic/GU72M65VRFP43L6)

HTML および CSS のガイドライン

[https://kdp.amazon.co.jp/ja\\_JP/help/topic/GP2Z9DD4F2N4QKCY](https://kdp.amazon.co.jp/ja_JP/help/topic/GP2Z9DD4F2N4QKCY)

Kindle Format 8 でサポートされる HTML タグおよび CSS タグ

[https://kdp.amazon.co.jp/ja\\_JP/help/topic/GG5R7N649LECKP7U](https://kdp.amazon.co.jp/ja_JP/help/topic/GG5R7N649LECKP7U)

HTMLタグでは、次のものが使えません。動的なものは不可とっておくとよいです。

`<base>`、`<canvas>`、`<command>`、`<datalist>`、`<iframe>`、`<input>`、`<keygen>`、`<marquee>`、`<noscript>`、`<param>`、`<script>`、`<video>`

次のものは制限があります。

`<embed>` (SVGのみ)、`<object>` (SVGのみ)

次のものは廃止予定です。これらは昔のHTMLタグで、今は使われていないものです。



`<big>`、`<center>`、`<font>`、

また、`<p>` タグをネストすると適切に変換されません。

本文については、「`<`」「`>`」「`&`」を必ずHTMLエンティティ（「`&lt;`」「`&gt;`」「`&amp;`」）で書きます。

CSSについては「使えないもの」ではなく、「使えるもの」を確かめた方がよいです。Webブラウザでは利用できる最新のCSSの機能は使えません。また制限も厳しいです。

注意しておいた方がよい点を抜粋します。

## マイナス値

テキストと余白の配置にマイナス値の使用を避けます。ページをまたぐときに途切れることがあります。

## CSSセレクター

CSSセレクターの次兄弟結合子 `E + F`、後続兄弟結合子 `E ~ F` は利用できません。

CSSの擬似要素（`E::after` のように `::` を使うもの）や、擬似クラス（`E:first-child` のように `:` を使うもの）も利用できません。

## max-

`max-height` と `max-width` は使えません。

このスタイルは使えませんが、画像はページを超えるサイズにはなりません。自動調整されます。

## outline

`outline` は使えません。 `border` は使えます。

## 行間やフォントのサイズ

行の幅は指定しない方がよいです。Kindle側でユーザーが設定します。

また、本文のフォントサイズも設定しない方がよいです。こちらも、Kindle側でユーザーが設定します。

## 最適化

CSSの指定どおりのレイアウトになるとは限りません。Kindle独自の最適化により、CSSを無視することも多いです。

### 等幅フォント

Kindleのフォントのデフォルト設定は「明朝」です。本の制作者が割り当てたフォントを適用するには、ユーザーにフォントの設定を「Publisher Font」に変えてもらう必要があります。

Kindleではデフォルトのままの設定では、等幅フォントで表示されるタグで囲っても等幅フォントにはなりません。また、CSSで `font-family: monospace;` と指定しても、等幅フォントにはなりません。デフォルトのフォントで表示されます。デフォルトのフォントはプロポーショナルフォントなので、文字により横幅が違います。

またデフォルトのフォントは、セリフ体（ひげのある文字）が使われます。また英語と日本語が混在すると、異なる見た目のフォントが適用されます。そのためコードがきれいに表示されず、プログラミング系の技術書を作るときに困ります。

この問題を避けるには、2つの対策をしなければなりません。

- 本の冒頭で指示し、ユーザーにフォントの設定を「**Publisher Font**」に変えてもらう。
- フォントを同梱してそのフォントをCSSで適用する。

同梱するフォントについては、拡張子 `.ttf` のTrueTypeフォントが使えます。ただし、フォントを同梱すると、日本語フォントの場合は電子書籍のファイルサイズが肥大化します。この問題は、使用している文字のグリフ（図形）のみを収録したサブセットを作ることで避けられます。

サブセットをプログラムで作る方法は、のちほど説明します。

### スペースの幅

Kindleでは、スペースの幅は一定ではありません。各行の単語の詰まり具合に合わせてKindleが自動でサイズを変えます。この自動調整はCSSでは制御できません。

そのためプログラムのコード部分を、スペースやタブでインデントしているとレイアウトが崩れます。

## ▶ スペースの幅の問題

```
for title in con['titles']:
    f = title['file']
    t = title['title']
    l = title['level']
    h = con['tocLevelHead'][l - 1]
    children = title['children']
    li.append(f'<li class="toc-level-{l}"><a href="{f}">
{h}{t}</a></li>')

for child in children:
    t = child['title']
    l = child['level']
    h = con['tocLevelHead'][l - 1]
    id = child['id']
    li.append(f'<li class="toc-level-{l}"><a href="{f}">
#{id}>{h}{t}</a></li>')
```

この問題については割り切って諦めるか、コード部分を画像化する方法もあります。一時私は画像化していましたが、ファイルサイズが大きくなり、長大なコードの分割が難しいという問題がありました。

私は最近では半角の中黒をスペースの代わりに使い、CSSで `opacity: 0.05` のように薄い半透明にしています。

スペース以外の文字を使うことで、スペースの自動幅調整がおこなわれず、適切なサイズで表示されます。サブセットフォントと組み合わせれば、意図に近い形で表示をおこなえます。

## ▶ スペースの幅の対策

```
11  for title in con['titles']:
12      f = title['file']
13      t = title['title']
14      l = title['level']
15      h = con['tocLevelHead'][l - 1]
16      children = title['children']
17      li.append(f'<li class="toc-level-{l}"><a href="{f}">{h}{t}</a></li>')
18
19      for child in children:
20          t = child['title']
21          l = child['level']
22          h = con['tocLevelHead'][l - 1]
23          id = child['id']
24          li.append(f'<li class="toc-level-{l}"><a href="{f}">#{id}>{h}{t}</a></li>')
25
```

## 実機での確認

「Send to Kindle」を利用して、実機にEPUBを送って見え方を確認できます。

Kindleライブラリへのドキュメントの送信について -

Amazonカスタマーサービス

<https://www.amazon.co.jp/gp/help/customer/display.html?nodeId=G5WYD9SAF7PGXRNA>

Web経由で手軽に送ることができます。

意図通りにテキストが表示できていない場合は、下記のページも参考にしてください。

テキストのガイドライン - リフロー型

[https://kdp.amazon.co.jp/ja\\_JP/help/topic/GH4DRT75GWWAGBTU](https://kdp.amazon.co.jp/ja_JP/help/topic/GH4DRT75GWWAGBTU)