MkDocs によるドキュメント作成

mebiusbox

2022 年 12 月 28 日

目次

1. はじめに

mkdocs は 静的サイトジェネレータです。コンテンツは基本的に markdown 形式で記述したソースファイルになります。また HTML 形式のファイルを使うことも出来ます。この記事では Windows で個人的な最低限の作成環境をまとめました。

MkDocs

2. python のインストール

python をインストールします。2.x 系でも 3.x 系でもどちらでも構いません。次に pip を使って mkdocs をインストールします。

```
pip install mkdocs
```

私は次の環境で動作確認しました.

```
python                  3.7.0
mkdocs                  1.0.4
mkdocs-material         4.1.1
fontawesome-markdown    0.2.6
mdx_unimoji             1.0
python-markdown-math    0.6
pymdown-extensions      6.0
```

3. MkDocs のコマンド

サイトに必要なファイルを生成するにはビルドを行います。ビルドは以下のコマンドを実行します。

```
mkdocs build
```

正常に生成されると site フォルダに作成されます。インターネット上に公開する場合はこのフォルダ内をアップロードします。

ビルドした内容を公開する前に、ローカルで確認したい場合は serve コマンドを実行し、サーバーを起動します。

```
mkdocs serve
```

コマンドの出力に `Serving on http://127.0.0.1:8000` のようにアドレスとポート番号が表示されますので、そのアドレスをブラウザに入力して確認することが出来ます。

serve を実行している間はファイルの追加や変更が検知されて自動的にビルドされます。

4. カスタマイズ

4.1 mkdocs.yml

mkdocs.yml ファイルを編集することでカスタマイズを行うことができます。このファイルにはサイトのタイトルやテーマ、拡張機能の設定などを記述します。プロジェクトの作成を行うと、このファイルは自動で作成されます。

4.2 テーマ

mkdocs にはテーマの機能があります。よく使われている(と思う)テーマは readthedocs です。このテーマは mkdocs に標準で入っており、サイトに適用させる場合は mkdocs.yml に以下を追加します。

```
theme: readthedocs
```

これ以外のテーマでは material がオススメです。

Material for MkDocs

詳しくは次の記事が参考になります。

カンタンにドキュメントが作れる mkdocs をはじめてみよう

このテーマを使う場合はインストールする必要があります。以下のコマンドを実行してインストールします。

```
pip install mkdocs-material
```

そして mkdocs.yml で theme に material を指定します。

readthedocs や material のデザインは素晴らしいのですが、いくつか気に入らないところがあるので調整していきます。

4.3 フォントのカスタマイズ

日本語も美しく表示するためにフォントを指定します。ここでは Google の Noto Fonts を使います。このフォントを使うにはスタイルシートを追加する必要があります。mkdocs.yml で extra_css で外部のスタイルシートを指定することができます。

```
extra_css:
    - "//fonts.googleapis.com/earlyaccess/notosansjp.css"
    - "//fonts.googleapis.com/css?family=Open+Sans:600,800"
```

フォントを変更するにはカスタム用のスタイルシートを作成してリンクさせます。docs フォルダに css フォルダを作成し、custom.css を作成します。内容は次の通りです。

```
body {
    font-family: "Noto Sans JP";
}
```

このファイルを extra_css に追記します。

```
extra_css:
    - "https://fonts.googleapis.com/earlyaccess/notosansjp.css"
    - "https://fonts.googleapis.com/css?family=Open+Sans:600,800"
    - "css/custom.css"
```

相対パスの場合は docs フォルダが基準パスになります。

また、material テーマの場合は通常のフォントとコード用のフォントを別々に指定することが出来ます。

```
extra:
    font:
        text: "Noto Sans JP"
        code: "Consolas"
```

extra で、テーマや拡張機能に対する設定を行うことが出来ます。この場合は custom.css は不要になります。

4.4 デザインのカスタマイズ

readthedocs のナビゲーション部分は個人的にビミョーに見づらく、もっとコントラストを強くしたほうがいい気がします。

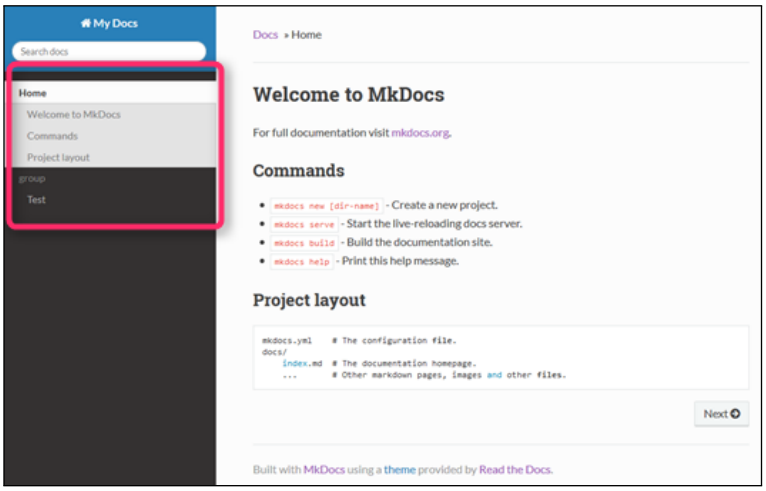

Fig.1: デザインのカスタマイズ

スタイルシート(custom.css)を編集してフォントカラーを変更します。

```
.wy-menu-vertical a, .wy-menu-vertical .subnav a {
    color: white;
}

.wy-menu-vertical * li.current a {
    color: black;
}

.wy-menu-vertical span {
    color: #ff0;
}
```

これで次のようになります。

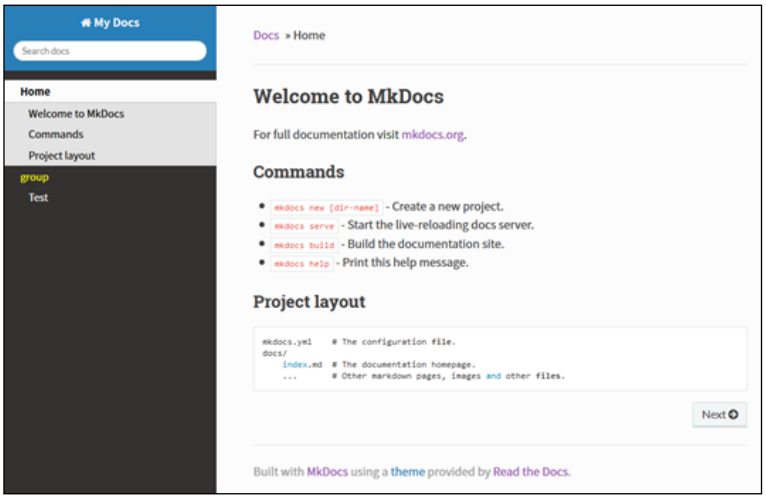

Fig.2: フォントの色を変更

material テーマの場合は extra でカラーの指定が出来ます。

```
extra:
    palette:
        primary: "indigo"
        accent: "red"
```

詳しくは公式サイトを参照してください。

4.5 数式

mathjax による数式の表示を行うことが出来ます。まず、**python-markdown-math** をインストールします。

```
pip install python-markdown-math
```

次に mkdocs.yml で拡張機能に mdx_math を指定します。

```
markdown_extensions:
    - mdx_math
```

また、外部 Javascript も追加する必要があります。これは extra_javascript で指定します。

```
extra_javascript:
    - http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML
```

これで数式が表示出来ます。

Fig.3: 数式

これは下記のように記述されています。

$$P \cdot Q = \|P\|\|Q\| \cos \alpha$$

インラインでの数式は `\\(` と `\\)` で囲みます。また `$` もよく使われますが、標準で無効になっているので、明示的に有効にする必要があります。その場合は mdx_math のオプション enable_dollar_delimiter で設定します。

```
markdown_extensions:
    - mdx_math:
        enable_dollar_delimiter: true
```

4.6 警告文

拡張機能 **Admonition** を使うと、文書内にメモ、ヒント、警告などが目立つようなスタイルで表示してくれます。Admonition も含め、これから紹介する機能は material テーマをインストールすると一緒にインストールされるので、material テーマをインストールすることをオススメします。

使う場合は mkdocs.yml で使うように指定します。

```
markdown_extensions:
    - admonition
```

Markdown では `!!! Note` のように使います。例えば以下のような内容の場合:

```
!!! Note
    これはノートです

!!! Tip
    これはヒントです

!!! Warning
    これは警告です

!!! Danger
    これは危険です
```

```
!!! Success
    これは成功です

!!! Failure
    これは失敗です

!!! Bug
    これは不具合です

!!! summary
    これは概要です
```

readthedocs テーマのとき

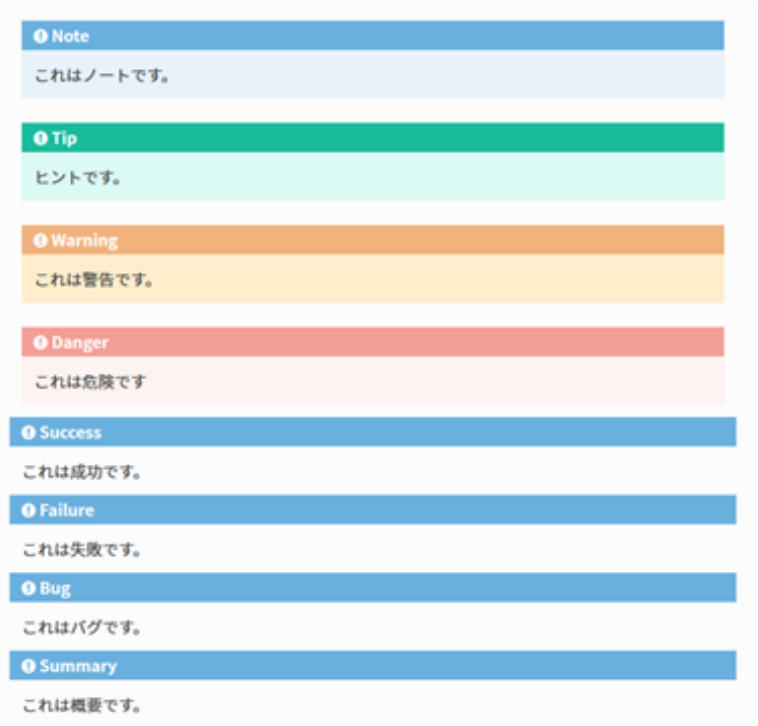

Fig.4: readthedocs テーマの admonition

material テーマのときは次のように表示されます.

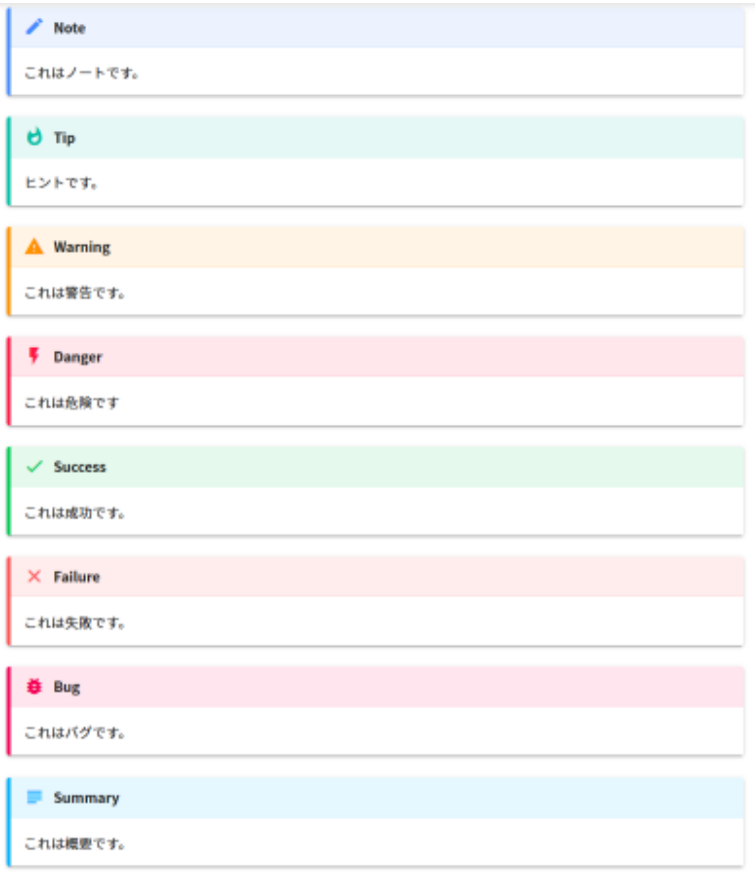

Fig.5: material テーマの admonition

4.7 注釈

拡張機能 **footnotes** を使います。これも material テーマに含まれています。注釈をつけるには、つけたい言葉の後ろに `[^1]` のように記述します。

```
Mkdocs とは静的サイトジェネレータです。
コンテンツは基本的に markdown[^1] 形式で記述したソースファイルになります。

[^1]: 文書を記述するための軽量マークアップ言語のひとつ
```

上記の場合、次のように表示されます。

Fig.6: 注釈

4.8 Font Awesome

Font Awesome とは Web アイコンフォントです。画像を用意しなくても、手っ取り早くアイコンを表示することができます。ベクター形式なので、サイズを変更しても綺麗です。もちろん色も変更することができます。

まず、**fontawesome_markdown** をインストールします。

```
pip install fontawesome_markdown
```

次に mkdocs.yml に追記します。

```
markdown_extensions:
    - fontawesome_markdown

extra_css:
    - "https://maxcdn.bootstrapcdn.com/font-awesome/4.6.1/css/font-awesome.min.css"
```

Markdown には `:fa-coffee:` のように記述します。例えば以下のように記述した場合:

```
:fa-external-link: [MkDocs](http://www.mkdocs.org/)
```

このように表示されます。

Fig.7: Font Awesome

4.9 定義リスト

定義リストは定義語のリストを作成する拡張機能です。この拡張機能を使用するには **def_list** を追加する必要があります。

使い方は以下のようになります。

```
定義語
:    ここに説明を書きます
```

これは次のように表示されます。

Fig.8: 定義リスト

4.10 絵文字

絵文字を使用するには **pymdownx.emoji** を拡張機能リストに追加します。

```
markdown_extensions:
    - pymdownx.emoji:
        emoji_generator: !!python/name:pymdownx.emoji.to_svg
```

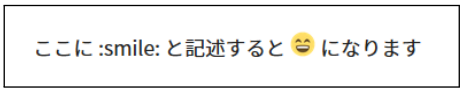

Fig.9: 絵文字

絵文字の種類を確認する場合は以下のサイトが便利です。

EMOJI CHEAT SHEET

この pymdownx.emoji ですが、material テーマのみフォントに合ったサイズで表示され、それ以外のテーマではサイズが合わずに表示されてしまいました。その場合は次のようにします.

```
  markdown_extensions:
    - pymdownx.emoji:
        emoji_generator: !!python/name:pymdownx.emoji.to_alt
```

また，**mdx_unimoji** というのも使えます。これは別途インストールする必要があります。

```
pip install mdx_unimoji
```

使用する場合は mdx_unimoji を拡張機能リストに追加します。mdx_unimoji で絵文字に変換されるコードは以下のソースコードを参照してください。

mdx_unimoji.py

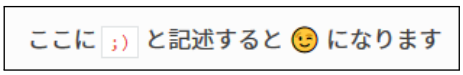

Fig.10: unimoji

4.11 SmartSymbols

SmartSymbols を使うと一部のシンボルを特定の文字の組み合わせで変換してくれます。拡張機能リストに pymdownx.smartsymbols を追加します。

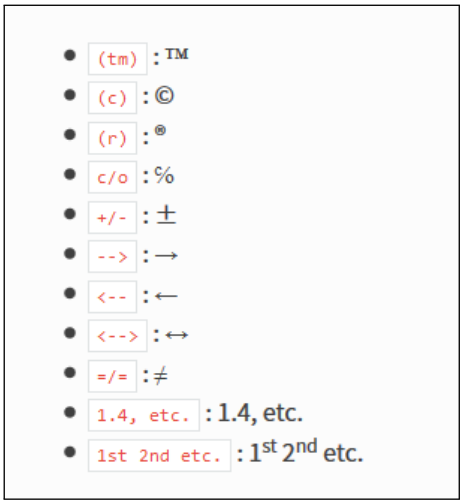

Fig.11: SmartSymbols

拡張機能を有効にした状態だと、上記の特定文字に一致するところがすべて変換されてしまいます。ですが、個別に有効・無効を設定することができます。詳しくは以下を参照してください。

SmartSymbols

4.12 Keys

PC でのキーボード操作を説明するときにキーコードを解りやすく装飾してくれる拡張機能です。ただし、material テーマのみ使えます。使用するには拡張機能リストに **pymdownx.keys** を追加します。

使い方は以下のようになります。

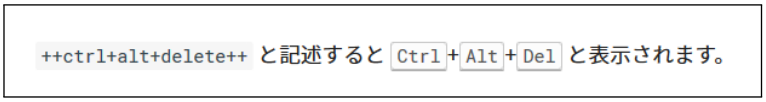

Fig.12: Keys

詳細は以下を参照してください。

Keys

4.13 タブ

material テーマにはタブによるページ管理の機能があります.有効にする場合は theme を次のようにします.

```
theme:
  name: material
  feature:
    tabs: true
```

トップレベルのページがタブになります.

Fig.13: タブ

4.14 コードハイライト

拡張機能 **codehilite** を使います.

```
markdown_extensions:
  - codehilite
```

コードは fenced code blocks (```) バッククォートを書いて囲みます.

````
```
#include <iostream>
void main() {
    std::cout << "Hello world!" << std::endl;
}
```
````

```
#include <iostream>
void main() {
  std::cout << "Hello world!" << std::endl;
}
```

Fig.14: codehilite

行番号を表示する場合は linenums を設定します.

```
markdown_extensions:
  - codehilite:
      linenums: true
```

```
1 #include <iostream>
2 void main() {
3   std::cout << "Hello world!" << std::endl;
4 }
```

Fig.15: codehilite(linenums)

ハイライトのテーマを変更する場合は次のようにします.なお, ここでのハイライトのテーマに関する内容は material テーマにおいて綺麗に表示されるようになっています.

```
markdown_extensions:
  - codehilite:
      use_pygments: true
      noclasses: true
      pygments_style: monokai
```

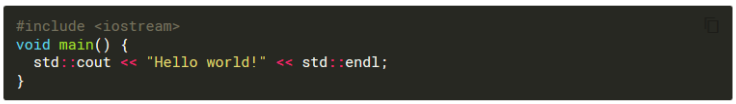

Fig.16: codehilite(monokai)

上記の方法でテーマを変更した状態で行番号を表示すると, おかしな見た目になるので注意です.

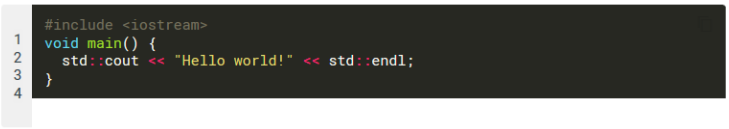

Fig.17: codehilite(monokai+linenums)

コードの表示では前後のスペース(マージン)をなくして表示しているものをよく見るようになりました. Qiita もそのように変更されていますね.

そこでスタイルシートを使って対応してみます.また, 行番号の表示も調整します.次のような内容をスタイルシートに追加します.

```
.md-typeset .codehilitetable {
  margin-left:-20px;
  margin-right: -20px;
  border-radius: 0;
}
.md-typeset .codehilitetable .linenodiv {
  background-color: #222 !important;
}
.md-typeset .codehilitetable .linenodiv pre {
  background-color: #222 !important;
  color: #aaa;
  margin: 0;
}
.md-typeset .codehilitetable .md-clipboard:before {
  color: rgba(240,240,240,.8);
}
.md-typeset .codehilitetable .md-clipboard:hover:before {
  color: rgba(102,217,224,1);
}
```

これを適用すると次のように表示されます.

```cpp
#include <iostream>
void main() {
  std::cout << "Hello world!" << std::endl;
}
```

Fig.18: codehilite(custom)

4.15 見出しのカスタマイズ

見出しは先頭にアイコンをつけたり, 下線を追加すると格段に見やすくなります.ここでは material テーマに対して見出しを調整します.

まず, アイコンは **Font Awesome** を使えば画像を用意しなくても表示することができます.例えば次のようなアイコンを指定してみます.

```
<i class="fa fa-arrow-circle-right" aria-hidden="true"></i>
```

そうすると以下のように表示されます.

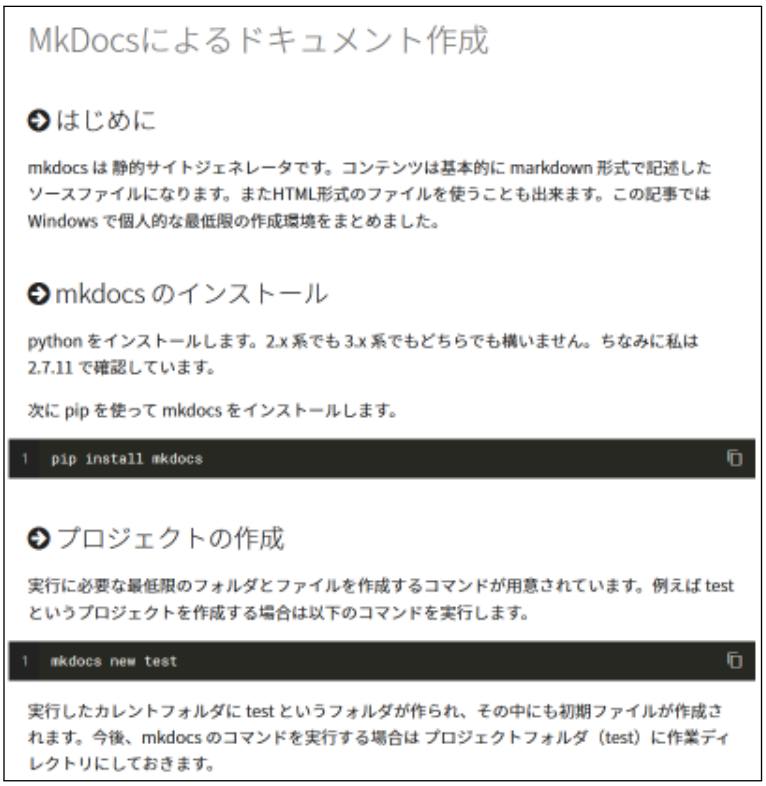

Fig.19: 見出しのカスタマイズ(アイコン)

続いて見出しを調整します.個人的に GitLab Documentation のドキュメントが見やすいのでそれを真似てみます.スタイルシートに以下を追加します.

```
.md-typeset h1 {
  /*font-size: 24pt;*/
  font-weight: bold;
  color: #000;
  border-bottom: solid 2px #f18b21;
  padding-bottom: 5px;
```

```
}
.md-typeset h2 {
  border-bottom: 1px dotted #888;
}
```

フォントサイズはお好みのサイズを指定してください.これは次のように表示されます.

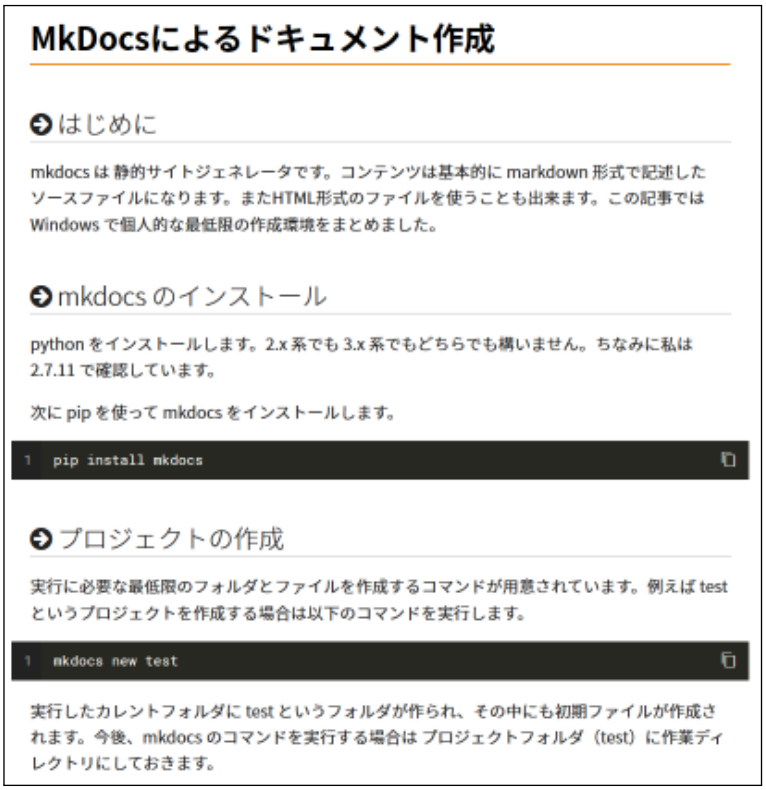

Fig.20: 見出しのカスタマイズ(下線)

4.16 Mathjax のカスタマイズ

(2019/3/30) mkdocs.yml の extra_javascript で mathjax のスクリプトの前に指定することで設定することができます.ここでは,‘js/extra.js‘ というファイルを作成した場合で, 内容は次のようにしました.

```
window.MathJax = {
  tex2jax: {
    inlineMath: [['$','$'], ['\\(','\\)']],
    displayMath: [['$$','$$'], ['\\[','\\]']],
    processEscapes: true
  },
  "HTML-CSS": {
    availableFonts: [],
    preferredFont: null,
    webFont: "STIX-Web"
  }
};
```

そして，mkdocs.yml の extra_javascript を

```
extra_javascript:
    - "js/extra.js"
    - "http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML"
```

とします．

(ここまで)

extra_javascript に追加することで一部の設定は反映されるようですが，Mathjax の設定は text/x-mathjax-config で指定する必要があるらしく，extra_javascript では設定できません．ではどうするかというと，テーマをカスタマイズすることで対応します．今回は material テーマをカスタマイズします．まず，material テーマを以下の場所からダウンロードします．

squidfunk/mkdocs-material

material フォルダ以下の次のフォルダとファイルを theme フォルダに入れます．

```
(root)
  + docs
  + theme
    + assets
    + partials
    - base.html
  - mkdocs.yml
  ...
```

Mathjax のフォントを変更してみます．base.html の以下の部分を探します．

```
{% for path in extra_javascript %}
  <script src="{{ path }}"></script>
{% endfor %}
```

そこに Mathjax の設定を追加します．

```
{% for path in extra_javascript %}
  <script src="{{ path }}"></script>
{% endfor %}
<script type="text/x-mathjax-config">
MathJax.Hub.Config({
  tex2jax: {
    inlineMath: [['$','$'], ['\\(','\\)']],
    displayMath: [['$$','$$'], ['\\[','\\]']],
    processEscapes: true
  },
  "HTML-CSS": {
    availableFonts: [],
```

```
    preferredFont: null,
    webFont: "STIX-Web"
  }
});
</script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/mathjax/2.7.1/MathJax.js?config=TeX-AMS-MML_HTMLorMML"></
script>
```

次にカスタマイズしたテーマを使用するように設定します. mkdocs.yml で custom_dir を追記します.

```
theme:
  name: material
  custom_dir: 'theme'
```

4.17 作図

mkdocs の機能ではありませんが、様々な図を挿入することが出来ます。詳しくは以下を参照してください。

MkDocs

また、UML 図を挿入したい場合は yUML がオススメです。

yUML

これについては以下が参考になると思います。

yUML で Qiita にテキストベースで UML を埋め込んでみる。Qiita, esa, GitHub Wiki なんでもござれ

5. 外部サービスとの連携

5.1 Google アナリティクス

Google アナリティクス用のトラッキングコードを各ページに埋め込むことが出来ます. google_analytics で, トラッキング ID を指定します.

```
google_analytics:
  - 'UA-XXXXXXXX-X'
  - 'auto'
```

5.2 Disqus

Disqus というサービスを使ってコメント欄をページに配置することができます.これは Material テーマが対応しています. extra に disqus を追加し, shortname を設定します. shortname は Disqus の管理画面で確認することができます.

```
extra:
  disqus: '<shortname>'
```

disqus を利用するために site_url には適切な URL を設定しておく必要があります.

5.3 その他の拡張機能

以下にまとまっていますので、参考にしてみてください。

Third Party Extensions

6. 検索の日本語対応

mkdocs でビルドしたサイトには検索機能が入っているのですが、残念ながら日本語には対応していません。そこで、なんとか対応してみます。これについては以下が参考になりました。

mkdocs の検索を日本語に対応させてみる

ただ、上記のサイト通りでは上手くいきませんでした。

ビルドしたサイトの検索は site/mkdocs/js/search.js で行われています。lunr.js を使っているのですが、これを多言語に対応させるライブラリ **lunr-languages** が公開されていますので、それを導入します。

lunr-languages

ライブラリをダウンロードして、その中から以下のファイルを site/mkdocs/js/にコピーします。

- lunr.stemmer.support.js
- lunr.ja.js
- tinyseg.js

次に search.js ファイルを編集します。

```
require([
    base_url + '/mkdocs/js/mustache.min.js',
    base_url + '/mkdocs/js/lunr.min.js',
    base_url + '/mkdocs/js/lunr.stemmer.support.js',//追加
    base_url + '/mkdocs/js/lunr.ja.js',//追加
    base_url + '/mkdocs/js/tinyseg.js',//追加
    ...
    //], function (Mustache, lunr, results_template, data) {
    ], function (Mustache, lunr, stemmer, ja, tiny, results_template, data) {
    ...

    stemmer(lunr);//追加
    tiny(lunr);//追加
    ja(lunr);//追加

    var index = lunr(function () {
        this.use(lunr.ja);//追加
        this.field('title', {boost: 10});
        this.field('text');
        this.ref('location');
    });
    ...
```

この方法ではデフォルトのテーマと readthedocs テーマで動作しました。ビルドすると search.js が再生成されるので、ビルドする度にパッチで変更を適用するといった対応が必要になります。

material テーマでは標準で日本語対応しているようです.以下のように設定ファイルに追加します.

```
theme:
    name: material
    language: 'ja'
extra:
    search:
        language: 'jp'
```

(2019/3/30) 最新の material テーマ(4.1.1)では次のように設定します.

```
theme:
    name: material
extra:
    search:
        language: 'ja'
```

7. GitHub Pages

7.1 GitHub Pages へのデプロイ

mkdocs には GitHub Pages にデプロイするコマンドが用意されています。

GitHub でリポジトリを作成し、クローンして mkdocs のプロジェクトを構築して、以下のコマンドを実行します。

```
mkdocs gh-deploy
```

これだけでデプロイすることが出来ます。

- サイト: https://mebiusbox.github.io/MkDocsTest/
- ソース: https://github.com/mebiusbox/MkDocsTest

7.2 GitHub Pages での Mathjax

GitHub Pages にデプロイする場合、最新の Mathjax だと正常に表示されないので、バージョンを指定する必要があります。

```
extra_javascript:
    - https://cdnjs.cloudflare.com/ajax/libs/mathjax/2.7.1/MathJax.js?config=TeX-AMS_HTML
```

こちらを使う場合は mdx_math ははずしておきます。

8. さくらサーバーでのデプロイ環境

さくらサーバーに git のリモートリポジトリを作成し、push したらサーバーでビルドしてデプロイするところまで構築する手順です。

まず、mkdocs をサーバーにインストールする必要があります。mkdocs をインストールするために pip が必要なので、こちらを先にインストールします。権限の問題で、インストールする場所を指定しなければなりません。

```
mkdir -p $HOME/.local/lib/python2.7/site-packages
easy_install --prefix=$HOME/.local pip
```

実行ファイルは `$HOME/.local/bin` になりますので、シェルからも実行出来るように.chsrc ファイルを編集して、path に$HOME/.local/bin を追加します。その後、設定を反映させるために `source $HOME/.chsrc` を実行しておきます。

これで mkdocs をインストール出来ます。インストールするときは場所の指定も忘れずに。

```
pip install --install-option="--prefix=$HOME/.local" mkdocs
```

拡張機能をインストールするときも `--install-option="--prefix=$HOME/.local"` が必要です。

次にサーバーに git のリモートリポジトリを作成します。

```
git init --bare --share $HOME/hoge.git
```

また、ローカルリポジトリもサーバーに作成しておきます。

```
git clone $HOME/hoge.git
```

push したときにスクリプトを実行させるためのフックを設定します。フックスクリプトはリモートリポジトリの hooks ディレクトリに作成します。push したときに実行されるスクリプトは post-receive になります。

```
vi $HOME/hoge.git/hooks/post-receive
```

スクリプトではローカルリポジトリで pull して、`mkdocs build` を実行、site ディレクトリを$HOME/www/hoge にコピーします。post-receive の内容は以下の通りです。

```
#!/bin/sh
cd $HOME/hoge
git --git-dir=.git pull
mkdocs build
\cp -rf ./site/ $HOME/www/hoge/
```

フックスクリプトには実行権限を与えておきます。

```
chmod +x $HOME/hoge.git/hooks/post-receive
```

これでプッシュしたときに自動でデプロイされる環境になりました。フックスクリプトの実行結果はプッシュしたときに確認することができます。

9. ちょっと詰まったところ

mkdocs.yml の設定で以下のようにするとエラーが表示されてしまいました。

```
ERROR   -  Config value: 'markdown_extensions'. Error: Invalid Markdown Extensions configuration
```

mkdocs.yml の内容:

```
markdown_extensions:
    ...
    - toc:
    permalink: true
```

これは mkdocs の公式に記載されている通りなのですが上手くいきません。原因はインデントのようでした。空白2つでインデントしていたのですが、それがダメなようです。空白を増やすことで正常に動作しているようです。YAML や mkdocs の仕様なのかバグなのかはわかりません。どちらもそんな仕様ではないと思いますけど。

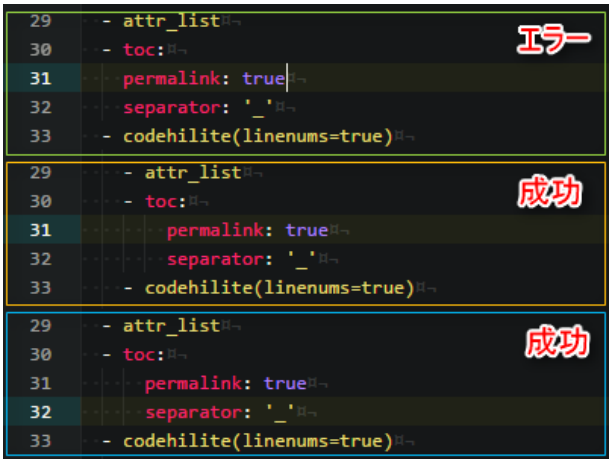

Fig.21: mkdocs.yml

9.1 pymdownx.* で見つからないとエラーが出る

install に失敗しているか、バージョンが古い可能性があります。再インストールしてみると上手くいく場合があります。

```
pip uninstall pymdown-extensions
pip install pymdown-extensions
```

10. Pages の自動生成

サイトにページを追加するにはソースファイル(markdown 形式)を作成し、mkdocs.yml の pages に追加していく必要があります。ページの数が多いと面倒ですし、見出しは mkdocs.yml にも書かなくてはいけないので、ページのソースファイルを変更した場合に mkdocs.yml 側も修正しなければなりません。そのため、ドキュメントフォルダ内をスキャンして自動で生成するスクリプトを作成してみました。スクリプトは python 2.7.11 で動作確認しています。

まず、目次に表示されるページのタイトルはソースファイルの YAML Front Matter から title を参照します。python で YAML Front

Matter を読むには **python-frontmatter** を使います。

```
pip install python-frontmatter
```

次に YAML の入出力を行うために **ruamel.yaml** を使います。

```
pip install ruamel.yaml
```

YAML の入出力は PyYAML というモジュールもあって、これは python-frontmatter と一緒にインストールされます。ruamel.yaml を使う理由は、シングルクォーテーション、ダブルクォーテーションの扱いが PyYAML よりも優れているからです。
このスクリプトはベースファイルを読み込んで、pages を書き換える仕組みになっていますので、あらかじめ mkdocs.yml を作成しておきます。
自動生成するといっても、目次の部分はある程度順番を決められるようにする必要があります。このスクリプトでは docs フォルダ内のフォルダ・ファイルを引数に渡して、その順番どおりに目次を作成するようにします。通常、データは辞書タイプになっていますが、順番を維持するために OrderedDict を使う必要があります。また、出力時にシングルクォーテーション・ダブルクォーテーションを追加するために ruamel.yaml.scalarstring.SingleQuotedScalarString と ruamel.yaml.scalarstring.DoubleQuotedScalarString を使います。

```
# ファイル列挙
import glob

# YAML FrontMatter
import frontmatter

# YAML
import ruamel.yaml
from ruamel.yaml.scalarstring import SingleQuotedScalarString, DoubleQuotedScalarString

# OrderedDict
from collections import OrderedDict
```

次のコードはベースファイル(mkdocs.yml)を読みこみ、ドキュメントフォルダをスキャンした結果で pages の内容を書き換えて出力します。

```
def gen_page_od(path, dirname):

    name = os.path.basename(path)
    post = frontmatter.load(path)
    if 'title' in post.keys():
        title_sq = SingleQuotedScalarString(post['title'])
        value_sq = SingleQuotedScalarString(dirname + name)
        od = OrderedDict()
        od[title_sq] = value_sq
        return od
    else:
```

```
        return SingleQuotedScalarString(dirname + name)
    # print 'F:' + f, name

def gen_pages_od(path, dirname):
    ret = []
    files = glob.glob(os.path.join(path, '*'))
    for f in files:
        if os.path.isdir(f):
            name = os.path.basename(f)
            if not name.startswith('_'):
                name_sq = SingleQuotedScalarString(name)
                od = OrderedDict()
                od[name_sq] = gen_pages_od(f, dirname + name + '/')
                ret.append(od)
        else:
            ret.append(gen_page_od(f, dirname))
    return ret

def main(argv=None):
    if argv is None:
        argv = sys.argv
    try:
        opts,args = getopt.getopt(sys.argv[1:], "h", ["help"])
    except getopt.error,msg:
        print msg
        print "for help use --help"
        return 2

    for o,a in opts:
        if o in ("-h","--help"):
            print __doc__
            return 0

    f = open('mkdocs.yml')
    data = ruamel.yaml.round_trip_load(f, preserve_quotes=True)
    f.close()

    pages = None;
    if len(args) == 0:
        pages = gen_pages_od('./docs/', '')
    else:
        pages = []
        for a in args:
            path = './docs/' + a
            if os.path.isdir(path):
```

```
                name_sq = SingleQuotedScalarString(a)
                od = OrderedDict()
                od[name_sq] = gen_pages_od(path, a + '/')
                pages.append(od)
            else:
                pages.append(gen_page_od(path, ''))

    f = codecs.open(filename, 'w+', 'utf-8')
    data['pages'] = pages
    f.write(ruamel.yaml.round_trip_dump(data, default_flow_style=False, encoding='utf-8', allow_unicode=True))
    f.close()

    return 0

if __name__ == '__main__':
    sys.exit(main())
```

10.0.1 preserve_quotes

`ruamel.yaml.round_trip_load(f, preserve_quotes=True)` で preserve_quotes を True にすることでクォーテーションも読み込んでくれます。これは SingleQuotedString や DoubleQuotedString で変換済みということです。

10.0.2 default_flow_style

default_flow_style を False にすることでブロックスタイルにします。

10.0.3 encoding

日本語にも対応させるため、encoding に utf-8 を指定します。ファイル出力のとき codecs.open を使います。

10.0.4 allow_unicode

ユニコード文字列を正しく変換するために allow_unicode を True にします。

10.0.5 OrderedDict

標準のままでは OrderedDict 形式のデータがあると YAML のタグ(!!omap)も出力されてしまいます。これに対応するため、以下のコードを追加します。

```
def ruamel_represent_odict(dumper, instance):
    return dumper.represent_mapping('tag:yaml.org,2002:map', dict(instance))
ruamel.yaml.representer.RoundTripRepresenter.add_representer(
    OrderedDict,
    ruamel_represent_odict)
```

最終的なスクリプトは以下のようになります。mkdocs.yml を書き換える前にバックアップファイルを取るようにしています。また、例外として

フォルダ名の先頭が _のときは無視するようになっています。

```
# -*- coding:utf-8
"""
gen-pages.py
replace pages in mkdocs.yml with contents in document folder
"""
import sys
import getopt
import os
import glob
import frontmatter
import ruamel.yaml
from ruamel.yaml.scalarstring import SingleQuotedScalarString, DoubleQuotedScalarString
from collections import OrderedDict
import codecs
import shutil

def ruamel_represent_odict(dumper, instance):
    return dumper.represent_mapping('tag:yaml.org,2002:map', dict(instance))

ruamel.yaml.representer.RoundTripRepresenter.add_representer(
    OrderedDict,
    ruamel_represent_odict)

def gen_page_od(path, dirname):
    name = os.path.basename(path)
    post = frontmatter.load(path)
    if 'title' in post.keys():
        title_sq = SingleQuotedScalarString(post['title'])
        value_sq = SingleQuotedScalarString(dirname + name)
        od = OrderedDict()
        od[title_sq] = value_sq
        return od
    else:
        return SingleQuotedScalarString(dirname + name)

def gen_pages_od(path, dirname):
    ret = []
    files = glob.glob(os.path.join(path, '*'))
    for f in files:
        if os.path.isdir(f):
            name = os.path.basename(f)
            if not name.startswith('_'):
                name_sq = SingleQuotedScalarString(name)
```

```
                od = OrderedDict()
                od[name_sq] = gen_pages_od(f, dirname + name + '/')
                ret.append(od)
        else:
            ret.append(gen_page_od(f, dirname))
    return ret

def main(argv=None):
    if argv is None:
        argv = sys.argv
    try:
        opts,args = getopt.getopt(sys.argv[1:], "h", ["help"])
    except getopt.error,msg:
        print msg
        print "for help use --help"
        return 2

    for o,a in opts:
        if o in ("-h","--help"):
            print __doc__
            return 0

    filename = 'mkdocs.yml'
    if not os.path.exists(filename):
        print 'error: not found file (mkdocs.yml)'
        return 0

    backup_filename = 'mkdocs.yml.bak'
    if os.path.exists(backup_filename):
        os.remove(backup_filename)

    filename = 'mkdocs.yml'
    shutil.copy2(filename, backup_filename)

    f = open('mkdocs.yml')
    data = ruamel.yaml.round_trip_load(f, preserve_quotes=True)
    f.close()

    pages = None;
    if len(args) == 0:
        pages = gen_pages_od('./docs/', '')
    else:
        pages = []
        for a in args:
            path = './docs/' + a
```

```
            if os.path.isdir(path):
                name_sq = SingleQuotedScalarString(a)
                od = OrderedDict()
                od[name_sq] = gen_pages_od(path, a + '/')
                pages.append(od)
            else:
                pages.append(gen_page_od(path, ''))

    f = codecs.open(filename, 'w+', 'utf-8')
    data['pages'] = pages
    f.write(ruamel.yaml.round_trip_dump(data, default_flow_style=False, encoding='utf-8', allow_unicode=True,
    explicit_start=False))
    f.close()

    return 0

if __name__ == '__main__':
    sys.exit(main())
```

使い方は次のようになります。

```
python gen-pages.py index.md dir1 dir2
```

この場合、次のようなフォルダ構成になっています。

```
project/
    gen-pages.py
    mkdocs.yml
    docs/
        index.md
        dir1/
            ...
        dir2/
            ...
```

11. PDF 出力

mkdocs は静的サイト用のコンテンツを作成する機能のみで、PDF 形式や他の電子書籍形式で出力する機能はありません。PDF 形式で保存するには何かしらの方法で mkdocs のコンテンツを変換する必要があります。mkdocs-pandoc を使うと、ドキュメントをもとに PDF や EPUB を作れるようです。その他に生成した静的サイトのコンテンツ(HTML)から PDF を作成する方法があります。ここでは HTML から PDF を生成するツール **wkhtmltopdf** を使って PDF を作成します。

11.1 wkhtmltopdf

まず wkhtmltopdf の Windows 版がありますので、インストールします。

wkhtmltopdf

wkhtmltopdf はコマンドラインで使うツールです。書式は次のようになっています。

```
wkhtmltopdf [GLOBAL OPTION]... [OBJECT]... <output file>
```

mkdocs で build すると site フォルダにコンテンツが作成されていますので、各記事の index.html を指定すればその記事の PDF ファイルを作成することができます。例えば、`site/hoge/index.html` の場合

```
wkhtmltopdf site/hoge/index.html output.pdf
```

これで output.pdf という名前で出力されます。

11.2 Material テーマでの出力

オプションを指定せずに出力してみるとレイアウトが崩れている状態となります。PDF で出力するときには、PDF 用(プリント用)に体裁を整える必要があります。

まず、Material テーマで表示されるページにはナビゲーション部分と、ページ内のアウトライン部分、本文の3つカラムに分かれています。PDF で出力するときはナビゲーション部分とアウトライン部分は不要ですので、css を使って表示されないようにします。カスタム CSS ファイルに以下を追加します。

```
@media print {
    .md-sidebar--primary, .md-sidebar--secondary {
        display: none;
    }
    .md-content {
        margin-left: 0;
    }
}
```

@media print を指定すると、印刷時のみ反映されます。.md-sidebar–primary はナビゲーション部分、.md-sidebar–secondary はアウトライン部分です。.md-content は本文で、ナビゲーションの部分を表示するために表示位置を右にずらしているため、margin-left:0 でリセットしています。

wkhtmltopdf で出力するときに印刷用のスタイルシートを反映させるため–print-media-type を指定します。また、追加のフォントやスタイルシート、Mathjax による数式なども完全に表示させるために、いくつかのオプションを追加します。

- –print-media-type
- –enable-javascript
- –no-stop-slow-scripts
- –javascript-delay

最終的には以下のようなコマンドを実行します。

```
wkhtmltopdf --print-media-type --enable-javascript --no-stop-slow-scripts --javascript-delay 15000 site/hoge/
index.html output.pdf
```

数式が不完全に出力されている場合は `--javascript-delay 15000` の数字を増やしてみてください。

試しにこの記事を mkdocs でビルドし、上記のコマンドで変換したものがこちらになります。

qiita-mkdocs.pdf

wkhtmltopdf にはページサイズなど細かい設定も可能です。詳しくは公式サイトなどを参照してください。複数の HTML ファイルを指定すれば、1 つのまとまった PDF として出力されます。

11.2.1 注意点

css や javascript を指定するときに http: や https: を省略して//fonts... としてしまうとエラーとなってしまい、正常に表示されなくなる場合がありますので、省略しないようにしましょう。

12. 印刷用 CSS

ブラウザの印刷機能に対応した CSS を用意すれば、紙に印刷したり、PDF に変換することもブラウザで簡単にできます。readthedocs テーマならそのままでもいいですが、materials テーマの場合は標準のままだと全体的に大きくなってしまうので調整しないといけません。

個人的に調整した印刷用 CSS は以下から取得できます。

qiita-mkdocs-print.css

この CSS を使用し、ブラウザの印刷機能で PDF に変換したものが以下から取得できます。

qiita-mkdocs-print.pdf

なお、wkhtmltopdf で PDF 出力するときにこの CSS を使うと全体的に小さくなってしまいますので、利用するときは注意してください。

13. 補足

(2019/3/14) Python 3.7.0, Mkdocs 1.0.4 で動作確認したところ, 一部変更があったのでメモしておきます.

13.0.1 pages ではなく nav になった

mkdocs.yml のページ列挙のところが pages から nav に変わったようです. pages は今のところ互換性はあるようですが, 将来的になくなるようです.

13.0.2 extensions のオプション記述ができなくなっていた

```
    markdown_extensions:
      - codehilite(linenums=true)
```

といった書き方が python 3 系になって認識できなくなったようです.その場合は

```
    markdown_extensions:
```

```
        - codehilite:
          linenums: true
```

と書きます