MkDocsによるドキュメント作成

MkDocsによるドキュメント作成

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 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。ちなみに私は 2.7.11 で確認しています。

次に pip を使って mkdocs をインストールします。

```
pip install mkdocs
```

プロジェクトの作成

実行に必要な最低限のフォルダとファイルを作成するコマンドが用意されています。例えば test というプロジェクトを作成する場合は以下のコマンドを実行します。

```
mkdocs new test
```

実行したカレントフォルダに test というフォルダが作られ、その中にも初期ファイルが作成されます。今後、mkdocs のコマンドを実行する場合は プロジェクトフォルダ（test）に作業ディレクトリにしておきます。

コンテンツのソースファイルは `docs` フォルダに入れます。（設定で別のフォルダに変更することが出来ます）

ビルド

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。

カスタマイズ

ここからがメインになります。デフォルトの状態でも悪くはないのですが、便利な機能を追加したり、見た目をもっと美しくしたりすることでそれなりに満足できるドキュメントが作成できるような環境を構築します。

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。

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 [http://squidfunk.github.io/mkdocs-material/]

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
[http://qiita.com/wamisnet/items/ed725d74f945f7c06b91]

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。

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。

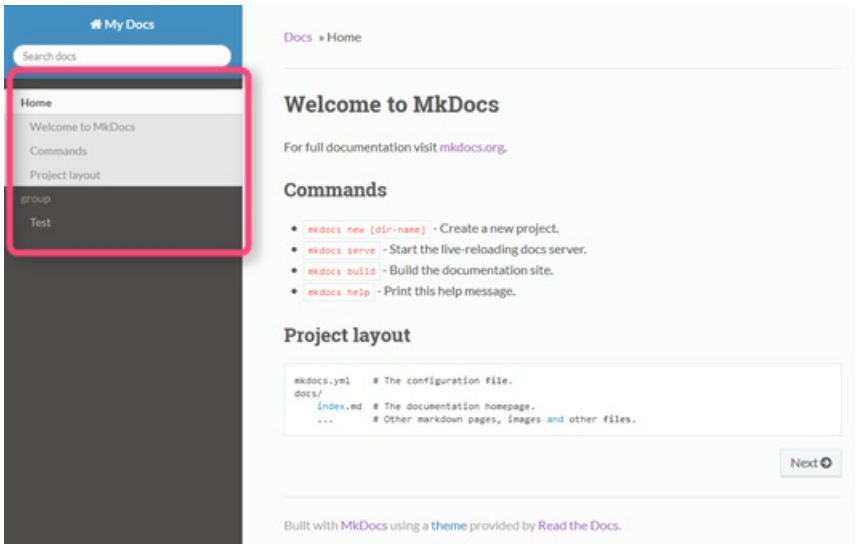
また、material テーマの場合は通常のフォントとコード用のフォントを別々に指定することが出来ます。

```
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。

デザインのカスタマイズ

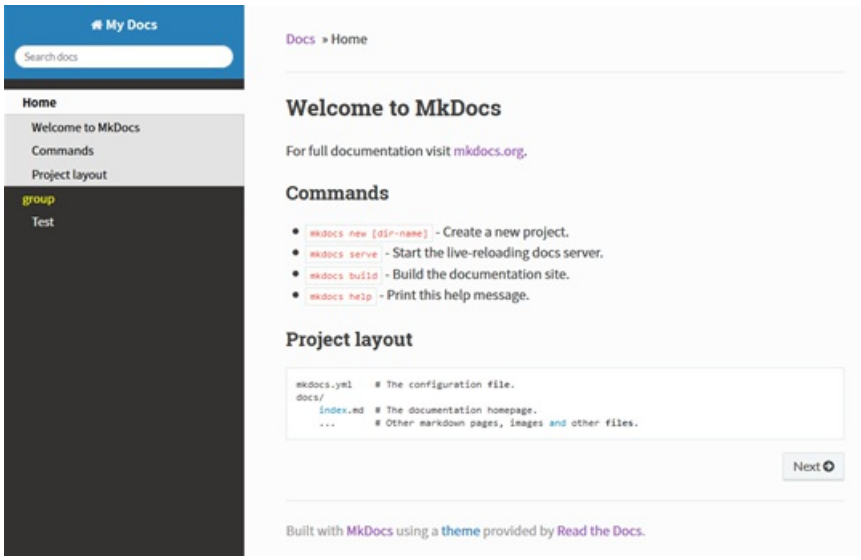
readthedocs のナビゲーション部分は個人的にビミョーに見づらく、もっとコントラストを強くしたほうがいい気がします。



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。



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。

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。

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
```

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。

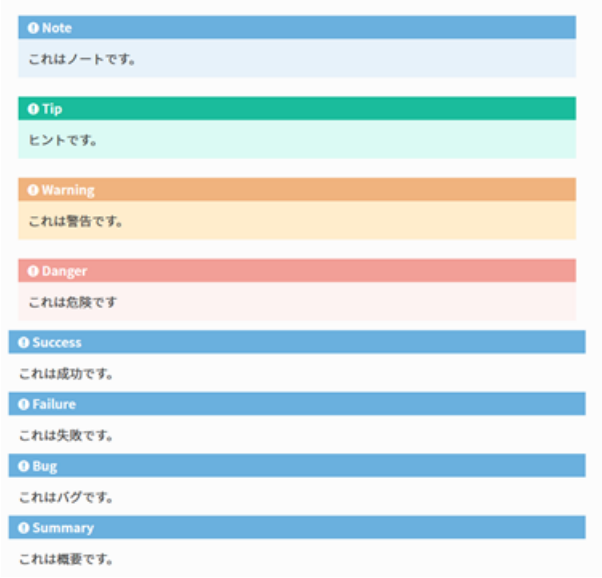
使う場合は mkdocs.yml で使うように指定します。

```
markdown_extensions:
    - admonition
```

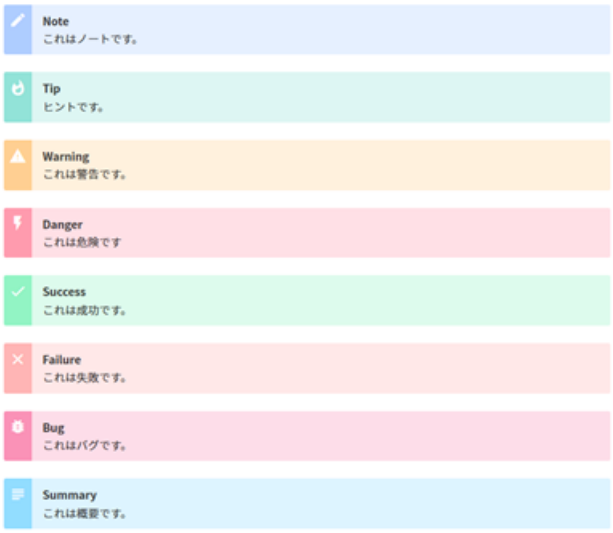
マークダウンでは `!!! Note` のように使います。 例えば以下のような内容の場合：

```
!!! Note
    これはノートです。

!!! Tip
    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
    これはバグです。

!!! summary
    これは概要です。
```

readthedocs テーマのとき



material テーマのとき



のように表示されます。

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。

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_extension:
    - fontawesome_markdown

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

マークダウンには `: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。

Font Awesome

MkDocs

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。

定義語

ここに説明です

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
```

ここに :smile: と記述すると 😄 になります

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 [https://www.webpagefx.com/tools/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。ちょっと調べてみましたが対処方法はわかりませんでした。他のテーマでも絵文字を使いたい場合は `mdx_unimoji` が使えます。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 [https://github.com/kernc/mdx_unimoji/blob/master/mdx_unimoji.py]

ここに `;)` と記述すると 😉 になります

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。

- `(tm)` : ™
- `(c)` : ©
- `(r)` : ®
- `c/o` : ℅
- `+/-` : ±
- `-->` : →
- `<--` : ←
- `<-->` : ↔
- `=/=` : ≠
- `1.4, etc.` : 1.4, etc.
- `1st 2nd etc.` : 1st 2nd etc.

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 [https://facelessuser.github.io/pymdown-extensions/extensions/smartsymbols/]

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。

`++ctrl+alt+delete++` と記述すると Ctrl + Alt + Del と表示されます。

詳細は以下を参照してください。

Keys [https://facelessuser.github.io/pymdown-extensions/extensions/keys/]

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+ [http://bwmarrin.github.io/MkDocsPlus/]

また、UML 図を挿入したい場合は yUML がオススメです。

yUML [https://yuml.me/]

これについ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
[http://qiita.com/tbpgr/items/a13114741d453d30188d]

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 [http://swfz.hatenablog.com/entry/2016/02/22/234434]

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 [https://github.com/MihaiValentin/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 テーマでは動作しましたが、material テーマでは動作しませんでした。いずれ動作出来たら書き直します。また、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。

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 [https://github.com/waylan/Python-Markdown/wiki/Third-Party-Extensions]

GitHub Pages

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/ [https://mebiusbox.github.io/MkDocsTest/]
- ソース：https://github.com/mebiusbox/MkDocsTest [https://github.com/mebiusbox/MkDocsTest]

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。

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。

ちょっと詰まったところ

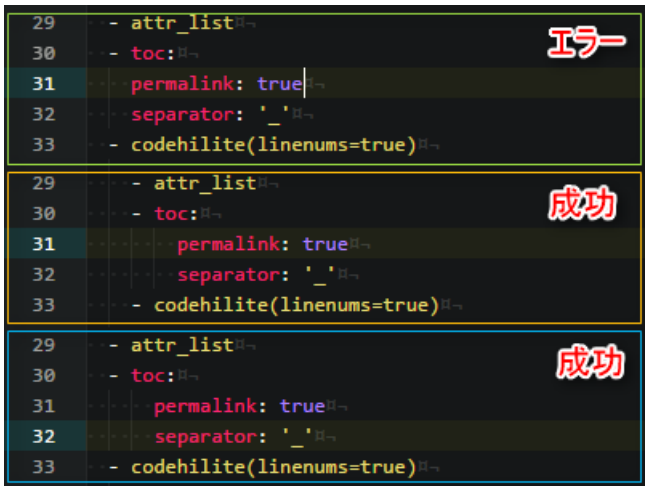
mkdocs.yml の設定で以下のようにするとエラーが表示されてしまいました。

```
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。



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
```

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
```

```
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_uni
    f.close()

    return 0

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

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

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。

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。

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。

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。

```python
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_uni
    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/
            ...
```

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。

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

wkhtmltopdf [https://wkhtmltopdf.org/]

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。

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 1500
```

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 [https://www.dropbox.com/s/avcz0224j9rfy0b/qiita-mkdocs.pdf?dl=0]

`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。

> **Note**
> 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。

最後に

PDF への変換については wkhtmltopdf で出来ますが、１つの html ファイルにまとめた状態で作成した方が意図した体裁で出力することが可能なので、何かしらの方法でまとめる必要がありそうです。pandoc を使って作成することも

可能みたいですが、texlive とか面倒そうなので見送っています。

他にも便利な機能や設定が多くあると思いますので、色々調べてみてはいかがでしょうか。