2020-02-23 1347329c

記事を Git で管理することにした

このサイトでは記事をデータベースに保存していたのですが、Git での管理に移行しました。 ここでは Markdown と Laravel で記事を Git 管理する方法をまとめます。

目次

1 Git と記事は相性が良い
2 FrontMatter でメタデータを管理する
3 Laravel での実装
  3.1 Markdown の解析
  3.2 記事一覧の取得
4 記事をキャッシュする

Git と記事は相性が良い

WordPress などの CMS には、下書き保存や承認フロー機能など、複数人で運営するための機能が実装されています。 また更新履歴を残せるため、誰がどこを書き換えたのか調査でき、以前のバージョンに戻すことも容易となっています。 このサイトはオリジナルのシステムで記事を管理していますが、このような機能を自力で実装するのはおおがかりです。

そこで、記事の管理をデータベースからファイルシステムに移行し、記事ディレクトリを Git で管理することにしました。 記事の管理機能と Git は共通点が多くあります。 たとえば、CMS の機能は以下のように代替可能です。

- 更新履歴 → コミットログ
- 下書き保存 → ブランチを切って作業
- 承認フロー →（GitHub の）Pull Request

さらに、次のようなメリットもあります。

- 記事をテキストエディタで編集するため、CMS に編集機能を実装する必要がない。
- 記事データをファイルとして配置するため、textlint や Prettier などのツールを適用しやすい。

というわけで、CMS での実装を減らしつつ記事管理を高機能にできます。

デメリットとしては、

- Git を非エンジニアに使わせるのは学習コストが高いかも。
- ファイルから記事データを読み込むため、読み込み負荷が非常に大きい。

が挙げられます。 視覚的に Git を操作できるツールを導入したりして、Git 独特の操作の難しさを解消する必要がありそうですね。 読み込み負荷の問題については後述します。

FrontMatter でメタデータを管理する

私は記事を Markdown で書いているのですが、記事のタイトルなどのメタデータはデータベースのカラムとして管理していました。 そこで、これらのメタデータも FrontMatter として記事本文の .md ファイルに埋め込みます。Markdown の先頭行に --- で囲んだ YAML データを定義できます。

```
ーーー (←半角に置き換える)
title: 記事を Git で管理することにした
description: 記事原稿を Git で管理するメリットや方法をまとめました。
thumbnail: /storage/articles/images/e7c0bb07.jpg
---

(記事本文)
```

記事の更新日時はファイルシステムを利用します。

Laravel での実装

まず、記事を管理する Git リポジトリを作成します。 記事データの配置は次のディレクトリ構成にします。

```
.
├─── README.md など
├─── (記事 ID)
|    └─── index.md
├─── (記事 ID)
|    └─── index.md
…
```

たとえば記事 ID staticgen、trimrate の記事を配置するには

```
.
├─── staticgen
|    └─── index.md
└─── trimrate
     └─── index.md
```

とします。 そして、Laravel のプロジェクトルートに記事の Git リポジトリをサブモジュールとして配置します。

```
(Laravel のプロジェクトルートにて)
$ git submodule add (記事リポジトリのアドレスやパス)
```

記事データを管理するクラスを app/Article.php として作成します。

```php
<?php

namespace App;

use Carbon\Carbon;
use cebe\markdown\Markdown;
use Hyn\Frontmatter\Parser;
use Hyn\Frontmatter\Frontmatters\YamlFrontmatter;

class Article
{
    public static function get(string $id): array
    {
        $articlesPath = base_path('apps-articles');
        $articlePath = "${articlesPath}/${id}/index.md";
        $file = file_get_contents($articlePath);

        $parser = new Parser(new Markdown());
        $parser->setFrontmatter(YamlFrontmatter::class);

        $article = $parser->parse($file);
        $article['meta']['id'] = $id;
        $article['meta']['updated_at'] = (new Carbon(filemtime($articlePath)))->format('Y-m-d');
        $article['meta']['hash'] = hash('crc32b', $file);

        return $article;
    }

    public static function getMetas(): array
    {
        $articlesPath = base_path('apps-articles');
        $articlePaths = glob("${articlesPath}/*/index.md");
        usort($articlePaths, function ($a, $b) {
            return filemtime($b) - filemtime($a);
        });

        $articles = [];
        foreach ($articlePaths as $articlePath) {
            $parser = new Parser(new Markdown());
            $parser->setFrontmatter(YamlFrontmatter::class);

            $article = $parser->parse(file_get_contents($articlePath));
            $meta = $article['meta'];
            $meta['id'] = str_replace('/index.md', '', str_replace($articlesPath . '/', "", $articlePath));
            $meta['updated_at'] = (new Carbon(filemtime($articlePath)))->format('Y-m-d');

            $articles[] = $meta;
        }

```

```
50.         return $articles;
51.     }
52. }
```

Markdown の解析

PHP にて Markdown の構文解析をし、HTML として出力するために cebe/markdown を利用しました。 また、YAML 形式の FrontMatter を解析するために hyn/frontmatter を併用します。 下記のプログラムによって、`$article['meta']` にメタデータが格納されます。 `$article['html']` には本文の HTML が格納されます。

```
1. $parser = new Parser(new Markdown());
2. $parser->setFrontmatter(YamlFrontmatter::class);
3. $article = $parser->parse(file_get_contents($articlePath));
```

記事一覧の取得

`getMetas()`では記事一覧を更新日順に取得しています。 `glob()` を使って、記事ファイルをワイルドカードで一括取得します。 得られたファイルパスの配列に対し、`filemtime()` で更新時刻の UNIX 時間を取得し、更新が新しい順にソートします。 あとは、各記事ファイルごとに Markdown を解析してメタデータを取得するだけです。

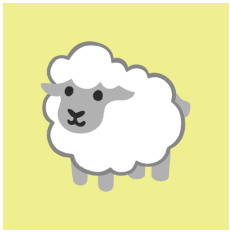
```
1. $articlesPath = base_path('apps-articles');
2. $articlePaths = glob("${articlesPath}/*/index.md");
3. usort($articlePaths, function ($a, $b) {
4.     return filemtime($b) - filemtime($a);
5. });
```

記事をキャッシュする

このプログラムは、サイトへのアクセスごとに記事ファイルを読み込んでいるため、サイトの速度などに影響がありそうです。 しかし、静的サイトの場合は問題ありません。（このサイトは静的です） 動的サイトの場合は、記事データをキャッシュしたりする必要があります。

記事リポジトリに CI を構築して、Redis やデータベースに自動デプロイするとよいかもしれません。 このプログラムを流用して、記事リポジトリを RAM ディスクに配置するなど対処法は割とあります。 または、先日耳にした microCMS のように、記事データをクライアントサイドでダウンロードして表示するのもモダンなやり方だと思います。

この記事の執筆者



Tomoyuki Hata Follow Follow @hata6502

中学生の頃に 6502 という CPU からプログラミングの世界に入りました。 C/C++ で1から作ることも好きですが、最近は Web 系に移行しつつあります。