

**リッチエディタで
Markdown も編集
ハイブリッド入力対応**



概要

リッチエディタを「HTML + Markdown」が扱えるハイブリッド入力(造語)に対応させてみたメモ。

- 背景
- リッチエディタの利点
- リッチエディタ + Markdown
- その他

背景

[mardock\(このサイトをビルドしているウェブアプリ\)](#) でスライドの編集機能を実装するにあたり。

- 当初は「テキストエリア」で Markdown を編集する予定だった
 - ⇒ 画像の扱いに制約がある
- 「カスタムフィールド + 繰り返しフィールド」も試してみた
 - ⇒ 編集領域の表示幅が狭くなっていく
- 試しにリッチエディタでスライドを編集したら結構便利だった

⇒ 「リッチエディタの出力を Markdown へ変換 + α」 となる。

リッチエディタの**利点**
どの辺が結構便利だったのか

前提

メリット・デメリット系の話は前提で変わってくるので一応。

- リッチエディタと Markdown(テキストエリア)で編集したときの比較
- できるだけ公式で許容されている操作にとどめる
- API や [外部データ連携](#)等による更新は考慮していない

画像の扱い

Markdown(テキストエリア)に軍配が上がるのでは？

- 実際にやってみると「当初の印象ほどには」差がでない
 - 「普通に」入力する分には調整できる項目は同程度
- 最終的には「どのようにビルドするか」に行きつく
 - API レスポンスを処理するなら HTML でも可能

「入力時のフィールド型は影響小」⇒「UI的にリッチエディタ有利」
となる。

操作系がシンプル

WYSIWYG 的なエディターは Markdown に比べると操作が煩雑では？

- マークダウン記法(「##」 + 「スペース」で見出し変換等)によりキー操作の違いは少ない
- 全体で 1 つのフィールドなのでカーソル移動が比較的的自然
- モバイル環境でも(慣れれば)装飾ボタンで編集可能

「画像のインライン表示可能な Markdown エディター」に近い感覚。プレビューとエディターを切り替える場合、画像のインライン表示は視線移動の目印になる。

文章入力に集中できる

「個人の感想とかそういうのいらないです」となりそうですが。

- 「繰り返しフィールドで入力」 「リッチエディタで入力」のサイトをそれぞれ作成
 - 繰り返しフィールドは入力と配置で思考の切り替えが必要
 - リッチエディタは入力に意識を向けたままにできる

用途にもよるのでどちらが良いというわけではないですが、今回はこの辺の「使用感」が大きく影響しています。

リッチエディタ + Markdown

= ハイブリッド入力

フォーマット変換 + α の必要性

編集上の利点からリッチエディタを使うとしても、そのままでは Marp スライドを作成できない。

- リッチエディタのデフォルトの出力は HTML
 - \Rightarrow フォーマット変換が必要
- リッチエディタでは「HTML コメント(プレゼンターノートなどに利用)の入力ができない」「テーブルが使えない」
 - \Rightarrow 何かしらの入力方法が必要

リッチエディタ(HTML)を Markdown へ変換

[unified](#) でプラグインを組み合わせることで対応可能。

- 基本的な変換は [rehype-remark](#) で実施
- パラグラフの分割など再利用可能なものは[パッケージ](#)を作成
- 今回用の特殊な処理も Transformer 作成で対応

コード

```
const htmlToMarkdownProcessor = unified()
  .use(rehypeParse, { fragment: true })
  .use(firstParagraphAsCodeDockTransformer)
  .use(splitParagraph)
  .use(rehypeSanitize, { allowComments: true })
  .use(rehype2Remark, {
    handlers: {
      pre: codeDockHandler,
      br: (h: any, node: any) => {
        return h(node, 'text', ' ');
      }
    }
  })
  .use(stringify)
  .freeze();
```

Markdown の埋め込み

「HTML コメント」「テーブル」対応として部分的な Markdown の利用。

- コードブロックで先頭行を `===md` とすると Markdown として扱う
- Markdown で HTML を使う手法の逆転版に近いイメージ

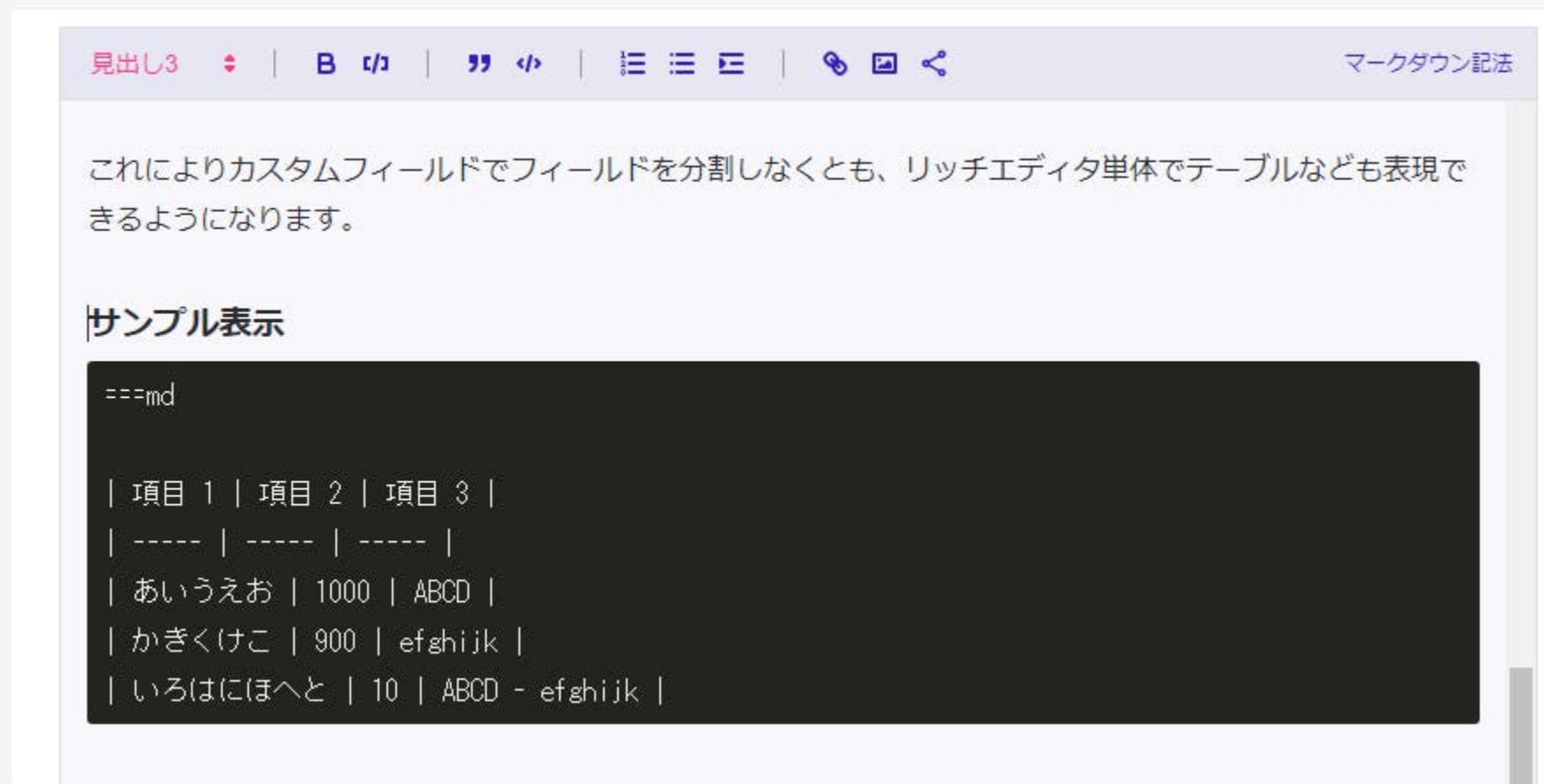
```
===md
```

ここは Markdown として扱われます。

エスケープに注意は必要ですが(リッチエディタでは文字参照の扱いが特殊)、基本的には思ったように記述可能。

サンプル表示

入力時の画面。



The screenshot shows a rich text editor interface. At the top, there is a toolbar with various icons for text formatting (bold, italic, underline, strikethrough, link, unlink, list, indent, outdent) and a 'マークダウン記法' (Markdown) button. Below the toolbar, there is a text area containing the following text:

これによりカスタムフィールドでフィールドを分割しなくとも、リッチエディタ単体でテーブルなども表現できるようになります。

サンプル表示

```
===md

| 項目 1 | 項目 2 | 項目 3 |
| ----- | ----- | ----- |
| あいうえお | 1000 | ABCD |
| かきくけこ | 900 | efghijk |
| いろはにほへと | 10 | ABCD - efghijk |
```

実際のテーブル表示。

項目 1	項目 2	項目 3
あいうえお	1000	ABCD
かきくけこ	900	efghijk
いろはにほへと	10	ABCD - efghijk

その他

ハイブリッド入力の功罪

Marp 以外での利用

その他のコンテンツでも Markdown へ変換することで以下のような利点が生まれます。

- Markdown 用のツールが適用できる
- 機械的な変換を通すことでコンテンツの構造を均一化しやすい

たとえば [highlight.js](#) の利用は、Markdown を HTML へ戻す処理に [remark-highlight.js](#) プラグインを追加するだけでほぼ完了します。

スライド以外でも試してみた[サンプル](#)。

コード

```
export function processorMarkdownToHtml() {  
  return unified()  
    .use(markdown)  
    .use(highlight)  
    .use(gfm)  
    .use(remark2rehype, { allowDangerousHtml: true })  
    .use(raw);  
}
```

セキュリティ

残念ながら Markdown を混在させることで問題点も出てきます。その1つとして、リッチエディタで制限されていた操作が編集者へ開放されることが挙げられます。

今回は [hast-util-sanitize](#) によるサニタイズを行ってはいますが、利用したい機能に合わせて制限を解除しているため、ほとんど意味がない状態となっています。

現状では自分だけが編集するので許容できますが、本来であれば何らかの制限を考える必要が出てきます。

その他

mardock では「[unified](#)」「[cheerio](#)」「[Markdown-it\(Marpit\)](#)」が混在しているので、もう少し整理したいところです。