

Groonga ではじめる全文検索

北市真 著

版 発行

MySQL で作る全文検索システム

Groonga は様々なソフトウェアに組み込み、全文検索機能を提供することができます。その一つ、MySQL に組み込むための Mroonga を使って、全文検索システムを作ってみましょう。

本書は、タイトルの通り、全文検索システムを作ったことがない人に向けて、PDF ファイルを対象とした簡単な全文検索アプリケーションを作成し、一通りシステムの完成までを案内しよう、という本です。

対象読者

本書は次のような人を主な対象と考えて執筆しています。

- 全文検索システムを作ってみたい
- PHP での開発ができる

本書ではアプリケーションを作成しながら進んでいきますが、飽くまで、一度完成までを体験してみるということを目的としているため、公開するには様々な物が足りていません。実際に全文検索システムを作成して不特定多数に使ってもらう場合には、ここで得た知識と感覚をもとに、改めて作るようにしてください。

第 1 章

Mroonga の概要

Mroonga は、MySQL のストレージエンジンの一つです^{*1}。MySQL のストレージエンジンには MyISAM や InnoDB などがあり、聞いたことがある人も多いでしょう。それぞれにパフォーマンスや耐障害性などに特徴があり、テーブルごとに使い分けることができます。

その一つとして Mroonga は、Groonga を組み込んで、高速かつ多機能な全文検索を可能にしたストレージエンジンです。これを MySQL に組み込んで使うことで、SQL を使って全文検索を行えるようになります。

Mroonga の特長を挙げてみましょう。

- SQL で全文検索機能を使用可能
- SQL の LIKE 句よりも高速
- SQL の LIKE 句よりも柔軟な検索が可能
- 検索結果を HTMLなどでハイライトすることが可能
- カラムごとに重み付けが可能
- MySQL のレプリケーション機能を用いでデータを入力することが可能

など、様々です。全てを本書で解説できるわけではありませんが、その一端にでも触れてもらえると幸いです。

Mroonga を全文検索システムの本番環境で使うには、[公式マニュアルのインストールのページ](#)を参照してください。ここでは、本書で試す目的で、簡易に環境を準備する方法を紹介します。

^{*1} MySQL から派生した MariaDB というデータベースシステムでも使用できます。最新の MariaDB では始めから Mroonga が組み込まれています。

第 2 章

Mroonga の環境の準備

本書では、MySQL 及びその他の環境の準備に Docker を使います。OS X El Capitan で確認していますが、異なる OS でも違いはないと思います。

以後、PHP と MySQL、Mroonga を使って PDF 文書の全文検索システムを作成していきますが、ラップトップなどの普段使っているシステムにインストール済みの物を使うと、バージョンが異なって本書の内容が当てはまらなかったり、また、将来別件で異なるバージョンをインストールした場合などに手間が増えてしまいます。設定ファイルの競合も問題になるかも知れません。Docker を使うと、システムの方に影響を与えずに、本書に必要なバージョンのソフトウェアを揃えることができます。不要になった場合には削除することも簡単で、その場合もシステムの方には影響を及ぼしません。

2.1 Docker のインストール

まず、Docker その物をインストールしましょう。Docker は OS X、Windows、Linux それぞれのプラットフォームで動作します。本番環境として動かすには Linux を使うべきですが、本書のようなお試し環境の共有目的ではどの環境で使っても問題ありません。ここでは、本書執筆の際にも使用した、OS X でのインストール方法を解説します。Linux、Windows でのインストールについては、それぞれ Docker 公式サイトでの解説を参照してください。

- Linux の場合 ... [Install Docker on Linux distributions](#)
- Windows の場合 ... [Get started with Docker for Windows](#)

OS X へのインストール

Docker 公式サイトでは [Get started with Docker for Mac](#) でインストール方法が解説されていますので、以下の方法でうまくいかない場合などには参照してください。

OS X へ Docker をインストールするには、公式サイトで配布されている Docker for Mac をインストールします。これは Docker 関連ツールをまとめたパッケージです。以下の配布ページから「Get Docker for Mac (stable)」と書かれたリンクからダウンロードし、インストーラーでインストールしてください。尚、Docker Toolbox のインストールには OS X 10.11 以上が必要です（2017 年 1 月現在）。

<https://docs.docker.com/docker-for-mac/#/download-docker-for-mac>

インストールが終わったら、Docker を起動します。アプリケーションディレクトリーで Docker をダブルタップするか、Spotlight で Docker を検索して実行してください^{*1}。するとメニューバーに Docker のクジラのアイコンが表示されます。これにより Docker の起動が確認できました。

2.2 Docker コンテナの起動

Docker を使用するには、必要なソフトウェアをインストールし、起動時の動作を定義した「イメージ」が必要です。イメージを取得した後に、そのイメージを雛形としてアプリケーションの動作環境となるコンテナを作ります。一つのイメージからは複数のコンテナを作成して使用することができます。

本書用に必要なソフトウェアをインストールした Docker イメージを作成してあります。これを元に皆さんのコンテナを作成し、その中で PHP を実行して全文検索システムを作っていきます。

まずはイメージを取得します。ターミナルで以下のコマンドを実行してください。

リスト 2.1 Docker イメージの取得

```
% docker pull kitaitimakoto/grnbook-mroonga
Using default tag: latest
latest: Pulling from kitaitimakoto/grnbook-mroonga
fdd5d7827f33: Pull complete
a3ed95cae02: Pull complete
0f35d0fe50cc: Pull complete
627b6479c8f7: Pull complete
67c44324f4e3: Pull complete
1429c50af3b7: Pull complete
8207a1b09d34: Pull complete
f4fd1f72cd2a: Pull complete
8c4074b3c552: Pull complete
d0e29d6de6ea: Pull complete
07672a754971: Pull complete
b0c92bbfd7a4: Pull complete
a42300873aad: Pull complete
2682e716a4cc: Pull complete
dd4fb0863e03: Pull complete
1cf726591e93: Pull complete
9435561d81cb: Pull complete
21da239ddc32: Pull complete
a563b423501c: Pull complete
Digest: sha256:b0c1a8cddde715ff679054eb8d22bf984fad29c285404d8365e135adf73d0bbd3
Status: Downloaded newer image for kitaitimakoto/grnbook-mroonga:latest
```

これにより、Docker Hub という Docker イメージ登録サービスから、本書用のイメージをダウンロードします。1GiB 以上あるので時間が掛かります。場合によってはネットワークを共有している同僚などに迷惑を掛けるかも知れませんが、周りの状況をよく見て実施しましょう。

次に、このイメージを元に、あなたのアプリケーションを作成し実施するための仮想環境「コンテナ」を作成します。本書サンプル用の適当なディレクトリーに移動し、以下のコマンドを実行します。

リスト 2.2 Docker コンテナの作成

```
% cd path/to/project
% docker create \
  --name=pdfsearch \
  --publish=8080:80 \
  --volume=$PWD:/var/lib/pdfsearch \
```

^{*1} 初回実行時には「Docker needs privileged access.」というウィンドウに続き、OS のパスワード入力が求められます。これは Docker が管理者権限を要する操作を行うためです。

```
kitaitimakoto/grnbook-mroonga
69277eea1becda07e4d6314485b6bd48fcfee065b23fa3692ad015d43d5f5c6f
```

コマンドを実行した結果表示される「69277eea1becda07e4d6314485b6bd48fcfee065b23fa3692ad015d43d5f5c6f」という行は、このコンテナの ID です。システム全体の中でそれぞれのコンテナを一意に特定するための物で、同じイメージから作成しても、別のコンテナには別の ID が割り振られます。今後、コンテナを操作する時にはこの ID を使用することができます。しかし、これを覚えるのは不可能ですし、一々確認のためのコマンドを実行するのも手間ですので、**name** オプションにより名前を付けました。以後は ID の代わりに、この名前 (**pdfsearch**) を使用していきます。なお、この名前もシステム内で一意である必要があり、後から同じ名前のコンテナを作ろうとしても失敗します。

publish オプションではポート番号のマッピングを定義しています。コロンの左側にホスト側 (Docker を実行している皆さんのコンピューター) のポート番号を書き、右側にコンテナ内のポート番号を書きます。コンテナ内では Apache HTTP Server が 80 番ポートをリッスンしますので、この場合はホスト側の 8080 番ポートにアクセスするとコンテナ内の Apache に接続できるようになっています。ホスト側のポート番号は他で使用している可能性もありますが、その場合には、次のように **publish** オプションで変更してください。

リスト 2.3 8000 番ポートを公開する例

```
% docker create \
  --name=pdfsearch \
  --publish=8000:80 \
  --volume=$PWD:/var/lib/pdfsearch \
  kitaitimakoto/grnbook-mroonga
```

また **volume** オプションによって、ホスト側とコンテナ内との共有ディレクトリーを指定しています。ここでもコロンの左側 (\$PWD) がホスト側のディレクトリー、右側 (/var/lib/pdfsearch) がコンテナ内のディレクトリーです。ここで指定したディレクトリー以外は、原則としてホスト側とは共有されません^{*2}。右側は、このディレクトリーが Apache のドキュメントルートとなるよう設定しているので、変更しないでください。

最後に、このコンテナを起動しましょう。**docker start** コマンドを使用します。

リスト 2.4 Docker コンテナの起動

```
% docker start pdfsearch
pdfsearch
```

実はここまでの **docker pull**、**create**、**start** をまとめて行う、**docker run** というコマンドがあります。

リスト 2.5 Docker イメージの取得とコンテナの起動

```
% docker run \
  --detach \
  --name=pdfsearch \
  --publish=8080:80 \
  --volume=$PWD:/var/lib/pdfsearch \
  kitaitimakoto/grnbook-mroonga
```

このように **docker run** コマンドを実行することで、以下の三つのことが行われます。

1. Docker イメージの取得 (もしシステム上に存在しない場合) (**docker pull**)

^{*2} 原則を外れることもできますが、本書の範囲外のため解説しません。気になる方は Docker の公式サイトなどで **volume** オプション及び「Dockerfile の **VOLUME** 命令」を調べてみてください。

2. 取得した Docker イメージを元にした Docker コンテナの作成 (`docker create`)
3. 作成した Docker コンテナの起動 (`docker start`)

先に説明した通り、`name` オプションを指定している場合は、複数回実行すると名前の重複によりコンテナの作成に失敗します。もし指定しなかった場合は、実行の度に新しいコンテナが作成・起動されます。コンテナを起動する二度目以降は `docker start` を使ってください。

行っていることの説明のため三つのステップで説明しましたが、実際には `docker run` コマンドを実行するのが簡単でしょう。

起動したコンテナを停止するには `docker stop` コマンドを実行します。

リスト 2.6 Docker コンテナの停止

```
% docker stop pdfsearch
pdfsearch
```

2.3 動作確認

Docker コンテナが実際に動作していることを確認しましょう。Docker コンテナを停止している場合には、`docker start pdfsearch` で再度起動してください。

このイメージには PHP が含まれているので、`docker run` または `create` を実行したディレクトリで以下のようなファイルを作成することで動作確認ができます。

リスト 2.7 info.php

```
<?php
phpinfo();
```

ブラウザで `http://localhost:8080/info.php` にアクセスしてみましょう。


| PHP Version 5.6.20-0+deb8u1  | |
|---|--|
| System | Linux 16c3696d1799 4.9.4-moby #1 SMP Wed Jan 18 17:04:43 UTC 2017 x86_64 |
| Build Date | Apr 27 2016 11:26:07 |
| Server API | Apache 2.0 Handler |
| Virtual Directory Support | disabled |
| Configuration File (php.ini) Path | /etc/php5/apache2 |
| Loaded Configuration File | /etc/php5/apache2/php.ini |
| Scan this dir for additional .ini files | /etc/php5/apache2/conf.d |
| Additional .ini files parsed | /etc/php5/apache2/conf.d/05-opcache.ini, /etc/php5/apache2/conf.d/10-pdo.ini, /etc/php5/apache2/conf.d/20-gd.ini, /etc/php5/apache2/conf.d/20-json.ini, /etc/php5/apache2/conf.d/20-mcrypt.ini, /etc/php5/apache2/conf.d/20-mysql.ini, /etc/php5/apache2/conf.d/20-mysqli.ini, /etc/php5/apache2/conf.d/20-pdo_mysql.ini, /etc/php5/apache2/conf.d/20-readline.ini |
| PHP API | 20131106 |
| PHP Extension | 20131226 |
| Zend Extension | 220131226 |
| Zend Extension Build | API220131226,NTS |
| PHP Extension Build | API20131226,NTS |
| Debug Build | no |
| Thread Safety | disabled |
| Zend Signal Handling | disabled |

図 2.1 phpinfo() の実行結果

PHP の情報が表示されれば、環境の準備は成功しています。

「Not Found」の画面が表示されてしまう場合は、どこかで間違えてしまったようです。これまでの手順を見ながらやり直してみてください。

`docker run` をやり直すには、以下のコマンドを実行して一旦 Docker コンテナを停止し、削除する必要があります。

1zwDocker コンテナの停止と削除

```
% docker stop pdfsearch
pdfsearch
% docker rm pdfsearch
pdfsearch
```

ありがちな間違いとして、Docker コンテナとの共有ディレクトリーの指定ミスがあります。`docker run` (`docker create`) コマンドの `volume` オプションをもう一度確認しましょう。`$PWD` は (`docker run` を実行した) 現在のディレクトリーを意味しますので、そこと別のディレクトリーにファイルを置いた場合は、コンテナ内の PHP が認識できません。事前にプロジェクトのディレクトリーに移動してから実行するようにしましょう。

第 3 章

作成する全文検索システムの概要

それでは、PHP と Mroonga を使って、全文検索システムを作っていきます。以下のようなシステムを作ることになります。

概要

登録済みの PDF ファイルを全文検索するシステムである

できること

検索対象となる文書をウェブ UI で登録することができる

ウェブ UI 上のテキストフィールドを使って検索することができる

実装方針

ウェブ UI は PHP を使用して作成する

全文検索用のデータは MySQL に格納する

PDF からテキストを抜き出す処理は本書では扱わない（Docker イメージ付属のツールを使う）

第 4 章

データ入力機能の作成

ここからアプリケーションを作っていきます。始めから検索機能に着手したいところですが、データがないことには何もできないので、入力機能から作成します。

PDF ファイルの内容を MySQL に登録するために、以下のようなステップを踏むことになります。

1. PHP によるウェブ UI から一時ディレクトリーにアップロードする
2. PDF ファイルからタイトルと本文テキストを抽出する
3. タイトルと本文を MySQL の適切なカラムに挿入する

4.1 PDF のアップロード機能を作る

まずは PDF のアップロード機能を作りましょう。PDF ファイルその物は後で使うことはないの
で、一時ディレクトリーに置いたままで構いません。

ユーザーが表示させる起点の画面を index.php、そのファイルから呼び出すアップロード機能の
部分を upload.php として実装することにします。

リスト 4.1 index.php

```
<?php
require_once __DIR__ . '/upload.php';

function h($string, $flags = ENT_QUOTES, $encoding = 'UTF-8')
{
    return htmlspecialchars($string, $flags, $encoding);
}

if ($_SERVER['REQUEST_METHOD'] === 'POST' && array_key_exists('pdf', $_FILES)) {
    $uploads = \PDFSearch\Upload::fromFileInfo($_FILES['pdf']);
}

?><!doctype html>
<title>PDF Search</title>
<h1>PDF Search</h1>

<h2>アップロードされたファイル</h2>
<?php if (! empty($uploads)): ?>
<?php foreach ($uploads as $upload): ?>
<p><?php echo h($upload->getName()); ?></p>
<?php endforeach; ?>
<?php endif; ?>

<form enctype="multipart/form-data" method="post">
    <input name="pdf[]" type="file" multiple>
    <input type="submit">
```

</form>

リスト 4.2 upload.php

```
<?php
namespace PDFSearch;

class Upload
{
    protected $name;
    protected $tmpName;

    public static function fromFileInfo(array $info)
    {
        $uploads = array();
        $fileCount = count($info['name']);
        for ($i = 0; $i < $fileCount; $i++) {
            $name = $info['name'][$i];
            $tmpName = $info['tmp_name'][$i];
            $uploads[] = new Upload($name, $tmpName);
        }
        return $uploads;
    }

    public function __construct($name, $tmpName)
    {
        $this->name = $name;
        $this->tmpName = $tmpName;
    }

    public function getName()
    {
        return $this->name;
    }
}
```

今の所、単にアップロードされたファイルの名前を表示しているだけです。後でその他の機能を実装していきます。複数のファイルをアップロードすると^{*1}、全てが表示されることも確認してみてください。

^{*1} ファイル選択ダイアログで Ctrl キーを押しながらファイルをクリックしていくことで、選んだファイルを同時にアップロードできます。また、Shift を押しながら最初のファイルと最後のファイルをクリックすることで、その間にあるファイルをすべて選択状態にできます。なお、PHP の設定により、合計で 1GiB を超えるアップロードは失敗します。



PDF Search

アップロードされたファイル

progit-ja.1016.pdf

未来の図書館を作るとは-1.0.1.pdf

ファイル選択 選択されていません 送信

図 4.1 ファイルアップロード画面

紙幅を減らすため、細かな検証は省略しています。実際に運用するアプリケーションでは安全性やエラーのチェックなどをしっかりと行ってください。

また、エラーが表示されてしまった場合など PHP のでバグが必要な場合は、本書末尾の「A.6 PHP のデバッグ」にデバッグ方法を記していますので、参考にしてください。

4.2 PDF からの情報を抽出する

次に、PDF ファイルからタイトルと本文を抜き出す処理を実装します。これを行うにはいくつか方法がありますが、ここでは **php-poppler** というライブラリーを使うことにします。php-poppler を使うと PDF を解析してタイトルや本文を抜き出すことができます^{*2}。

php-poppler は Poppler 名前空間にいくつかのクラスを定義しており、以下のように使います。

1zwphp-poppler の基本的な使い方

```
<?php
// pdfinfo コマンドのラッパー
$pdfinfo = \Poppler\Driver\Pdfinfo::create();
// pdftotext コマンドのラッパー
$pdftotext = \Poppler\Driver\Pdftotext::create();
// pdftohtml コマンドのラッパー
$pdftohtml = \Poppler\Driver\Pdftohtml::create();

$pdf = new \Poppler\Processor\Pdffile($pdfinfo, $pdftotext, $pdftohtml);

// PDF からタイトルや著者、作成日時などの情報を連想配列として抜き出す
echo $pdf->getInfo('path/to/document.pdf');
```

^{*2} C 製の PDF ライブラリーで Poppler という物があり、PDF を扱う様々なコマンドラインツールの提供もしています。php-poppler はこれらのコマンド呼び出しをラップして PHP から扱いやすくした物です。

```
// PDFから本文テキストを抜き出す
echo $pdf->toText('path/to/document.pdf');
```

HTML 関連の機能を使わないとしても、\Poppler\Processor\PdfFile のコンストラクターには \Poppler\Driver\Pdftohtml のインスタンスを渡す必要があるので注意してください。

Docker イメージの中には、既に Poppler と php-poppler がインストール済みです。php-poppler は Composer でインストールしているので、vendor/autoload.php を読み込むことで、オートロードが有効になります。

これを使って、先ほど作った \PDFSearch\Upload クラスに PDF 関連の機能を追加します。

リスト 4.3 upload.php

```
<?php
namespace PDFSearch;

class Upload
{
    // :
    // (省略)
    // :
    public function __construct($name, $tmpName)
    {
        $this->name = $name;
        $this->tmpName = $tmpName;
        $this->pdf = new \Poppler\Processor\PdfFile(
            \Poppler\Driver\Pdfinfo::create(),
            \Poppler\Driver\Pdftotext::create(),
            \Poppler\Driver\Pdftohtml::create()
        );
    }
    // :
    // (省略)
    // :
    public function getTitle()
    {
        $info = $this->pdf->getInfo($this->tmpName);
        return $info['Title'];
    }

    public function getContent()
    {
        return $this->pdf->toText($this->tmpName);
    }
}
```

これに合わせ、index.php の方も少し変更します。

リスト 4.4 index.php

```
<?php
// Dockerイメージ内にComposerでインストール済みのライブラリーを読み込む
require_once 'vendor/autoload.php';
require_once __DIR__ . '/upload.php';
// :
// (省略)
// :
<?php // getName() を getTitle() に 変更します。 ?>
<p><?php echo h($upload->getTitle()); ?>(<?php echo h($upload->getName()); ?>)</p>
<p><?php echo h(mb_substr($upload->getContent(), 0, 120)); ?>&hellip;&hellip;</p>
// :
// (省略)
// :
```

実際に PDF ファイルをアップロードして、タイトルなどの情報が取得できているか確認しましょう。



図 4.2 PDF 内の情報を取得

4.3 データベースにデータを挿入する

それでは、いよいよ Mroonga にデータを入力しましょう。と言っても、いつも通り PDO クラスで INSERT 文を実行するだけです。Mroonga の方で自動的に検索用のインデックスを作成してくれます。Docker イメージ内に既に MySQL 用のアダプターはインストールされていますので、単に PHP から呼び出すだけで使用できます。

データ挿入に必要な MySQL のデータベースとテーブルは、コンテナの中に既に作成してあります。構造は以下の通りです。

1zw テーブル作成時の全文検索インデックスの作成

```
CREATE TABLE `pdfs` (
  id INT PRIMARY KEY AUTO_INCREMENT,
  file VARCHAR(255),
  title VARCHAR(255),
  content LONGTEXT,
  FULLTEXT INDEX (title, content)
) ENGINE = Mroonga DEFAULT CHARSET utf8;
```

FULLTEXT INDEX や ENGINE = Mroonga については後述しますので、ここではカラム定義に注目してください。

このテーブルデータベースを扱う \PDFSearch\Table クラスを実装し、index.php でそのクラスを使うようにしたのが以下の内容です。 \PDFSearch\Upload には変更がないため省略しています。

リスト 4.5 index.php

```
<?php
// Dockerイメージ内にComposerでインストール済みのライブラリーを読み込む
require_once 'vendor/autoload.php';
require_once __DIR__ . '/upload.php';
require_once __DIR__ . '/table.php';

define('DBNAME', 'pdfsearch');

$dsn = 'mysql:host=localhost;dbname=' . DBNAME . ';charset=utf8';
```

```

$table = new \PDFSearch\Table($dsn, 'root', '');

function h($string, $flags = ENT_QUOTES, $encoding = 'UTF-8')
{
    return htmlspecialchars($string, $flags, $encoding);
}

if ($_SERVER['REQUEST_METHOD'] === 'POST' &&
    array_key_exists('pdf', $_FILES)) {
    $uploads = \PDFSearch\Upload::fromFilesInfo($_FILES['pdf']);
    try {
        foreach ($uploads as $upload) {
            $table->insert($upload);
        }
        header('Location:␣');
    } catch (\RuntimeException $e) {
        header('HTTP', true, 500);
        header('content-type:␣text/plain');
        echo $e;
    }
    exit;
}

$records = $table->records();

?><!doctype html>
<title>PDF Search</title>
<h1>PDF Search</h1>

<h2>登録済みPDF一覧</h2>
<table>
    <tr>
        <th>ファイル名</th>
        <th>タイトル</th>
        <th>内容</th>
    </tr>
    <?php foreach ($records as $record): ?>
        <tr>
            <td><?php echo h($record['file']); ?></td>
            <td><?php echo h($record['title']); ?></td>
            <td><?php echo h(mb_substr($record['content'], 0, 120)); ?>&hellip;&hellip;</td>
        </tr>
    <?php endforeach; ?>
</table>

<form enctype="multipart/form-data" method="post">
    <input name="pdf[]" type="file" multiple>
    <input type="submit">
</form>

```

リスト 4.6 table.php

```

<?php
namespace PDFSearch;

class Table
{
    const INSERT =
        "INSERT INTO␣'pdfs'␣(file,␣title,␣content)
        VALUES(:file,␣:title,␣:content);";
    const SELECT = "SELECT␣*␣FROM␣'pdfs'␣ORDER BY␣'id'";

    protected $pdo;

    public function __construct($dsn, $username, $password)
    {
        $this->pdo = new \PDO($dsn, $username, $password);
    }

    public function insert(Upload $upload)
    {
        $params = [
            ':file' => $upload->getName(),
            ':title' => $upload->getTitle(),
            ':content' => $upload->getContent()
        ];
        $sth = $this->pdo->prepare(self::INSERT);
        $isSucceeded = $sth->execute($params);
        if (!$isSucceeded) {

```

```
        throw new \RuntimeException(implode(PHP_EOL, $sth->errorInfo()));
    }
}

public function records()
{
    $sth = $this->pdo->prepare(self::SELECT);
    $isSucceeded = $sth->execute();
    if (! $isSucceeded) {
        throw new \RuntimeException(implode(PHP_EOL, $sth->errorInfo()));
    }
    return $sth->fetchAll();
}
}
```

ウェブブラウザでアクセスして、データベースの中身を表示してみましょう。

PDF Search

登録済みPDF一覧

| ファイル名 | タイトル | 内容 |
|-----------------------|-------------------|---|
| progit-ja.1016.pdf | Pro Git | This work is licensed under the Creative Commons AttributionNonCommercial-ShareAlike 3.0 Unported License. To view a cop…… |
| 未来の図書館を作るとは-1.0.1.pdf | 未来の図書館を作るとは 長尾真 著 | 未来の図書館を作るとは makoto nagao LRG 編 未来の図書館を作るとは 長尾真 著 2014-07-03 版 達人出版会 発行 本書のライセンスについては-1.0.1.pdf るとは ついて 本書に掲載されている翻訳は、クリエイティ…… |
| ファイル選択 | | 選択されていません |
| | | 送信 |

図 4.3 アップロードした PDF の情報を表示

第 5 章

全文検索機能の作成

ようやく、全文検索機能の実装です。実は、これも Mroonga ならではの特別なことは必要なく、MySQL の通常の `MATCH(...)` `AGAINST(...)` 構文を使用するだけです。テーブル作成時に、ストレージエンジンに Mroonga を選択しておけば、他の操作は通常の MySQL と同じように実行できるようデザインされているのです。

`MATCH(...)` `AGAINST(...)` 構文は元々 MySQL に備わっている全文検索用の構文です。`MATCH()` には検索対象にするカラム名を渡します。ここで指定できるのは、テーブル作成時に `FULLTEXT` インデックスを指定したカラムのみです。Docker 内のテーブルでは、予め `title` カラムと `content` カラムのセットに `FULLTEXT` インデックスを張っています。`AGAINST()` には検索クエリーを渡します。オプションを渡すこともできます。詳細は [MySQL のドキュメント](#) を参照してください。

Mroonga ではこの構文を流用しており、元の挙動を知っている人にはおおそ期待通りに動作するようになっていますが、厳密に同じではありません。Mroonga でのみ使える様々な機能が使えるようになっています。これについては、一旦基本の検索機能を作ったあとで触れたいと思います。

5.1 一検索語用の実装

今の「登録済み PDF 一覧」の上に検索フォームを作成し、検索を実行した場合にはフォームの下に検索結果を表示するようにしてみましょう。そのために、`\PDFSearch\Table` クラスに検索用のメンバー関数を追加します。

リスト 5.1 table.php

```
<?php
namespace PDFSearch;

class Table
{
    const INSERT =
        "INSERT INTO `pdfs` (`file`,`title`,`content`)
        VALUES (:file,`:title`,`:content)`";
    const SELECT = "SELECT * FROM `pdfs` ORDER BY `id`";
    const SEARCH = <<<EOS
SELECT * FROM `pdfs` WHERE MATCH(title, content) AGAINST(:query IN BOOLEAN MODE);
EOS;

    protected $pdo;
    // :
    // :
    public function search($query)
    {
```

```

        $sth = $this->pdo->prepare(self::SEARCH);
        $params = ['query' => $query];
        $isSucceeded = $sth->execute($params);
        if (! $isSucceeded) {
            throw new \RuntimeException(implode(PHP_EOL, $sth->errorInfo()));
        }
        return $sth->fetchAll();
    }
}

```

SEARCH 定数と search メンバー関数を追加しています。

先ほど言ったような検索フォームと結果のエリアを追加すると、index.php はこうなります。

リスト 5.2 index.php

```

<?php
// :
// :
// 検索処理
$searchQuery = null;
$searchResult = null;
if ($_SERVER['REQUEST_METHOD'] === 'GET' &&
    array_key_exists('q', $_GET) && $_GET['q']) {
    $searchQuery = $_GET['q'];
    $searchResult = $table->search($searchQuery);
}
// :
// :
<h2>検索</h2>
<form method="get">
    <input name="q" type="search" value="<?php echo h($searchQuery); ?>">
    <input type="submit" value="検索">
</form>

<?php if ($searchResult && count($searchResult) > 0): ?>
<h2>「<?php echo h($searchQuery); ?>」を含むPDF</h2>
<table>
    <tr>
        <th>ファイル名</th>
        <th>タイトル</th>
        <th>内容</th>
    </tr>
    <?php foreach ($searchResult as $record): ?>
        <tr>
            <td><?php echo h($record['file']); ?></td>
            <td><?php echo h($record['title']); ?></td>
            <td><?php echo h(mb_substr($record['content'], 0, 120)); ?>&hellip;&hellip;</td>
        </tr>
    <?php endforeach; ?>
</table>
<?php endif; ?>

```

SQL 一文で済むので簡単ですね。では、検索してみましょう。

PDF Search

検索

「検索エンジン」を含むPDF

| ファイル名 | タイトル | 内容 |
|-----------------------|-------------|--|
| 未来の図書館を作るとは-1.0.1.pdf | 未来の図書館を作るとは | 長尾真 著 未来の 図書館を 作るとは makoto nagao LRG 編 未来の図書館を作るとは 長尾真 著 2014-07-03 版 達人出版会 発行 本書のライセンスについて 本書に掲載されている翻訳は、クリエイティ…… |

登録済みPDF一覧

| ファイル名 | タイトル | 内容 |
|--|------|----|
| This work is licensed under the Creative Commons | | |

図 5.1 全文検索結果の表示

以上で、基本的な機能は完成です。

5.2 AND 検索の実装

一単語で検索している間は気になりませんが、検索フォームに二単語入れると、検索結果に違和感を覚えるかも知れません。例えば「Commons ライセンス」で検索した場合に、『Commons』は含んでいるが、『ライセンス』は含んでいない」という文書もヒットする（OR 検索になっている）ためです。こういう結果が望ましい場面もありますが、Google 検索はこうではなく「Commons ライセンス」で検索した場合には『Commons』も『ライセンス』も両方とも含んでいるページ」が検索結果上位に登場するようになっています（AND 検索になっている）。Google 検索には非常にたくさんの方が馴染んでいるため、どちらの挙動でもいい場合には合わせておくのがいいでしょう。

Mroonga でこのような挙動になっているのは、MySQL のブーリアンモードによる全文検索が、デフォルトでこのようになっているためです。AND 検索になるよう、何らかの処理を追加する必要があります。

方法は二つあり、一つは、PHP でクエリーパラメーター（`$_GET['q']`）を単語に分割し、それぞれに AND 検索用の命令を追加していく方法です。Mroonga に限らず MySQL の全文検索でも使える汎用的な方法ですが、SQL インジェクション脆弱性などを作り込まずに PHP を書いていく必要があります、本書のスコープ外になるためここでは採用しません。

もう一つは、SQL のクエリーを実行するタイミングで、クエリー内容の解釈を変更させる「プラグマ」という Mroonga 特有の命令を使う方法です。プラグマは `AGAINST()` 句内で使用し、次のよ

うになります。

1zw プラグマを追加したクエリー

```
AGAINST('*D+_Commons_ライセンス' IN BOOLEAN MODE)
```

検索語の先頭に追加されたのがプラグマです。以下のような構文になっています。

- 「*」で開始する
- プラグマの種類を指定する。種類には「D」や「W」などがある
- 種類の後に、そのプラグマの詳細な挙動を指定するパラメーターを指定する（指定方法はプラグマごとに異なる）。ここでは「+」を指定している
- 検索語の前に空白文字を置く

検索を AND 検索にするには「D+」というプラグマを使用します。「D」によって「複数の検索語を AND で扱うのか OR なのか NOT なのか」という検索語の扱いを指定する、と宣言し、「+」を使うことで「AND 検索として扱う」ということを指定します。「+」の部分を「-」など他のキーワードに変えることで別の指定もできますが、ここでは扱いません。

本書の範囲を外れる詳細については、Mroonga の[ブーリアンモード](#)のマニュアルページを参照してください。

それでは実際に、これを使って AND 検索を実装しましょう。\\PDFSearch\\Table\\SEARCH 定数の SQL と、その中のパラメーターをバインドする変数を変更します。

リスト 5.3 table.php

```
<?php
namespace PDFSearch;

class Table
{
    // :
    // :
    const SEARCH = <<<EOS
SELECT * FROM 'pdfs' WHERE MATCH(title, content) AGAINST(:query_with_pragma IN BOOLEAN MODE
);
EOS;
    // :
    // :
    public function search($query)
    {
        $sth = $this->pdo->prepare(self::SEARCH);
        $params = [':query_with_pragma' => '*D+' . $query];
    }
    // :
    // :
```

これで「Commons ライセンス」と検索した場合に、両方を含む文書しかヒットしないようになります。

次章では、Mroonga に特徴的な幾つかの機能を紹介し、実装していきます。

第 6 章

Mroonga ならではの機能の作成

ここからは、MySQL の全文検索機能にはない、Mroonga ならではの特徴を活かした機能を実装していきます。

6.1 検索語のハイライト

現在の検索機能では、検索結果中、どこに検索語があるのか、自分で探さなくてはなりません。次の画像のように、検索語だけがハイライトされていると、とても見やすくなります。また、菌類についての PDF を探すつもりで「きのこ」で検索し「お気づきのこと」といった関係ない語がヒットした場合に、すぐにそのことに気付けるというメリットもあります。

検索

「きのこ」を含むPDF

| ファイル名 | タイトル | 内容 |
|--------------------|---------|--|
| progit-ja.1016.pdf | Pro Git | ットとするつもりだったのに間違えて git add * と打ち込んでしまったときのことを考えましょう。ファイルが両方ともステージされてしまいました。細は、“何が変わるのかの把握”を参照ください。もうひとつお気づきのことがあることでしょう。GitHub は、このプルリクエストが問題なくマー |

図 6.1 検索語をハイライトして表示

このように検索語をハイライトする機能を実装してみましょう。

すぐに思い付く実装方法は、取得した検索結果の `content` カラムを、再度 PHP で解析してタグを挿入することですが、実は Mroonga には、このための機能が備わっています。SQL 中で利用可能な `mroonga_snippet_html()` 関数です。これは特定のカラム（ここでは `content`）のうち、指定した語（「cat」）の前後数十バイトを抜き出し、文字列として返す関数です。

`mroonga_snippet_html()` 関数が返すスニペットは `<div class="snippet">...</div>` という

タグで囲まれ、更に検索語は...で囲まれます*1。一レコード中に検索語が複数回現れる場合は、その数に応じて複数のスニペットを結合した一つの文字列を返します。余談ですが「スニペット」は「断片」という意味で、この場合は content カラムのうちの一部を返すことを意味しています。

mroonga_snippet_html() は以下のようにして使用します。

lzwmmroonga_snippet_html の簡単な使い方

```
SELECT mroonga_snippet_html(content, 'きのこ') FROM 'pdfs';
```

結果は、例えば次のようになります*2。

lzwmmroonga_snippet_html() の結果例

```
<div class="snippet">ットとするつもりだったのに間違えて git add * と打ち込んでしまったと<span class="keyword">きのこ</span>とを考えましょう。 ファイルが両方ともステージされてしまいました</div><div class="snippet">細は、“何が変わるのかの把握” を参照くださいもうひとつお気づ<span class="keyword">きのこ</span>とがあることでしょう。 GitHub は、このプルリクエストが問題なくマー</div>
```

ここでは、本書のアプリケーションに必要なことのみ説明しているので、詳細については公式ドキュメント (5.5.5. mroonga_snippet_html()) を参照してください。

現在の実装では検索でヒットした時に content カラムの抜粋を表示していますが、mroonga_snippet_html() の戻り値で置き換えてみましょう。PDFSearch\Table::search() で使用している SQL (PDFSearch\Table::SEARCH 定数) を変更します。mroonga_snippet_html() も COUNT() などと同様の関数なので、AS 句を使って別名を付けることができます。

リスト 6.1 table.php

```
<?php
namespace PDFSearch;

class Table
{
    const INSERT =
        "INSERT INTO `pdfs` (file, title, content)
        VALUES (:file, :title, :content);";
    const SELECT = "SELECT * FROM `pdfs` ORDER BY `id`";
    const SEARCH = <<<EOS
SELECT file, title, mroonga_snippet_html(content, :query AS query) AS snippets
FROM `pdfs` WHERE MATCH(title, content) AGAINST(:query IN BOOLEAN MODE);
EOS;

    protected $pdo;

    public function __construct($dsn, $username, $password)
    {
        $this->pdo = new \PDO($dsn, $username, $password);
    }

    public function insert(Upload $upload)
    {
        $params = [
            ':file' => $upload->getName(),
            ':title' => $upload->getTitle(),
            ':content' => $upload->getContent()
        ];
        $sth = $this->pdo->prepare(self::INSERT);
        $isSucceeded = $sth->execute($params);
        if (!$isSucceeded) {
            throw new \RuntimeException(implode(PHP_EOL, $sth->errorInfo()));
        }
    }
}
```

*1 タグや class 属性を変更したい場合は、柔軟性の高い mroonga_snippet() 関数を使用してください。

*2 読みやすさのため、改行を調整しています。

```

public function records()
{
    $sth = $this->pdo->prepare(self::SELECT);
    $isSucceeded = $sth->execute();
    if (! $isSucceeded) {
        throw new \RuntimeException(implode(PHP_EOL, $sth->errorInfo()));
    }
    return $sth->fetchAll();
}

public function search($query)
{
    $sth = $this->pdo->prepare(self::SEARCH);
    $params = [':query' => $query];
    $isSucceeded = $sth->execute($params);
    if (! $isSucceeded) {
        throw new \RuntimeException(implode(PHP_EOL, $sth->errorInfo()));
    }
    return $sth->fetchAll();
}
}

```

HTML 中で `content` カラムを表示していた所を、`snippets` に置き換えます。

リスト 6.2 index.php

```

<?php
// Dockerイメージ内にComposerでインストール済みのライブラリーを読み込む
require_once 'vendor/autoload.php';
require_once __DIR__ . '/upload.php';
require_once __DIR__ . '/table.php';

define('DBNAME', 'pdfsearch');

$dsn = 'mysql:host=localhost;dbname=' . DBNAME . ';charset=utf8';
$table = new \PDFSearch\Table($dsn, 'root', '');

function h($string, $flags = ENT_QUOTES, $encoding = 'UTF-8')
{
    return htmlspecialchars($string, $flags, $encoding);
}

// 検索処理
$searchQuery = null;
$searchResult = null;
if ($_SERVER['REQUEST_METHOD'] === 'GET' &&
    array_key_exists('q', $_GET) && $_GET['q']) {
    $searchQuery = $_GET['q'];
    $searchResult = $table->search($searchQuery);
}

// ファイルアップロード処理
if ($_SERVER['REQUEST_METHOD'] === 'POST' &&
    array_key_exists('pdf', $_FILES)) {
    $uploads = \PDFSearch\Upload::fromFilesInfo($_FILES['pdf']);
    try {
        foreach ($uploads as $upload) {
            $table->insert($upload);
        }
        header('Location:␣');
    } catch (\RuntimeException $e) {
        header('HTTP', true, 500);
        header('content-type:␣text/plain');
        echo $e;
    }
    exit;
}

$records = $table->records();

?><!doctype html>
<title>PDF Search</title>
<style>
    .keyword {
        font-weight: bold;
        color: red;
    }
</style>

```

```

<h1>PDF Search</h1>

<h2>検索</h2>
<form method="get">
  <input name="q" type="search" value="<?php echo h($searchQuery); ?>">
  <input type="submit" value="検索">
</form>

<?php if ($searchResult && count($searchResult) > 0): ?>
<h2>「<?php echo h($searchQuery); ?>」を含むPDF</h2>
<table>
  <tr>
    <th>ファイル名</th>
    <th>タイトル</th>
    <th>内容</th>
  </tr>
  <?php foreach ($searchResult as $record): ?>
    <tr>
      <td><?php echo h($record['file']); ?></td>
      <td><?php echo h($record['title']); ?></td>
      <td><?php echo $record['snippets']; ?></td>
    </tr>
  <?php endforeach; ?>
</table>
<?php endif; ?>

<h2>登録済みPDF一覧</h2>
<table>
  <tr>
    <th>ファイル名</th>
    <th>タイトル</th>
    <th>内容</th>
  </tr>
  <?php foreach ($records as $record): ?>
    <tr>
      <td><?php echo h($record['file']); ?></td>
      <td><?php echo h($record['title']); ?></td>
      <td><?php echo h(mb_substr($record['content'], 0, 120)); ?>&hellip;&hellip;</td>
    </tr>
  <?php endforeach; ?>
</table>

<form enctype="multipart/form-data" method="post">
  <input name="pdf[]" type="file" multiple>
  <input type="submit">
</form>

```

snippets の表示では HTML エスケープをしていないことに気が付いたでしょうか。mroonga_snippet_html() に HTML タグが含まれているため、ここで HTML エスケープしてしまうと、タグとしての役を果たさなくなってしまう。そのため、直接出力しています。

ここにはセキュリティ上の懸念があります^{*3}。mroonga_snippet_html() の戻り値では、カラム中の HTML タグなどは全てエスケープされますが、取得したスニペットと他の文字列を PHP で結合すると、セキュリティホールになり得ます。加工せず、可能であれば単独で出力するようにしてください。加工や他の文字列との結合が必要な場合は、慎重に行ってください。

これで、検索結果のハイライトも出来ました。実際に検索して結果を楽しみましょう。

6.2 スコアによる並び替え

これまでの、検索結果の並び順については考慮していませんでした。やはり、検索語に近い PDF が先頭に並んでほしいものです。

この「検索語に近い」または「より望ましい」という状態を、Mroonga ではスコアという形で、

^{*3} 実際には HTML を正しく出力するために必要な注意点です。仮にセキュリティ上問題とならなくても、HTML が正しく出力されず、アプリケーションが壊れてしまう可能性があります。

数値として計算することができます。「検索語がより多く含まれている PDF の方が望ましい」「検索語が本文に含まれているよりもタイトルに含まれている方がより望ましい」といった判断が可能になるのです。

Mroonga が使っているスコアの値は、WHERE 句に指定している MATH()...AGAINST() の戻り値として取得できます。AS で別名を付けると使いやすいでしょう。

1zw スコアの取得

```
SELECT MATCH(title, content) AGAINST('cat' IN BOOLEAN MODE) as score FROM 'pdfs';
```

1zw

```
+-----+
| score |
+-----+
|    315 |
|     10 |
|      0 |
|      0 |
|      4 |
+-----+
5 rows in set (0.47 sec)
```

このスコアでソートすることで、より望ましい結果を上の方に表示することができます。

では、このスコアというのは何によって決められているのでしょうか。一つは、結果文書の中に含まれている検索語の数です。検索語が多く含まれている文書ほど検索する人の要求に合っていると見做して、検索上位に表示させられるわけです。

以下の用に PDFSearch\Table::SEARCH の SQL にちょっとした変更を加えるだけで、多くの場合、期待される検索結果になるでしょう。

1zw 変更前の\PDFSearch\Table::SEARCH 定数

```
SELECT file, title, mroonga_snippet_html(content, :query) AS snippets
FROM 'pdfs' WHERE MATCH(title, content) AGAINST(:query IN BOOLEAN MODE);
```

1zw 変更前の\PDFSearch\Table::SEARCH 定数

```
SELECT file, title, mroonga_snippet_html(content, :query) AS snippets,
MATCH(title, content) AGAINST(:query IN BOOLEAN MODE) AS score
FROM 'pdfs' WHERE MATCH(title, content) AGAINST(:query IN BOOLEAN MODE)
ORDER BY score DESC;
```

もう一つの基準は重みと呼ばれます。重みは、どのカラムをより重視するかという指標です。例えば、次のような二つの PDF が登録されているとします。

表 6.1 Groonga という語を含む二つの PDF

| タイトル | 内容 |
|---------------------|---|
| Groonga で作る全文検索システム | Groonga を使って全文検索システムを作ります。 |
| わたしのエッセイ集 | 今日、Groonga という言葉を耳にしました。不思議な言葉ですね、Groonga。検索してみ |

Groonga のことが知りたくて検索する時に、どちらがヒットしてほしいでしょうか。始めの「Groonga で作る全文検索システム」の方ではないでしょうか。でも、「Groonga」という言葉が多く含まれているのは「わたしのエッセイ集」の方なので、今までの実装だと、こちらが上位に表示されることになってしまいます。前者を上位に持って来るために、「検索語がタイトルに含まれてい

る方が、本文に含まれているよりも 100 倍くらい重要だ」という指標を導入することにしましょう。Mroonga または全文検索の言葉で「title カラムの重みを content カラムの 100 倍にする」ということになります。

| 「すべて」を含むPDF | | |
|--|--|--------|
| ファイル名 | タイトル | 内容 スコア |
| まつもとゆきひろ直伝 組込Ruby「mruby」のすべて総集編-1.0.0.pdf | まつもとゆきひろ直伝 組込向けRuby mrubyのすべて | 131 |
| APIデザインケーススタディ——Rubyの実例から学ぶ。問題に即したデザインと普遍の考え方_00.pdf | APIデザインケーススタディ—Rubyの実例から学ぶ。問題に即したデザインと普遍の考え方 | 49 |
| Dockerエキスパート養成読本 [活用の基礎と実践ノウハウ満載!] _00.pdf | Docker エキスパート養成読本 [活用の基礎と実践ノウハウ満載!] | 26 |

図 6.2 検索語がタイトル含まれている場合スコアが大きくなる

上の画像にある三つの PDF ではどれも本文中に「すべて」という語が含まれますが、タイトルにも含まれる「mruby のすべて」のスコアが、含まれない「API デザインケーススタディ」「Docker エキスパート養成読本」に比べ、非常に大きくなっていることが分かります。実際、タイトルの重視をやめると、「mruby のすべて」と「API デザインケーススタディ」の順位は逆転します。

このように重み付けするには、W プラグマを使用します。

```
MATCH(title,content) AGAINST('W1:100,2:1API')
```

W が重み付けのためのプラグマ使用を指示し、その後はカラムごとの重み付けをカンマ区切りで指定します。「1:100」は「(MATCH で指定した) 1 カラム目の重みを 100 に」、「2:1」は「2 カラム目の重みを 1 に」という命令になります。これにより、「検索語が 1 つタイトルに含まれていると、本文に 100 個含まれているのと同等のスコアになる」ということを実現しています（実際には 100 倍の差が付きさえすればいいので、2 と 200 でも、10 と 1000 でも構いません）。

まとめると、実装は次のようになります。

リスト 6.3 table.php

```
<?php
namespace PDFSearch;

class Table
{
    const INSERT =
        "INSERT INTO `pdfs` (file, title, content)
        VALUES (:file, :title, :content);";
    const SELECT = "SELECT * FROM `pdfs` ORDER BY `id`";
    const SEARCH = <<<EOS
SELECT file, title, mroonga_snippet_html(content, :query AS query) AS snippets,
MATCH(title, content) AGAINST(:query_with_pragma IN BOOLEAN MODE) AS score
FROM `pdfs` WHERE MATCH(title, content) AGAINST(:query_with_pragma IN BOOLEAN MODE)
ORDER BY score DESC;
EOS;
    // :
    // :
    public function search($query)
    {
```

```
$sth = $this->pdo->prepare(self::SEARCH);
$params = [
    ':query' => $query,
    ':query_with_pragma' => '*D+W1:100,2:1□' . $query
];
$isSucceeded = $sth->execute($params);
if (! $isSucceeded) {
    throw new \RuntimeException(implode(PHP_EOL, $sth->errorInfo()));
}
return $sth->fetchAll();
}
```

W プラグマを色々な値に変えて試してみてください。本書のようなチュートリアルではなく、実際の全文検索システムに置いてもスコアリングは非常に大切なポイントです。重み付けを使いこなしたスコアリングができると、いかにも全文検索システムを作っているという実感が出て、楽しいと思います。

付録 A

付録

A.1 Mroonga のインストール

Mroonga を使った全文検索システムに興味が出て来たら、ぜひ自分でインストールしてみてください。以下に、典型的なインストール方法をご案内します。

Debian/GNU Linux

Groonga プロジェクト（？）は、本書執筆時点で Debian/GNU Linux jessie の APT 用のリポジトリを持っており、Mroonga もそこからインストールできます。サンプルで用いた Docker イメージでもこのパッケージを使用しています。

まず、以下の内容で `/etc/apt/sources.list.d/groonga.list` というを作成し、ソースを登録します。

```
lzw/etc/apt/sources.list.d/groonga.list
```

```
deb http://packages.groonga.org/debian/ jessie main
deb-src http://packages.groonga.org/debian/ jessie main
```

次にインストールします。

```
lzw
```

```
% sudo apt-get update
% sudo apt-get install -y --allow-unauthenticated groonga-keyring
% sudo apt-get update
% sudo apt-get install -y -V mysql-server-mroonga
```

以上でインストールは終了です。

実際にインストールする際には細部が変わっている可能性もあるため、一度以下の公式ドキュメントを確認しましょう。

<http://mroonga.org/ja/docs/install/debian.html>

Windows、OS X、Ubuntu、CentOS

Windows 用にも公式に Mroonga のインストーラー及びコンパイル済みパッケージの ZIP アーカイブが提供されています。OS X の Homebrew、Ubuntu の APT、CentOS の Yum 用にはリポジトリが提供されています。

いずれも公式サイトにインストール方法がありますので、参照してください。

<http://mroonga.org/ja/docs/install.html>

Docker

Mroonga の Docker イメージも Docker Hub から配布されています。以下のように `docker run` コマンドを実行することで、自動的にイメージの取得からコンテナの起動までが行われます。

1zwMroonga の Docker イメージの取得

```
% docker run --detach --publish=3306:3306 groonga/mroonga
```

これでポート番号 3306 をリスンしながら Mroonga インストール済みの MySQL サーバーが起動します。通常の MySQL クライアントを使って接続が可能です。

1zwMroonga への接続

```
% mysql --host=127.0.0.1 --port=3306 --user=root
```

(Linux 以外は `--host` が違いそう)

但し、Docker コンテナは、削除と同時にデータが失われてしまいます。これは多くの場合望ましい挙動ではないでしょう。次のように `volume` オプションを追加することで、ホスト上のディレクトリをコンテナと共有できます。

1zwvolume オプションの指定

```
% docker run --detach --publish=3306:3306 --volume=/tmp/mroonga:/var/lib/mysql groonga/mroonga
```

これで、ホスト側の `/tmp/mroonga` 以下に MySQL のデータファイルが作成されるようになります。

Mroonga を使ったテーブルの作成については、「A.2 データベースとテーブルの作成」を参照してください。

また、Mroonga の Docker イメージは、MySQL と Mroonga それぞれのバージョンの組み合わせごとに多数のイメージが作られています。入手可能な組み合わせは Docker Hub の Mroonga のページで確認できます。

<https://hub.docker.com/r/groonga/mroonga/>

A.2 データベースとテーブルの作成

データベースの作成には、通常の MySQL と変わることは何也没有什么ありません。MySQL サーバーに接続し、`CREATE DATABASE` 文を実行することでデータベースを作成します。

1zw データベースの作成

```
CREATE DATABASE pdfsearch;
```

必要に応じて権限設定などを行ってください。

テーブルの作成には気を付けることがあります。まず、ストレージエンジンを Mroonga にする必要があります。MySQL で何も指定せずに `CREATE TABLE` を実行した場合、ストレージエンジンはデフォルトの InnoDB になります。これを他のストレージエンジンに変更するには、`ENGINE` オプションを使用します。

1zw テーブルのストレージエンジンに Mroonga を指定

```
CREATE TABLE 'pdfs' (  
  -- カラム定義等  
) ENGINE = Mroonga DEFAULT CHARSET utf8;
```

また、検索対象にしたいカラムに、(通常のインデックスとは異なる) 全文検索用インデックスを作成する必要があります。

1zw 全文検索インデックスの追加

```
ALTER TABLE 'pdfs' ADD FULLTEXT INDEX (title, content);
```

これ自体は通常の MySQL での操作なので、ALTER TABLE のほか、CREATE TABLE でインデックスを作成することもできます。

1zw テーブル作成時の全文検索インデックスの作成

```
CREATE TABLE 'pdfs' (  
  id INT PRIMARY KEY AUTO_INCREMENT,  
  file VARCHAR(255),  
  title VARCHAR(255),  
  content LONGTEXT,  
  FULLTEXT INDEX (title, content)  
) ENGINE = Mroonga DEFAULT CHARSET utf8;
```

これは、本書で使った Docker イメージを作る際に、実際に使った SQL 文です。

また、SQL を使わず [phpMyAdmin](#) の UI でテーブルを作成することもあるかも知れません。Mroonga がインストール済みであれば、phpMyAdmin でもストレージエンジンの選択肢に Mroonga が追加されていることを確認しているので、同じように Mroonga を使い始めることができます。

A.3 Mroonga のコミュニティ、サポート

Mroonga を含む Groonga については、開発者が日本人であることもあり、日本語で気軽に問い合わせられます。公式サイトにはメーリングリスト、チャット、Twitter、Facebook ページについて紹介されているので、質問や要望がある際にはぜひ活用してください。

<http://mroonga.org/ja/docs/community.html>

また、開発や運用などの有償のサポートサービスもあります。業務で使用する場合には検討するといでしょう。

<http://groonga.org/ja/support/>

A.4 Docker コンテナの作り直し

本書のサンプルでは Docker コンテナを使って Apache、PHP、MySQL を動かしています。時には始めからやり直したくなることもあるかも知れません。そうした場合は、以下のように一度コンテナを削除して作り直すことができます。

1zw コンテナの削除

```
% docker rm pdfsearch
```

1zw コンテナの作成

```
% docker run --detach --name=pdfsearch --publish=8080:80 \
  --volume=$PWD:/var/lib/pdfsearch kitaitimakoto/grnbook-mroonga
```

コンテナを削除すると、データベース内のデータは全て失われます。PDF の登録からやり直しとなるので注意しましょう。ホストと共有している PHP ファイルは消えることはありません。

A.5 phpMyAdmin によるデータの確認・操作

本書の Docker イメージには phpMyAdmin もインストールされており、`http://localhost:8080/phpmyadmin` でアクセスできます。名前が `root`、パスワードはなし、というユーザーが登録されており、全権限を所有しています（PHP で使用しているユーザーと同じです）。

これを使って実際に入っているデータを確認・削除したり、コンテナはそのままデータベースやテーブルの再作成を行ったりできます。テーブル作成時はストレージエンジンに `Mroonga` を指定することを忘れないようにしてください。詳細は「A.2 データベースとテーブルの作成」で解説しています。

A.6 PHP のデバッグ

本書サンプルで作成する Docker コンテナの中では Apache の `mod_php` モジュールを使って PHP 5.6 が動作しています。もし、PHP スクリプトがうまく動作しない場合は、Apache のエラーログを確認するといいいでしょう。

まず、`docker exec` コマンドを使ってコンテナの中に入ります（正確にはコンテナの中で `bash` を実行します）。

1zwdocker exec でコンテナの中に入る

```
% docker exec --interactive --tty pdfsearch /bin/bash
```

Apache のエラーログは `/var/log/apache2/error_log` なので、これを `less` や `tail` などのコマンドで見ながら PHP を実行することで、どんなエラーが起きているのかを知ることができます。

1zw ログファイルを監視する

```
# less /var/log/apache2/error_log
# tail -f /var/log/apache2/error_log
```