Groonga ではじめる全文検索

版　　発行

第 1 章

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。

1.1 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対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 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。

1.2 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。

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、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い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、本書で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。

[*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。

Docker のインストール

（後で書く）

Docker コンテナの起動

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。ターミナルで上の `docker run` を実行したディレクトリーに移動し、以下のコマンドを実行してください。

リスト 1.1　Docker イメージの取得とコンテナの起動

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

`docker run` コマンドにより、以下の三つのことが行われます。

1. インターネット経由の Docker イメージの取得（もしシステム上に存在しない場合）
2. 取得した Docker イメージを元にした Docker コンテナの作成
3. 作成した Docker コンテナの起動

Docker イメージの取得とコンテナの作成は一度行えば充分なので、コンテナの二度目以降の起動には別のコマンドを使用します。

1zwDocker コンテナの起動

```
% docker start pdfsearch
```

1zwDocker コンテナの停止

```
% docker stop pdfsearch
```

コンテナの操作に使用している `pdfsearch` という名前は、リスト 2.1 における `name` オプションで指定した物を使用しています。`docker run` 実行時に異なる名前を指定した場合は、そちらを使いましょう。（name をつけ忘れた時は docker ps で探して消すが、説明いる？）

もし、二度目も `docker run` コマンドを使用してしまうと、新たに**別の**コンテナを作成してしまいます。MySQL のデータ等は引き継がれませんし、同じ目的のコンテナが二つ以上あると混乱の元ですので、通常は一つになるようにしておきましょう。

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。このイメージには PHP が含まれているので、`docker run` を実行したディレクトリー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 1.2　info.php

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

ブラウザーで http://localhost:8080/info.php にアクセスしてください。PHP の情報が表示されれば、環境の準備は成功しています。（Docker Toolbox 使っている場合は localhost じゃなさそう）

「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
% docker rm 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 コンテナとホストマシンは、`docker run` コマンドの `volume`（v）オプションで指定したディレクトリーを共有します。上の例では`$PWD:/var/lib/pdfsearch` を指定しています。コロンの左側がホストマシンのディレクトリー、右側がコンテナ内のディレクトリーです。`$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（コンテナ内は`/var/lib/pdfsearch` がドキュメントルートとなるよう設定されており、こちらは常にこの値で大丈夫です）。

1.3　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しょう。以下のようなシステムを作ることにします。

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）

1.4　データベースの作成

まず、PHP を使って、MySQL にデータベースを作成します。作成の手順は通常の MySQL でのデータベース作成と同様です。Docker コンテナ内では既に MySQL が動作しており、`root` ユーザーでパスワード無しでログインできます。

ソースコードは以下のようになります。[*2]

[*2] 限定された状況での一度きりの処理のため、褒められない書き方をしている箇所もあります。実際のアプリケーションで使用する場合には、フレームワークなどのやり方に則った適切な方法でデータベースを操作してください。

リスト 1.3　create-database.php

```
<?php
define('DBNAME', 'pdfsearch');

$dbh = new PDO('mysql:host=localhost;charset=utf8', 'root', '');

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $result = $dbh->exec('CREATE␣DATABASE␣' . DBNAME);

    header("Location:␣{$_SERVER['REQUEST_URI']}");
}

$result = $dbh->query('SHOW␣DATABASES')->fetchAll();
$databases = array_map(function($row) {
    return $row[0];
}, $result);
?>
<!doctype html>
<html>
  <head>
    <title>データベースの作成</title>
  </head>
  <body>
    <h1>データベースの作成</h1>
    <?php if (! (in_array(DBNAME, $databases))): ?>
    <form method="post">
      <input type="submit" value="pdfsearchデータベースの作成">
    </form>
    <?php endif; ?>
    <h2>現在のデータベース一覧</h2>
    <ul>
    <?php foreach ($databases as $database): ?>
      <li><?php echo htmlspecialchars($database, ENT_QUOTES, 'UTF-8'); ?></li>
    <?php endforeach; ?>
    </ul>
  </body>
</html>
```

通常の MySQL に於けるデータベースと全く同じ手順でデータベースを作成していることが分かると思います。Mroonga を使う場合にも、データベースの作成には特別なことはありません。

うまくできたら、http://localhost:8080/create-database.php を表示して、実際に画面上のボタンを押してデータベースを作成してみましょう。「現在のデータベース一覧」に「pdfsearch」というデータベースが加わるはずです。

■コラム: PHP のデバッグ

PHP が期待通りに動作しない場合や画面が真っ白になって何が悪いか分からない場合は、`docker logs` コマンドでログを確認することができます。

リスト 1.4　docker logs で PHP のログを確認

```
% docker log pdfsearch
13.03.2016 09:41:25 INFO  -- [web:php] switch :starting [:unmonitored => :starting] (
reason: monitor by user)
160313 09:41:26 mysqld_safe Can't log to error log and syslog at the same time.  Remove
all --log-error configuration options for --syslog to take effect.
160313 09:41:26 mysqld_safe Logging to '/var/log/mysql/error.log'.
160313 09:41:27 mysqld_safe Starting mysqld daemon with databases from /var/lib/mysql
[Sun Mar 13 09:41:54 2016] 172.17.0.1:58804 [200]: /create-database.php
[Sun Mar 13 09:41:55 2016] 172.17.0.1:58808 [404]: /favicon.ico - No such file or
directory
[Sun Mar 13 09:42:01 2016] PHP Parse error:  syntax error, unexpected '$databases' (
T_VARIABLE) in /var/lib/pdfsearch/create-database.php on line 14
[Sun Mar 13 09:42:01 2016] 172.17.0.1:58812 [500]: /create-database.php - syntax error,
unexpected '$databases' (T_VARIABLE) in /var/lib/pdfsearch/create-database.php on line
14
```

`-f` オプションを付け

1zw

```
% docker logs -f pdfsearch
```

と実行すると、ログが出力する度にそのログが表示されます。Docker コンテナの中では PHP のビルトインサーバーが動作しているため、ログの内容もそれに準じます。

また、ブラウザーでエラー内容を確認したい場合には、PHP ファイルの冒頭で

1zw

```
<?php
ini_set('dispaly_errors', 'stdout');
```

と設定するとよいでしょう。

1.5　テーブルの作成

データベースが出来たので、以下のカラムを持つ pdfs テーブル作りましょう。

表 1.1　作成する pdfs テーブル

カラム	内容	データ型	備考
path	システム内のパス（場所）	VARCHAR(255)	主キー。検索対象ではない
title	PDF 文書のタイトル	VARCHAR(255)	検索対象
content	PDF 内のテキスト	LONGTEXT	検索対象

やり方はデータベースの作成と同様です。

リスト 1.5　create-table.php

```
<?php
define('DBNAME', 'pdfsearch');
define('TABLE_NAME', 'pdfs');

$dsn = 'mysql:host=localhost;dbname=' . DBNAME . ';charset=utf8';
$dbh = new PDO($dsn, 'root', '');
$table = TABLE_NAME;

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $statement = <<<STATEMENT
CREATE TABLE `{$table}` (
    id      INT PRIMARY KEY AUTO_INCREMENT,
    file    VARCHAR(255),
    title   VARCHAR(255),
    content LONGTEXT,
    FULLTEXT INDEX (title),
    FULLTEXT INDEX (content)
) ENGINE = Mroonga DEFAULT CHARSET utf8;
STATEMENT;
    $dbh->exec($statement);

    header("Location: {$_SERVER['REQUEST_URI']}");
}

$result = $dbh->query('SHOW TABLES')->fetchAll();
$tables = array_map(function($row) {
    return $row[0];
}, $result);

?>
<!doctype html>
```

```
<html>
  <head>
    <title>テーブルの作成</title>
  </head>
  <body>
    <h1>テーブルの作成</h1>
    <?php if (! (in_array(TABLE_NAME, $tables))): ?>
    <form method="post">
      <input type="submit" value="pdfsテーブルの作成">
    </form>
    <?php endif; ?>
    <h2>現在のテーブル一覧</h2>
    <ul>
    <?php foreach ($tables as $table): ?>
      <li><?php echo htmlspecialchars($table, ENT_QUOTES, 'UTF-8') ?></li>
    <?php endforeach; ?>
    </ul>
  </body>
</html>
```

http://localhost:8080/create-table.php にアクセスして pdfs テーブルを作成してみましょう。事前に pdfsearch テーブルを作っておかないとエラーになるので注意してください。

データベース作成時と違って、Mroonga を使ったテーブルを作る時にはいくつか注意点があります。

一番大事なのは、CREATE TABLE 文の最後に、ストレージエンジンとして Mroonga を指定することです。

1zw

```
CREATE TABLE (
-- ...
) ENGINE = Mroonga DEFAULT CHARSET utf8;
```

普段は InnoDB や MyISAM を指定しているところを切り替えるだけなので、難しいことはないでしょう。

次に、インデックスの作成方法が InnoDB などと異なります。PDF のタイトルや内容を完全一致で検索することはまずないでしょうから、通常のインデックスは必要ありません。代わりに、全文検索のための特別なインデックスを作る、FULLTEXT INDEX 構文を使用します。これは、MySQL に元々備わっている、全文検索機能のための構文です。Mroonga でも用途は同じなので、全く同じ構文で使用できます。

それ以外は、通常の MySQL 使い方と同じです。主キーにはパスといった意味のある物ではなく、ただレコードを識別するためだけのサロゲートキーを使う人もいることでしょう。それでも構いません。

1.6　データ入力機能の作成

いよいよアプリケーションらしい所を作っていきます。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

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 1.6　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)) {
    $uploads = \PDFSearch\Upload::fromFilesInfo($_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 1.7　upload.php

```
<?php
namespace PDFSearch;

class Upload
{
    protected $name;
    protected $tmpName;

    public static function fromFilesInfo(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[*3]、全てが表示されることも確認してみてください。

図 1.1　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。

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[*4]。

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

リスト 1.8　php-poppler の基本的な使い方

```
<?php

// pdfinfoコマンドのラッパー
```

[*3] 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。

[*4] 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。

```
$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();

$pdf = new \Poppler\Processer\Pdffile($pdfinfo, $pdftotet, $peftohtml);

// 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');
// 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\Processer\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 1.9　upload.php

```
<?php
namespace PDFSearch;

class Upload
{
    protected $name;
    protected $tmpName;

    public static function fromFilesInfo(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;
    }
}
```

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 1.2　PDF 内の情報を取得

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 1.10　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) {
            $tableb->insert($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 1.11　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();
    }
}
```

ウェブブラウザーでアクセスして、データベースの中身を表示してみましょう。

図 1.3　アップロードした PDF の情報を表示

1.7　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、通常の MySQL の全文検索機能、即ち `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に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よ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。

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 1.12　search.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 = "SELECT␣*␣FROM␣`pdfs`␣WHERE␣MATCH(content)␣AGAINST(:query␣IN␣BOOLEAN␣MODE
);";

    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();
    }

    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 1.13　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
$searchQuery = null;
if ($_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) {
            $tableb->insert($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>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; ?>

<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>
```

SQL 一文で済むので簡単ですね。では、検索してみましょう。

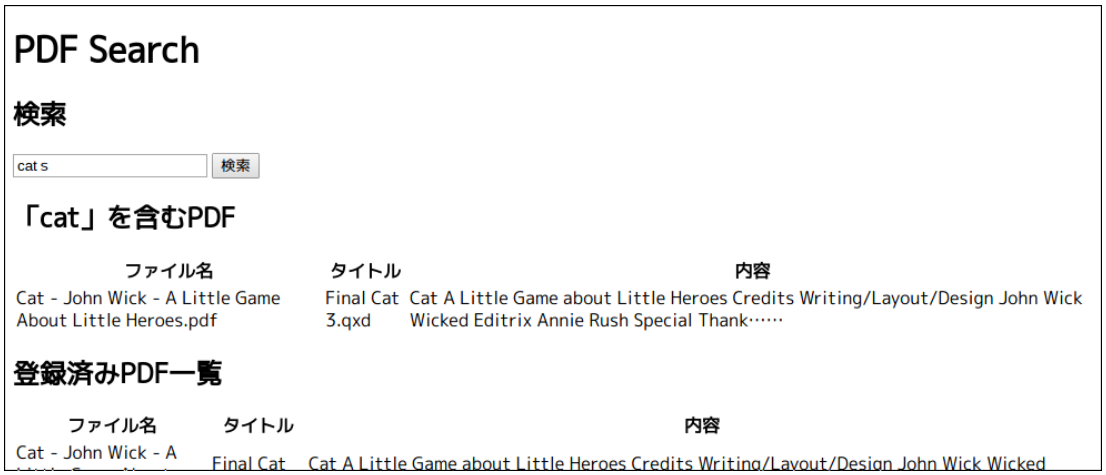

図 1.4　全文検索結果の表示

以上で、基本的な機能は完成です。以降では、Mroonga に特徴的な幾つかの機能を紹介し、実装していきます。