PostgreSQLとPGroongaで作る PHPマニュアル 高速全文検索システム

須藤功平 クリアコード

PHPカンファレンス2017
2017-10-08

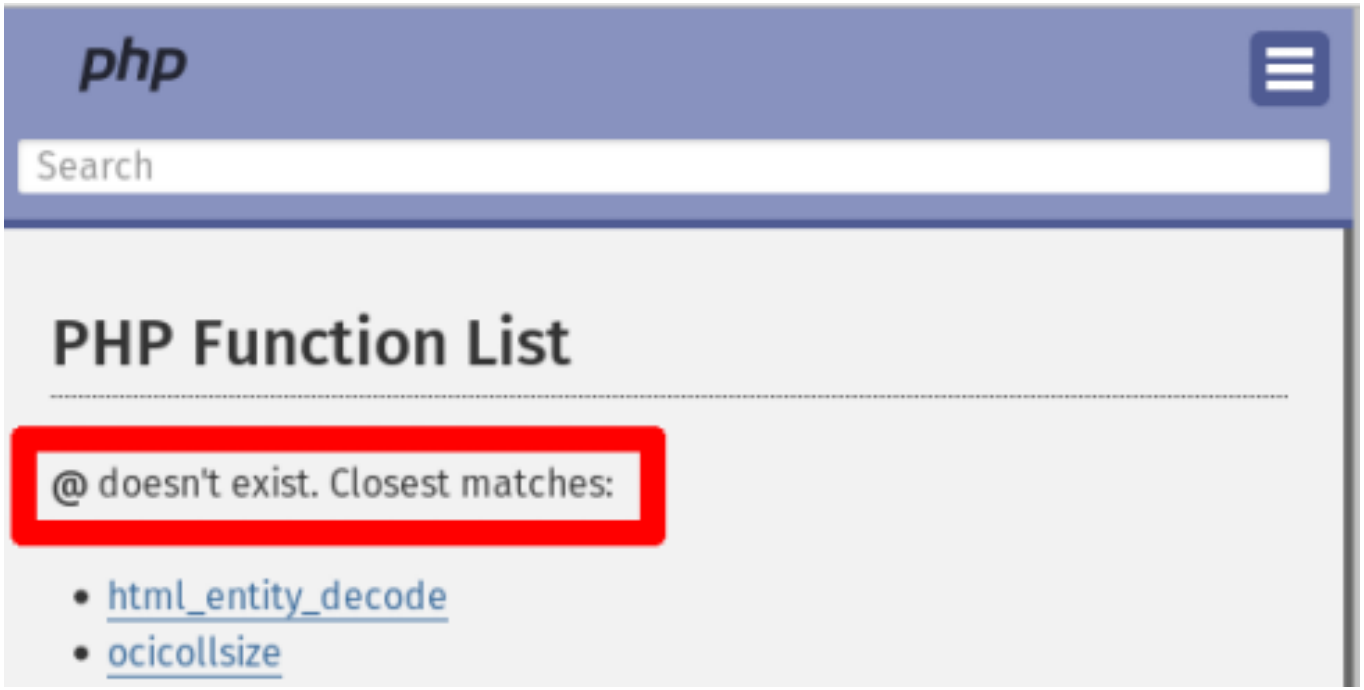
PHPマニュアル

http://php.net/manual/ja/

@で検索

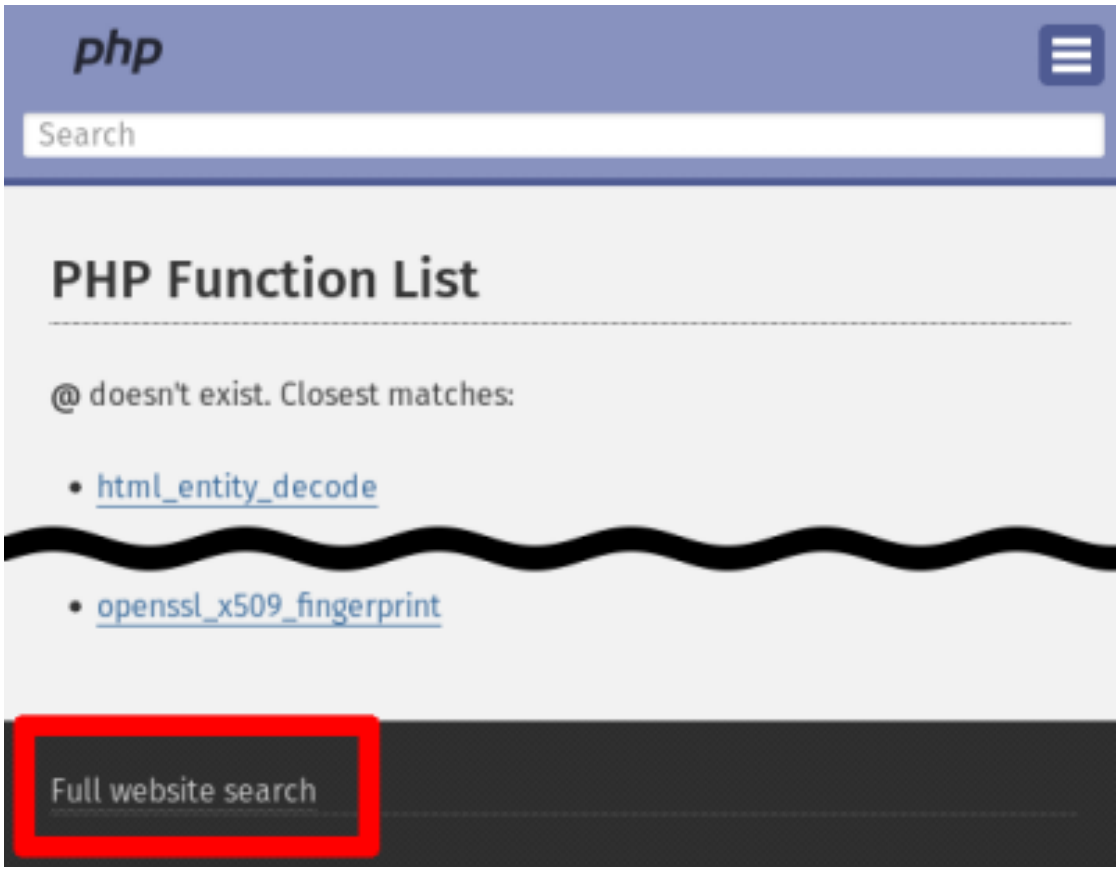
@：エラー制御演算子

@でnot found



関数・クラス・例外のみ検索対象

全文検索あり！



Googleカスタム検索

@でnot found

Googleは自然言語向けだから

マニュアル検索

- 自然言語向けと傾向が違う
  - @：自然言語ではノイズ
    特に日本語ではノイズ
  - @：マニュアルでは重要語
- プログラミング言語用のチューニングが必要
  - 🔍欲しい情報が見つかる！

マニュアル検索システムの作り方

PHP
+
PostgreSQL

PostgreSQLと全文検索

- LIKE：組込機能
- textsearch：組込機能
- pg_trgm：標準添付
  - アーカイブには含まれている
  - 別途インストールすれば使える

LIKE

- 少ないデータ
  - 十分実用的
  - 400文字×20万件くらいなら1秒とか
- 少なくないデータ
  - 性能問題アリ

PHPマニュアルのデータ

件数	13095
平均文字数	871文字

PHPマニュアルでLIKE

- 👍速度は十分実用的
  - LIKE '%@%'で約100ms
- 👎それっぽい順のソート不可
  - 全文検索ではソート順が重要
  - ユーザーは先頭n件しか見ない

textsearch

- インデックスを作るので速い
- 言語毎にモジュールが必要
  - 英語やフランス語などは組込
  - 日本語は別途必要
- 日本語用モジュール
  - 公式にはメンテナンスされていない
    forkして動くようにしている人はいる

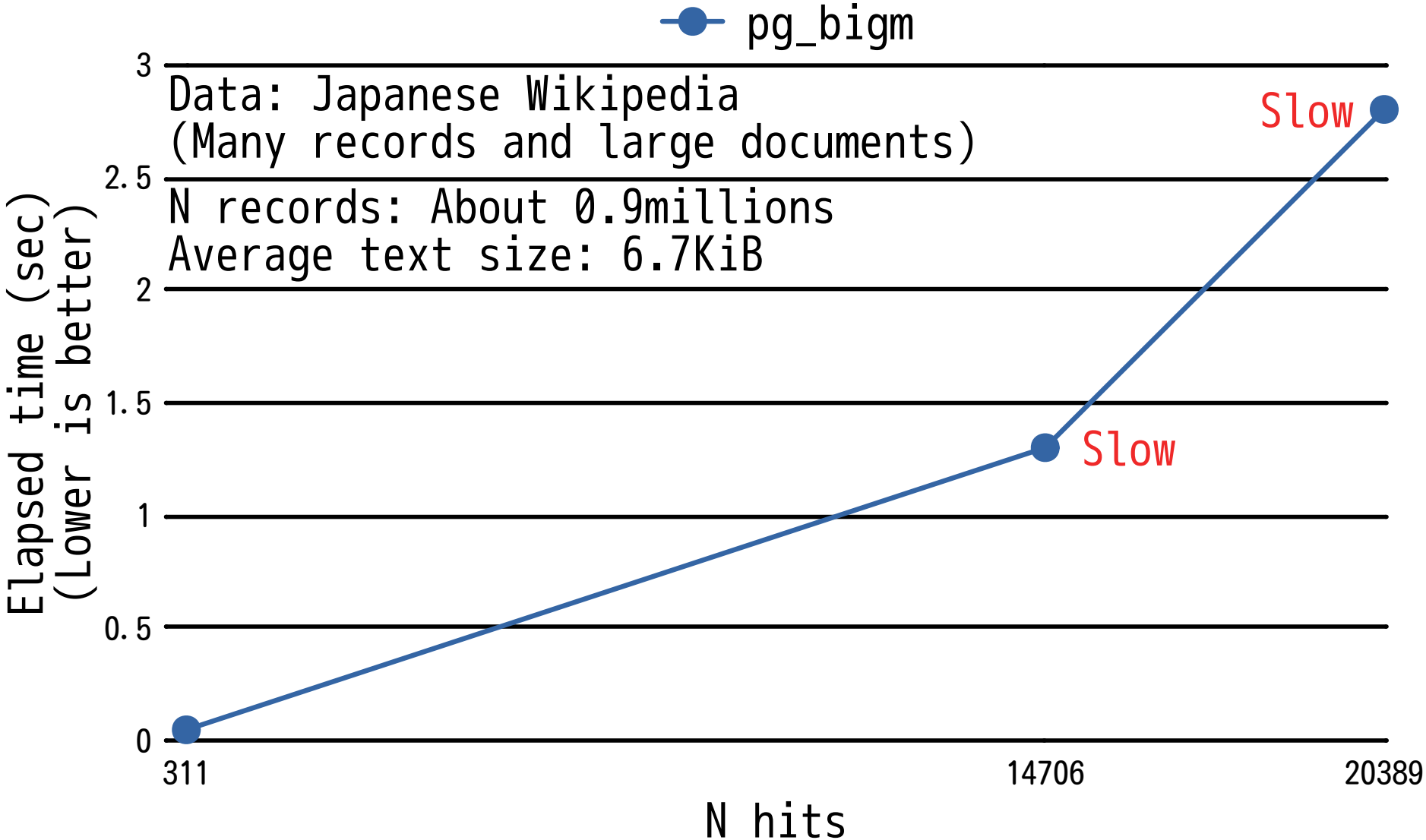
pg_trgm

- インデックスを作るので速い
  - 注：ヒット件数が増えると遅い
  - 注：テキスト量が多いと遅い
  - 注：1,2文字の検索は遅い（米・日本）
- 日本語を使うにはひと工夫必要
  - C.UTF-8を使う
  - ソースを変更してビルド

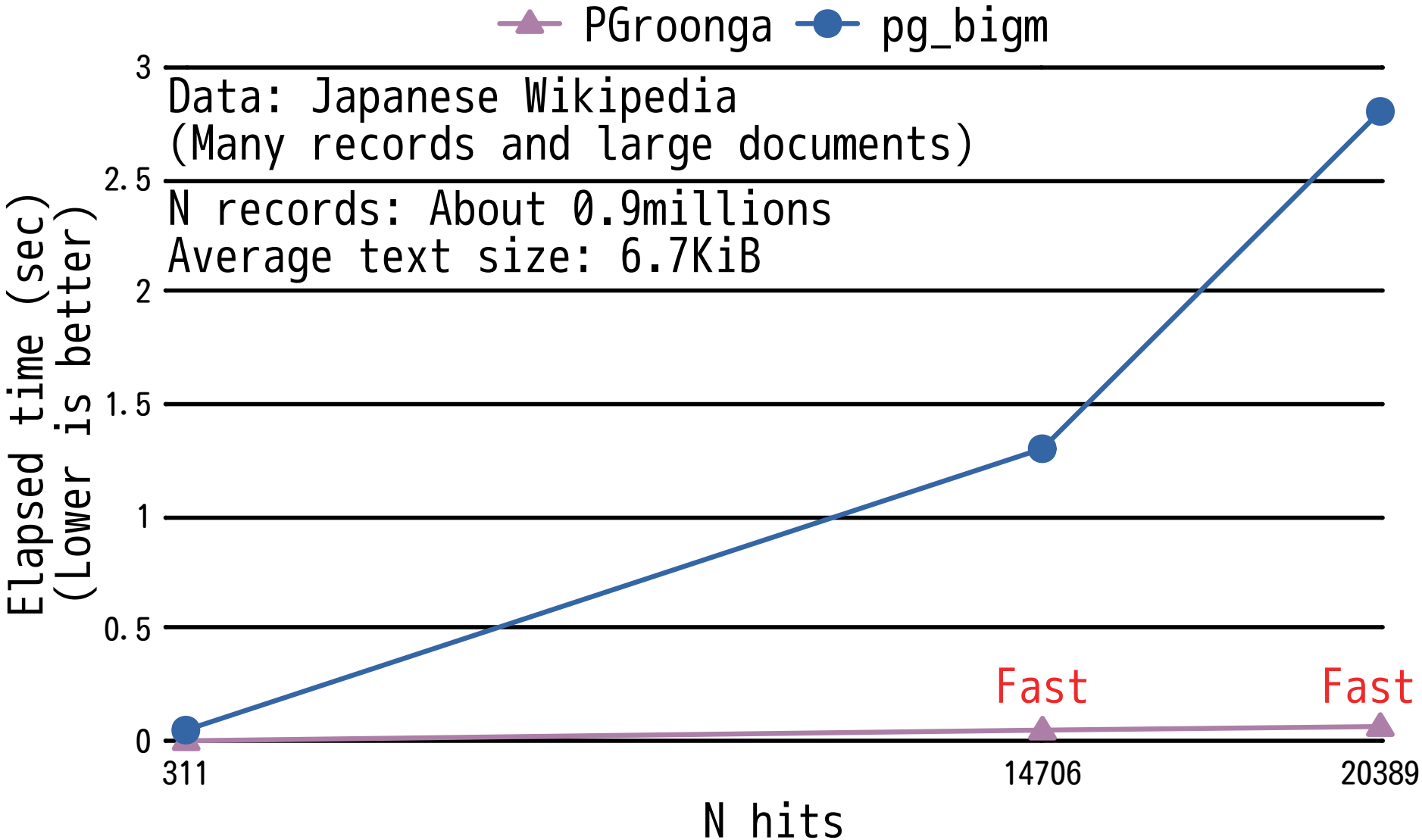
プラグイン

- pg_bigm
  - pg_trgmの日本語対応強化版
  - それっぽい順のソート不可
- PGroonga
  - 本気の全文検索エンジンを利用
  - 速いし日本語もバッチリ！
  - それっぽい順のソート可

ベンチマーク：pg_bigm



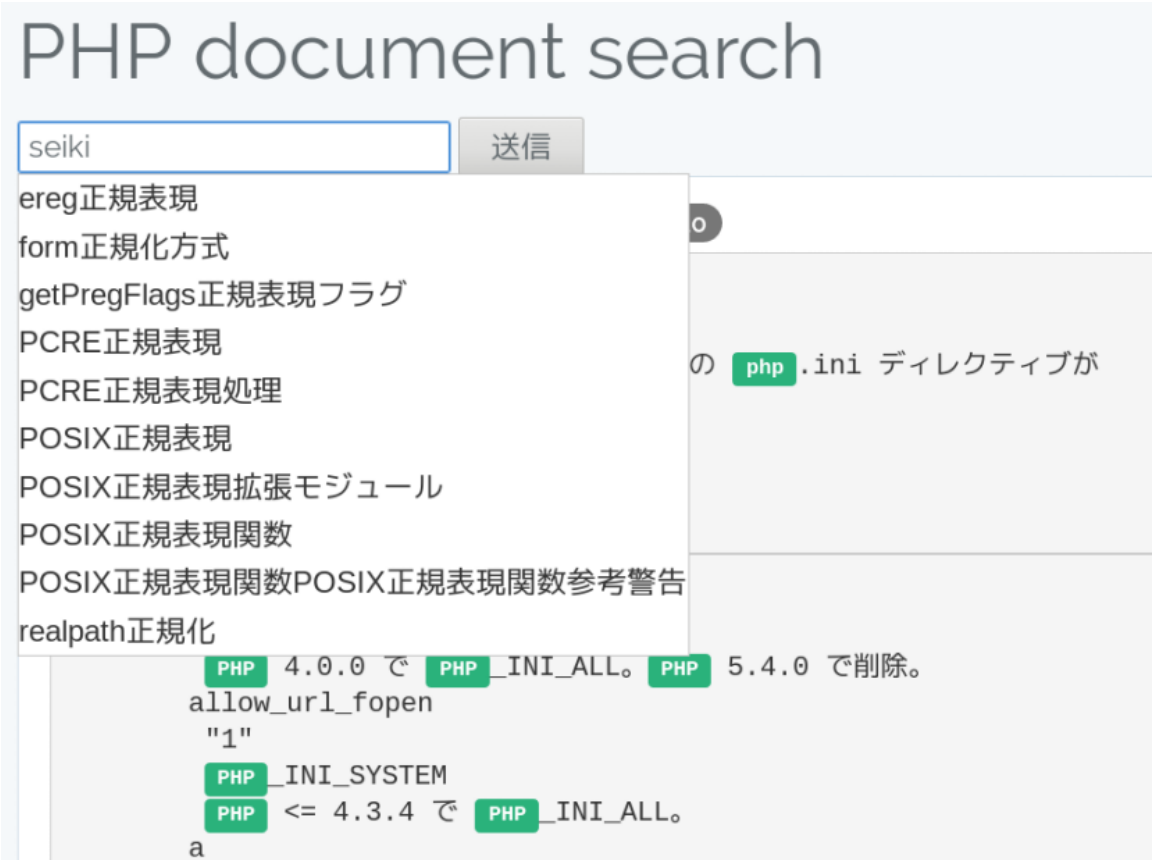
ベンチマーク：PGroonga



PostgreSQLで全文検索システム

- PostgreSQLで全文検索
  - PGroongaがベスト！💯
- PGroonga
  - 高速
  - 日本語対応
  - それっぽい順でソート可

PHP document search



PHP + PostgreSQL + PGroonga

基本機能

- 高速全文検索＋ソート
- 検索キーワードハイライト
- キーワード周辺テキスト表示

高度な機能

- オートコンプリート
  - ローマ字対応（seiki→正規表現）
- 類似マニュアル検索
- 同義語展開
  - 「@」→「@ OR エラー制御演算子」

作り方：ツール

- フレームワーク
  - Laravel
- RDBMS
  - PostgreSQL
- 高速日本語全文検索機能
  - PGroonga

作り方：インストール

- Laravel
  - 省略
- PostgreSQL
  - パッケージで
- PGroonga
  - パッケージで
    https://pgroonga.github.io/ja/install/

初期化：Laravel

```
% laravel new php-document-search
% cd php-document-search
% editor .env
```

初期化：データベース

```
% sudo -u postgres -H \
    createdb php_document_search
```

初期化：PGroonga

```
-- ↓を実行する必要がある
CREATE EXTENSION pgroonga;
```

初期化：PGroonga

マイグレーションファイル作成

```
% php artisan \
    make:migration enable_pgroonga
```

マイグレーション

```php
public function up()
{
  DB::statement("CREATE EXTENSION pgroonga;");
  // CREATE EXTENSION IF NOT EXISTS ...の方がよい
}

public function down()
{
  DB::statement("DROP EXTENSION pgroonga;");
}
```

モデル作成

- マニュアルをモデルにする
  - 名前：Entry
  - 1ページ1インスタンス

モデル作成

```
% php artisan \
    make:model \
        --migration \
        --controller \
        --resource \
        Entry
```

マイグレーション

```
Schema::create("entries", function ($table) {
  $table->increments("id");
  $table->text("url");
  $table->text("title");
  $table->text("content");
  // PGroongaインデックス（デフォルト：全文検索用）
  // 主キー（id）も入れるのが大事！
  // それっぽい順のソートで必要
  $table->index(
     ["id", "title", "content"], null, "pgroonga");
});
```

データ登録

1. PHPのマニュアルを
   ローカルで生成
   - PHPのマニュアルの作り方
     http://doc.php.net/tutorial/
   - フィードバックチャンスが
     いろいろあったよ！
2. ページ毎にPostgreSQLに挿入

コマンド作成

```
% php artisan \
    make:command \
      --command=doc:register \
      RegisterDocuments
```

登録コマンド実装（一部）

```php
public function handle()
{
  foreach (glob("public/doc/*.html") as $html_path) {
    $document = new \DOMDocument();
    @$document->loadHTMLFile($html_path);
    $xpath = new \DOMXPath($document);
    $entry = new Entry();
    $entry->url = "/doc/" . basename($html_path);
    // XPathでテキスト抽出（詳細はソースを参照）
    $this->extract_title($entry, $xpath);
    $this->extract_content($entry, $xpath);
    $entry->save();
  }
}
```

登録

```
% php artisan doc:register
```

検索用コントローラー

```
public function index(Request $request)
{
  $query = $request["query"];
  $entries = Entry::query()
    // ↓はモデルに作る（後述）
    ->fullTextSearch($query)
    ->limit(10)
    ->get();
  return view("entry.search.index",
              ["entries" => $entries,
               "query"   => $query]);
}
```

検索対象モデル

```
public function
  scopeFullTextSearch($query, $search_query)
{
  if ($search_query) {
    return ...; // クエリーがあったら検索
  } else {
    return ...; // なかったら適当に返す（省略）
  }
}
```

検索対象モデル：検索

```
return $query
  ->select("id", "url")
  // それっぽさの度合い
  ->selectRaw("pgroonga_score(entries) AS score")
  // キーワードハイライト
  ->highlightHTML("title", $search_query)
  // キーワード周辺のテキスト（キーワードハイライト付き）
  ->snippetHTML("content", $search_query)
  // タイトルと本文を全文検索（タイトルの方が重要）
  ->whereRaw("title &@~ ? OR content &@~ ?",
             [">($search_query)", $search_query])
  // それっぽい順に返す
  ->orderBy("score", "DESC");
```

キーワードハイライト

```
public function scopeHighlightHTML($query,
                                   $column,
                                   $search_query)
{
  return $query
    // PGroonga提供ハイライト関数
    ->selectRaw("pgroonga_highlight_html($column, " .
    // PGroonga提供クエリーからキーワードを抽出する関数
                "pgroonga_query_extract_keywords(?)) " .
                "AS highlighted_$column",
                [$search_query]);
}
```

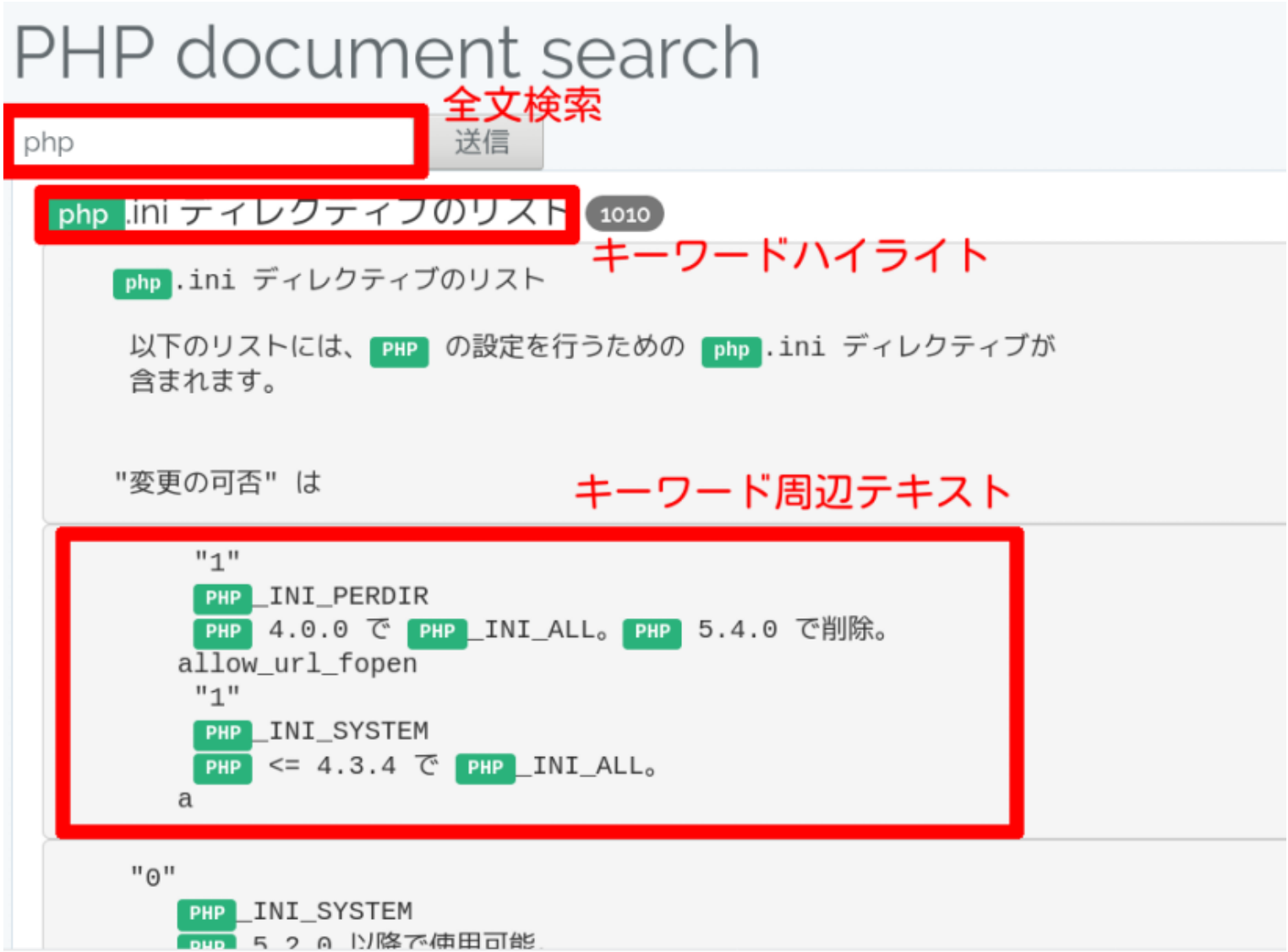
検索結果

```
<div class="entries">
  @foreach ($entries as $entry)
    <a href="{{ $entry->url }}">
      <h4>
        {{-- マークアップ済み！ --}}
        {!! $entry->highlighted_title !!}
        <span class="score">{{ $entry->score }}</span>
      </h4>
      {{-- 周辺テキストはtext[]（後で補足） --}}
      @foreach ($entry->content_snippets as $snippet)
        <pre class="snippet">{!! $snippet !!}</pre>
      @endforeach
    </a>
  @endforeach
</div>
```

検索対象モデル：配列

```
public function getContentSnippetsAttribute($value)
{ // PostgreSQLは配列をサポートしているがPDOは未サポート
  // '["...","..."]'という文字列になるのでそれを配列に変換
  // ※これは回避策なのでPDOに配列サポートを入れたい！
  return array_map(
    function ($e) {
      // 「"」が「\"」になっているので戻す
      return preg_replace('/\\\\(.)/', '$1', $e);
    },
    explode('","', substr($value, 2, -2)));
}
```

高速日本語全文検索！


PHP document search
全文検索
php
送信
php.ini ディレクティブのリスト
1010
キーワードハイライト
php.ini ディレクティブのリスト
以下のリストには、PHP の設定を行うための php.ini ディレクティブが含まれます。
"変更の可否" は
キーワード周辺テキスト
"1"
PHP_INI_PERDIR
PHP 4.0.0 で PHP_INI_ALL。PHP 5.4.0 で削除。
allow_url_fopen
"1"
PHP_INI_SYSTEM
PHP <= 4.3.4 で PHP_INI_ALL。
a
"0"
PHP_INI_SYSTEM

オートコンプリート

- 必要なもの
  - 候補用テーブル
  - 候補のヨミガナ（カタカナ）
  - PGroonga!!!

モデル作成

```
% php artisan \
    make:model \
        --migration \
        --controller \
        --resource \
        Term
```

マイグレーション：カラム

```
Schema::create("terms", function ($table) {
  $table->increments("id");
  $table->text("term");
  $table->text("label");
  $table->text("reading"); // 本当は配列にしたい
  $table->timestamps();
  // インデックス定義（後述）
});
```

マイグレーション インデックス1

```
$table->index([
  // ヨミガナに対する前方一致RK検索用
  // RK：ローマ字・カナ（後述）
  DB::raw("reading " .
    "pgroonga_text_term_search_ops_v2"),
], null, "pgroonga");
```

マイグレーション インデックス2

```
// 候補に対するゆるい全文検索用（中間一致用）
DB::statement(
  "CREATE INDEX terms_term_index " .
  "ON terms " .
  "USING pgroonga (term) " .
  // ↓がポイント
  // ※LaravelがWITHを未サポートなのでSQLで書いている
  // ※回避策なのでLaravelにWITHサポートを入れたい！
  "WITH (tokenizer='TokenBigramSplitSymbolAlphaDigit')");
```

データ登録

1. マニュアルから候補を抽出
   （手動作成はコスト高すぎ）
   - 例：名詞は候補
   - 例：連続した名詞→1つの候補に
2. ヨミガナも自動で推測
   （MeCabを利用）
3. 候補をPostgreSQLに挿入

コマンド作成

```
% php artisan \
    make:command \
      --command=terms:register \
      RegisterTerms
```

登録コマンド実装（一部）

```
public function handle()
{
  foreach (Entry::query()->cursor() as $entry) {
    // processTextの実装はソースを参照
    // タイトルから抽出
    $this->processText($entry->title);
    // 本文から抽出
    $this->processText($entry->content);
  }
}
```

登録

```
% php artisan terms:register
```

前方一致RK検索

- 日本語特化の前方一致検索
  - ローマ字・ひらがな・カタカナで
    カタカナを前方一致検索できる
  - gy→ギュウニュウ
  - ぎ→ギュウニュウ
  - ギ→ギュウニュウ

候補モデル：検索

```
public function scopeComplete($query, $search_query)
{
  return $query
    ->select("label")
    ->highlightHTML("label", $search_query)
                // 前方一致RK検索
    ->whereRaw("reading &^~ :query OR " .
                // ゆるい全文検索
                "term &@~ :query",
                ["query" => $search_query])
    ->orderBy("label")
    ->limit(10);
}
```

コントローラー

```php
public function index(Request $request)
{
  $query = $request["query"];
  // モデルに実装した検索処理を呼び出し
  $terms = Term::query()->complete($query);
  $data = [];
  foreach ($terms->get() as $term) {
    $data[] = [
      "value" => $term->label,
      "label" => $term->highlighted_label,
    ];
  }
  // JSONで候補を返す
  return response()->json($data);
}
```

UI

```
$("#query").autocomplete({
  source: function(request, response) {
    $.ajax({
      url: "/terms/", // コントローラー呼び出し
      dataType: "json",
      data: {query: this.term},
      success: response
    });
  }
}).autocomplete("instance")._renderItem = function(ul, item) {
  return $("<li>")
    .attr("data-value", item.value) // 候補には生データを使う
    .append(item.label) // ハイライトしたデータを表示
    .appendTo(ul);
};
```

オートコンプリート！

PHP document search

seiki

送信

- ereg正規表現
- form正規化方式
- getPregFlags正規表現フラグ
- PCRE正規表現
- PCRE正規表現処理
- POSIX正規表現
- POSIX正規表現拡張モジュール
- POSIX正規表現関数
- POSIX正規表現関数POSIX正規表現関数参考警告
- realpath正規化

の php.ini ディレクティブが

PHP 4.0.0 で PHP_INI_ALL。PHP 5.4.0 で削除。

allow_url_fopen

"1"

PHP_INI_SYSTEM

PHP <= 4.3.4 で PHP_INI_ALL。

a

類似マニュアル検索


PHP document search
エラー制御演算子
送信
エラー制御演算子
11
エラー関連のマニュアルを提示
Similar entries
1. BEGIN_SILENCE
381303
2. END_SILENCE
381303
3. 出力する PHP エラーの種類を設定する
327711

実現方法

- 類似検索用インデックスが必要
  - 自然言語に合わせた処理で精度向上
  - 日本語ならMeCabを活用
- 類似検索用の演算子を使う
  - 類似検索クエリー
    →対象マニュアルのテキスト全体
  - 参考：全文検索クエリー
    →キーワード

インデックス作成

マイグレーションファイル作成

```
% php artisan \
    make:migration \
      add_similar_search_index
```

マイグレーション

```
public function up()
{ // WITHを使っているのでDB::statementを使用
  DB::statement(
    "CREATE INDEX similar_search_index " .
    "ON entries " .
    // タイトルと内容を合わせたテキストをインデックス
    // 理由1：タイトルも重要→対象に加えて精度向上
    // 理由2：PostgreSQLが全文検索インデックスと
    //        区別できるように
    "USING pgroonga (id, (title || ' ' || content)) " .
    // ポイント：MeCabを使う
    "WITH (tokenizer='TokenMecab')");
}
```

類似検索：スコープ

```
public function scopeSimilarSearch($query, $text)
{
  return $query
    ->select("id", "url", "title")
    ->selectRaw("pgroonga_score(entries) AS score")
    // インデックス定義と同じ式↓を指定すること！
    //   title || ' ' || content
    // &@*が類似検索の演算子
    ->whereRaw("(title || ' ' || content) &@* ?",
               [$text])
    ->orderBy("score", "DESC");
}
```

類似検索：インスタンスメソッド

```
public function similarEntries()
{
  return Entries::query()
    // タイトルと内容がクエリー
    ->similarSearch("{$this->title} {$this->content}")
    // 自分自身を除くこと！
    ->where("id", "<>", $this->id)
    // 最も類似している3件のみ取得
    ->limit(3)
    ->get();
}
```

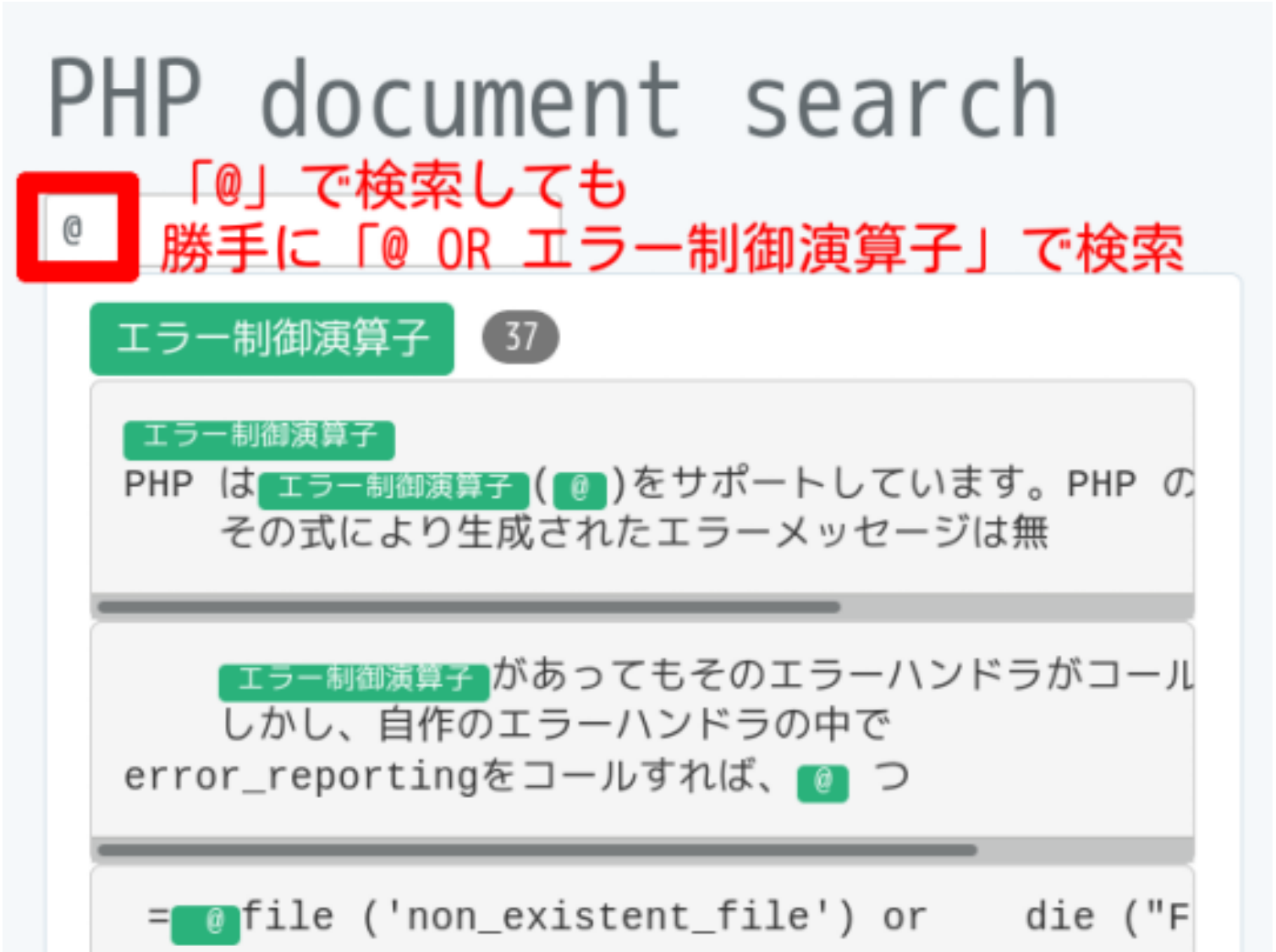
類似検索：使い方

```
@foreach ($entries as $entry)
  <ol> {{-- ↓マニュアル毎に類似文書検索 --}}
  @foreach ($entry->similarEntries() as $similarEntry)
    <li>
      <a href="{{ $similarEntry->url }}">
        {{ $similarEntry->title }}
        <span class="score">{{ $similarEntry->score }}</span>
      </a>
    </li>
  @endforeach
  </ol>
@endforeach
```

類似マニュアル検索


PHP document search
エラー制御演算子
送信
エラー制御演算子
11
エラー関連のマニュアルを提示
Similar entries
1. BEGIN_SILENCE
381303
2. END_SILENCE
381303
3. 出力する PHP エラーの種類を設定する
327711

同義語展開


PHP document search
@
「@」で検索しても
勝手に「@ OR エラー制御演算子」で検索
エラー制御演算子 37
エラー制御演算子
PHP は エラー制御演算子 (@)をサポートしています。PHP の
その式により生成されたエラーメッセージは無
エラー制御演算子があってもそのエラーハンドラがコール
しかし、自作のエラーハンドラの中で
error_reportingをコールすれば、 @ つ
= @file ('non_existent_file') or die ("F

実現方法

- 同義語管理テーブルを作成
- 同義語を登録
- 同義語を展開して検索

モデル作成

```
% php artisan \
    make:model \
        --migration \
        --controller \
        --resource \
        Synonym
```

マイグレーション：カラム

```
public function up() {
  Schema::create('synonyms', function ($table) {
    $table->increments('id');
    $table->text('term');      // 展開対象の語
    $table->text('synonym');   // 展開後の語
    // 例：term: @, synonym: @
    // 例：term: @, synonym: エラー制御演算子
    //       「@」→「@ OR エラー制御演算子」
    $table->timestamps();
    // インデックス定義（後述）
  });
}
```

マイグレーション インデックス

```
$table->index(
  // termで完全一致できるようにする設定
  [DB::raw(
    "term pgroonga_text_term_search_ops_v2")],
  null,
  "pgroonga");
```

同義語登録

シーダー作成

```
% php artisan \
    make:seeder \
      SynonymsTableSeeder
```

シーダー

```php
public function run()
{
  $synonyms = [
// @そのもので検索させないならこれはいらない
    ["term" => "@", "synonym" => "@"],
    ["term" => "@",
     // synonymでは演算子を使える
     // ">": 重要度を上げる演算子
     "synonym" => ">エラー制御演算子"],
  ];
  DB::table("synonyms")->insert($synonyms);
}
```

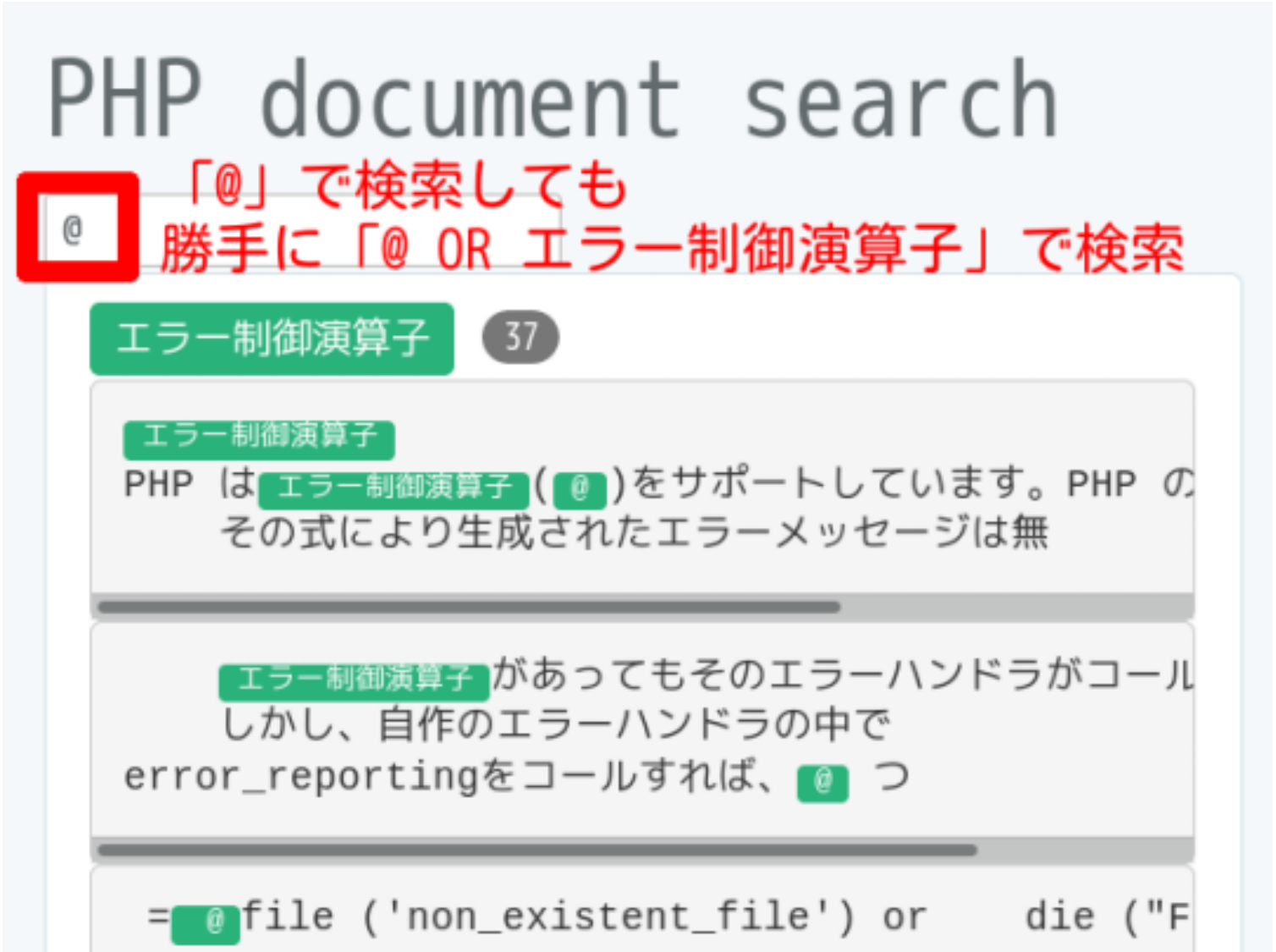
動作確認

```
SELECT pgroonga_query_expand(
  'synonyms', -- テーブル名
  'term', -- 展開対象語のカラム名
  'synonym', -- 展開後の語のカラム名
  '@'); -- 展開対象のクエリー
--      pgroonga_query_expand
-- ------------------------------
--  ((@) OR (>エラー制御演算子))
-- (1 row)
```

検索

```
whereRaw("title &@~ ? OR content &@~ ?",
         [">({$search_query})", $search_query]);
// ↓クエリーをpgroonga_query_expand()で展開して利用
whereRaw(
  "title &@~ pgroonga_query_expand(?, ?, ?, ?) OR " .
  "content &@~ pgroonga_query_expand(?, ?, ?, ?)",
  ["synonyms", "term", "synonym", ">({$search_query})",
   "synonyms", "term", "synonym", $search_query]);
```

同義語展開


PHP document search
@
「@」で検索しても
勝手に「@ OR エラー制御演算子」で検索
エラー制御演算子 37
エラー制御演算子
PHP は エラー制御演算子 (@)をサポートしています。PHP の
その式により生成されたエラーメッセージは無
エラー制御演算子があってもそのエラーハンドラがコール
しかし、自作のエラーハンドラの中で
error_reportingをコールすれば、 @ つ
= @file ('non_existent_file') or die ("F

おさらい：基本機能

- 高速全文検索＋ソート
- 検索キーワードハイライト
- キーワード周辺テキスト表示

おさらい：高度な機能

- オートコンプリート
  - ローマ字対応（seiki→正規表現）
- 類似マニュアル検索
- 同義語展開
  - 「@」→「@ OR エラー制御演算子」

開発者募集！

- 公式検索システム置き換え！？
  - 必要そう：複数バージョン対応
  - 必要そう：複数言語対応
- マニュアルをさらによく！
  - 検索を便利に！→ユーザー増加！
  - →フィードバックする人も増加！

使いたい！

- WEICさんが運用予定！
  - 2017年10月中にリリース予定
  - URL: http://phpdocs.weic.co.jp/
- 宣伝（運用スポンサーの宣伝枠）
  - PHPエンジニア大募集！
  - 業務時間内にこれの開発もできる!?

PHP document search情報

- ソース
  - https://github.com/kou/php-document-search
- OSS
  - MITライセンス

まとめ

- PostgreSQL + PGroonga
  - 高速日本語全文検索サービスを PHPで簡単に作れる！
- 開発者募集！
- サービス提供予定 by WEIC！
  - URL: http://phpdocs.weic.co.jp/

MySQLでもできる？

- MySQL・PostgreSQLだけで作る
  高速でリッチな
  全文検索システム
  - db tech showcase Tokyo 2017の資料
  - PGroongaの代わりにMroongaを使う
  - SQLでの書き方だけでPHPの話はない

https://slide.rabbit-shocker.org/authors/kou/db-tech-showcase-tokyo-2017/

PHP+MySQL+Mroonga入門

- Groongaではじめる全文検索
  - https://grnbook-ja.tumblr.com/
  - 著者：北市真
  - PHP+Mroonga入門の電子書籍
  - 今はまだ無料！

PHPの開発へ参加！

- PHPの開発に参加しませんか？
  - 例：PDO/LaravelのPostgreSQL関連
  - 例：マニュアル生成まわり
  - 例：PHP document search関連
- やりたいけど自分はムリそう…
  - そんなことはないんですよ！

OSS Gate

- OSS Gate
  - OSS開発者を増やす取り組み
- OSS Gateワークショップ
  - OSS開発未経験者を経験者にするワークショップ
  - PHPもOSS！

ワークショップ

- このカンファレンス内で開催！
  - 午後にこの部屋で開催（2時間45分）
  - 参加希望者は私に声をかけて！
- PHP関連のOSSの開発に参加する人を増やそう！
  - 今回だけで終わりにしないで**今回を始まりにしたい！**