Mroonga
と
PGroonga
導入方法例

須藤功平

クリアコード

MySQLとPostgreSQLと日本語全文検索
2016-09-29

Mroonga・PGroonga

- Mroonga（むるんが）
  - MySQLに
    高速日本語全文検索機能を追加する
    プロダクト
- PGroonga（ぴーじーるんが）
  - PostgreSQLに
    高速日本語全文検索機能を追加する
    プロダクト

高速？

速さ：検索1

キーワード：テレビアニメ
（ヒット数：約2万3千件）

InnoDB ngram	3m2s
InnoDB MeCab	6m20s
Mroonga:1	0.11s
pg_bigm	4s
PGroonga:2	0.29s

詳細は第1回の資料を参照
http://slide.rabbit-shocker.org/authors/kou/mysql-and-postgresql-and-japanese-full-text-search/

導入方法例

既存システムへの導入方法を紹介

- Redmine
  - チケット管理システム
  - Ruby on Railsを使用
- Zulip
  - チャットツール
  - Djangoを使用

Redmine


ホーム プロジェクト ヘルプ
ログイン 登録する
Redmine
検索:
概要 Download 活動 ロードマップ チケット ニュース Wiki フォーラム リポジトリ
チケット
ここから全文検索できる
フィルタ
ステータス
未完了
フィルタ追加
オプション
適用 クリア
すべてのチケットを見る
サマリー
カスタムクエリ
Documentation issues
Open defects
Open features
Patch queue
Plugin issues
Translation patches
トラッカー ステータス 題名 更新日 カテゴリ
23951 Feature New Social login api 2016-09-28 15:03
23950 Feature New Feed Integration 2016-09-28 15:02
23947 Patch New Single issue edit from context menu fix 2016-09-28 08:18 Issues
23946 Patch New News appearance 2016-09-28 08:51 UI
23940 Defect Needs feedback Private issues notifications 2016-09-27 12:45 Email notifications
23939 Defect Needs feedback Watchers not recieving e-mail notifications 2016-09-27 03:57 Email receiving
23932 Feature New Update TinyTds to recent version. 2016-09-26 13:10 Gems support
23922 Defect New Time Entries context menu shows activities not available for the time entry's project 2016-09-25 09:48

全文検索プラグイン

GitHub: okkez/redmine_full_text_search

- MySQL・PostgreSQL両方対応
  - MySQLのときはMroongaを利用
  - PostgreSQLのときはPGroongaを利用

速さ

MySQL + Mroongaのケース

プラグイン	チケット数	時間
なし	約3000件	467ms
あり	約3000件	93ms
あり	約200万件	380ms

速さ：コメント

Kuniharu AKAHANE
@akahane92

フォローする

200万チケット@MySQLでやってみたよ。検索時間は約380ms。 #Redmine の未来が広がって嬉しいな。ありがたいな。／Redmineで高速に全文検索する方法 - ククログ(2016-04-11)
clear-code.com/blog/2016/4/11…
@_clear_code

7 リツイート　7 いいね

10:31 - 2016年5月21日

https://twitter.com/akahane92/status/733832496945594368

使いどころ

- Mroonga
  - 速さが欲しい
  - トランザクションはいらない
- PGroonga
  - 機能が欲しい
  - トランザクションも欲しい

Redmine

- トランザクション必須
  - Mroongaを使うときは一工夫必要
  - PGroongaはそのままで大丈夫

Redmine+Mroonga：方針

- チケットテーブルは変えない
- 全文検索用テーブルを別途作成
- 全文検索用テーブルから
  チケットテーブルを参照

マイグレーション

```
def up
  create_table(:fts_issues, # 全文検索用テーブル作成
               id: false, # idは有効・無効どっちでも可
               options: "ENGINE=Mroonga") do |t|
    t.belongs_to :issue, index: true, null: false
    t.string :subject, default: "", null: false
    t.text :description, limit: 65535, null: false
  end
  execute("INSERT INTO " + # データをコピー
            "fts_issues(issue_id, subject, description) " +
            "SELECT id, subject, description FROM issues;")
  add_index(:fts_issues, [:subject, :description],
            type: "fulltext") # 静的インデックス構築（速い）
end
```

モデル

```ruby
class FtsIssue < ActiveRecord::Base
  # 実際はissue_idカラムは主キーではない。
  # 主キーなしのテーブルなので
  # Active Recordをごまかしているだけ。
  self.primary_key = :issue_id
  belongs_to :issue
end
```

保存

```
class Issue
  # この後にロールバックされることがあるのでカンペキではない
  # 再度同じチケットを更新するかデータを入れ直せば直る
  after_save do |record|
    fts_record =
      FtsIssue.find_or_initialize_by(issue_id: record.id)
    fts_record.subject     = record.subject
    fts_record.description = record.description
    fts_record.save!
  end
end
```

全文検索

```
issue.
  joins(:fts_issue).
  where(["MATCH(fts_issues.subject, " +
                 "fts_issues.description) " +
           "AGAINST (? IN BOOLEAN MODE)",
          # ↓デフォルトANDで全文検索
          "*D+ #{keywords.join(', ')}"])
```

Redmine+Mroonga：まとめ

- トランザクション必須
  - 元テーブルを置き換えない
  - 全文検索用テーブルを作成
- データ
  - アプリが複数テーブルに保存
- 全文検索
  - JOINしてMATCH AGAINST

Redmine+PGroonga：方針

- 全文検索用インデックス作成
- インデックスに主キーを含める
  - 検索スコアーを取得するため

マイグレーション

```
def up
  enable_extension("pgroonga")
  add_index(:issues,
            [:id, :subject, :description],
            using: "pgroonga")
end
```

モデル

追加・変更なし

保存

追加・変更なし

全文検索

```
issue.
  # 検索対象のカラムごとに
  # クエリーを指定
  where(["subject @@ ? OR " +
         "description @@ ?",
         keywords.join(", "),
         keywords.join(", ")])
```

Redmine+PGroonga：まとめ

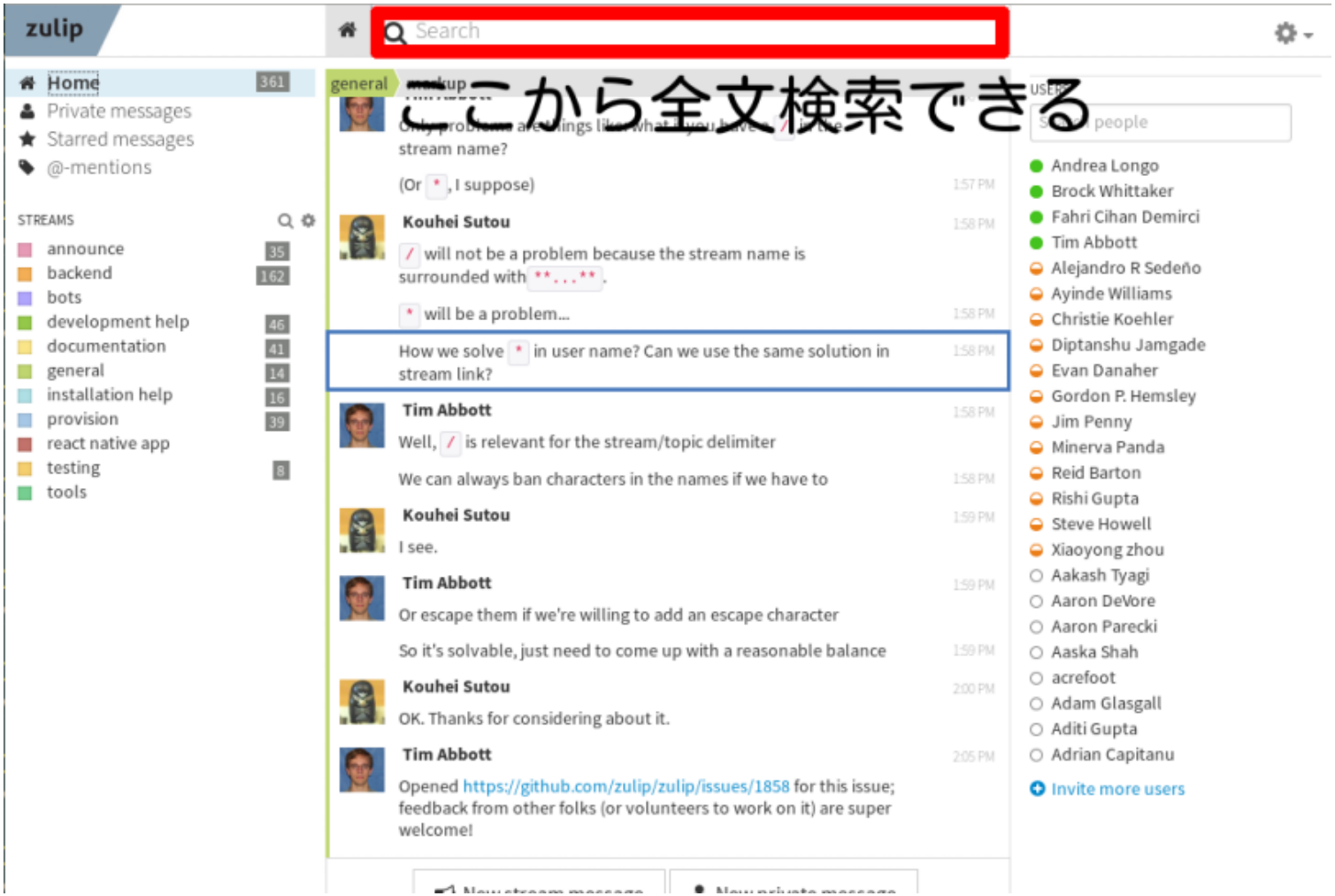
- インデックス追加のみでOK
  - トランザクション対応
  - データ保存も変更なし
- 全文検索

```
カラム1 @@ 'クエリー' OR
カラム2 @@ 'クエリー' OR ...
```

Redmine：まとめ

- 速い！
- Mroonga
  - 全文検索用テーブルで実現
- PGroonga
  - 全文検索用インデックスで実現

Zulip



ここから全文検索できる

Zulipの全文検索機能

- 現在：textsearch
  - PostgreSQL標準機能
  - 英語のみ対応
- NEW!：PGroonga
  - オプション（＝PGroongaに切替可）
  - 全言語対応（日本語を含む）

Zulip+PGroonga：方針

- 書き込み速度を落とさない
  - チャットは書き込みが遅いと微妙
  - インデックスは裏で更新
    PGroongaならリアルタイム更新でも大丈夫かも

詳細：https://github.com/zulip/zulip/pull/700/files

マイグレーション

```
migrations.RunSQL("""
ALTER ROLE zulip SET search_path
  TO zulip,public,pgroonga,pg_catalog;
ALTER TABLE zerver_message
  ADD COLUMN search_pgroonga text;
UPDATE zerver_message SET search_pgroonga =
  subject || ' ' || rendered_content;
CREATE INDEX pgrn_index ON zerver_message
  USING pgroonga(search_pgroonga);
""", "...")
```

遅延インデックス更新

- メッセージ追加・更新時にログ
- 別プロセスでログを監視

追加・更新時にログ

```
CREATE FUNCTION append_to_fts_update_log()
  RETURNS trigger
  LANGUAGE plpgsql AS $$
    BEGIN
      INSERT INTO fts_update_log (message_id)
        VALUES (NEW.id);
      RETURN NEW;
    END
  $$;
CREATE TRIGGER update_fts_index_async
  BEFORE INSERT OR UPDATE OF
    subject, rendered_content ON zerver_message
  FOR EACH ROW
    EXECUTE PROCEDURE append_to_fts_update_log();
```

別プロセスに通知

```
CREATE FUNCTION do_notify_fts_update_log()
  RETURNS trigger
  LANGUAGE plpgsql AS $$
    BEGIN
      NOTIFY fts_update_log;
      RETURN NEW;
    END
  $$;
CREATE TRIGGER fts_update_log_notify
  AFTER INSERT ON fts_update_log
  FOR EACH STATEMENT
    EXECUTE PROCEDURE do_notify_fts_update_log();
```

インデックス更新プロセス

```
import psycopg2
conn = psycopg2.connect("user=zulip")
cursor = conn.cursor
cursor.execute("LISTEN fts_update_log;")
while True:
    if select.select([conn], [], [], 30) != ([], [], []):
        conn.poll()
        while conn.notifies:
            conn.notifies.pop()
            update_fts_columns(cursor)
```

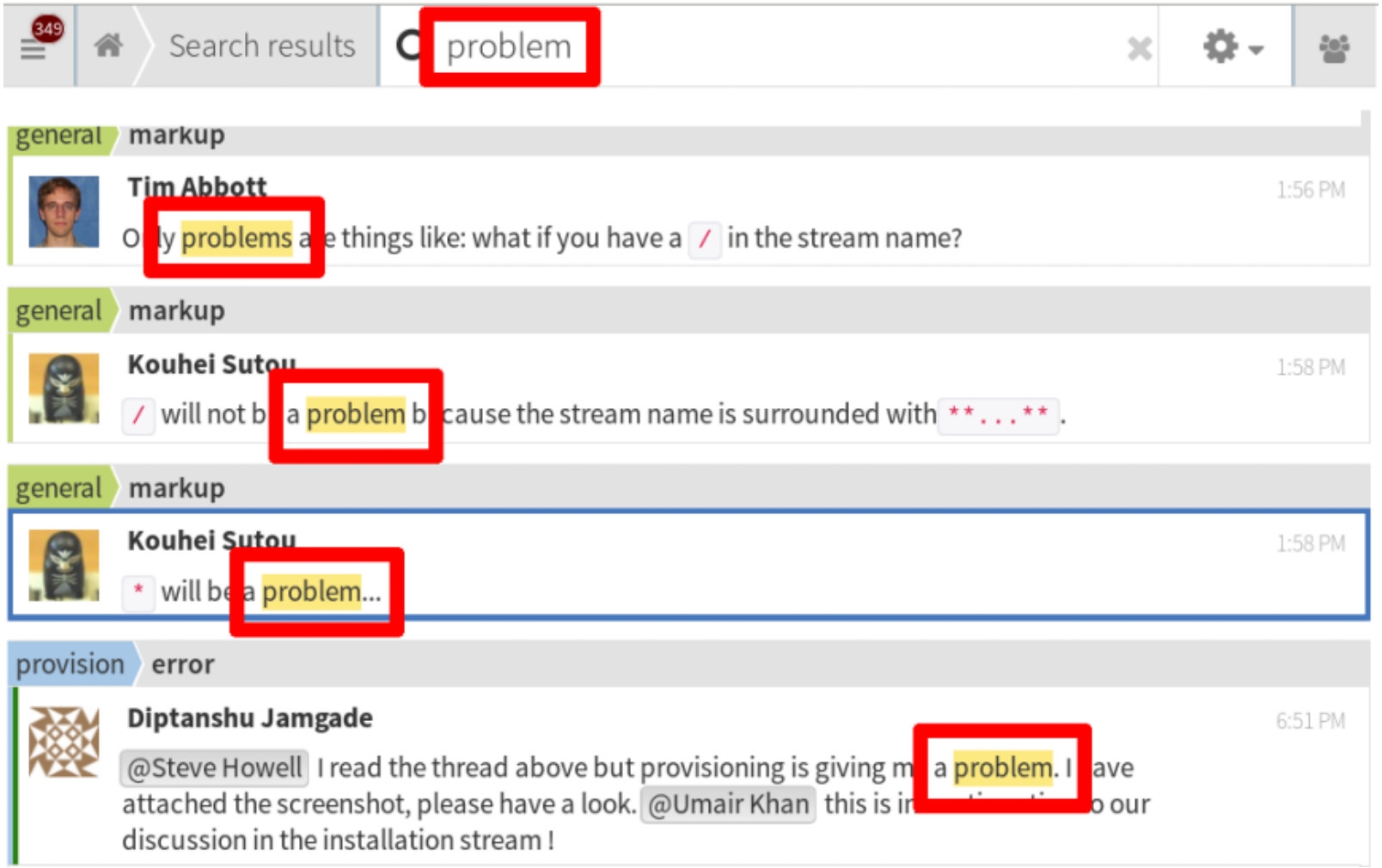
インデックス更新

```
def update_fts_columns(cursor):
    cursor.execute("SELECT id, message_id "
                   "FROM fts_update_log;")
    ids = []
    for (id, message_id) in cursor.fetchall():
        cursor.execute("UPDATE zerver_message SET "
                       "search_pgroonga = "
                       "subject || ' ' || rendered_content "
                       "WHERE id = %s", (message_id,))
        ids.append(id)
    cursor.execute("DELETE FROM fts_update_log "
                   "WHERE id = ANY(%s)", (ids,))
```

全文検索

```
from sqlalchemy.sql import column
def _by_search_pgroonga(self, query, operand):
    # WHERE search_pgroonga @@ 'クエリー'
    target = column("search_pgroonga")
    condition = target.op("@@")(operand)
    return query.where(condition)
```

ハイライト


349
Search results
problem
general markup
Tim Abbott
1:56 PM
O ly problems a e things like: what if you have a / in the stream name?
general markup
Kouhei Sutou
1:58 PM
/ will not b a problem b cause the stream name is surrounded with **...**.
general markup
Kouhei Sutou
1:58 PM
* will be a problem...
provision error
Diptanshu Jamgade
6:51 PM
@Steve Howell I read the thread above but provisioning is giving m a problem. I ave attached the screenshot, please have a look. @Umair Khan this is in o our discussion in the installation stream !

ハイライト：SQL

```
SELECT
 pgroonga.match_positions_byte(
   rendered_content,
   pgroonga.query_extract_keywords('クエリー'))
   AS content_matches,
 pgroonga.match_positions_byte(
   subject,
   pgroonga.query_extract_keywords('クエリー'))
   AS subject_matches
```

ハイライト：SQLAlchemy

```
from sqlalchemy import func
def _by_search_pgroonga(self, query, operand):
    match_positions_byte = func.pgroonga.match_positions_byte
    query_extract_keywords = func.pgroonga.query_extract_keywords
    keywords = query_extract_keywords(operand)
    query = query.column(
      match_positions_byte(column("rendered_content"),
                           keywords).label("content_matches"))
    query = query.column(
      match_positions_byte(column("subject"),
                           keywords).label("subject_matches"))
    # ...
```

ハイライト：Python

```
def highlight_string_bytes_offsets(text, locs):
    # type: (AnyStr, Iterable[Tuple[int, int]]) -> text_type
    string = text.encode('utf-8')
    highlight_start = b'<span class="highlight">'
    highlight_stop = b'</span>'
    pos = 0
    result = b''
    for loc in locs:
        (offset, length) = loc
        result += string[pos:offset]
        result += highlight_start
        result += string[offset:offset + length]
        result += highlight_stop
        pos = offset + length
    result += string[pos:]
    return result.decode('utf-8')
```

ハイライト：補足

- 通常はハイライト関数で十分
  - pgroonga.highlight_html
- ただしts_headlineでは不十分
  - HTML出力に使えない

ハイライト：ts_headline

```
SELECT
  ts_headline('english',
              'PostgreSQL <is> great!',
              to_tsquery('PostgreSQL'),
              'HighlightAll=TRUE');
--              ts_headline
-- ------------------------------
--  <b>PostgreSQL</b> <is> great!
-- (1 row)  不正なHTML↑
```

ハイライト：pgroonga.highlight_html

```
SELECT
  pgroonga.highlight_html(
    'PostgreSQL <is> great!',
    pgroonga.query_extract_keywords('PostgreSQL'));
--                 highlight_html
-- ---------------------------------------
--  <span class="keyword">PostgreSQL</span>
--     &lt;is&gt; great!
--     ↑      ↑HTMLエスケープされている
```

Zulip：まとめ

- 全言語対応全文検索
  - textsearch（1言語のみ）→ PGroonga（全言語）
- 書き込み性能は維持
  - 遅延インデックス更新

まとめ

- Mroonga・PGroongaの導入方法を実例ベースで紹介
  - Redmine：チケット管理システム
  - Zulip：チャットツール
- トランザクション必須の場合
  - Mroonga：別テーブル作成
  - PGroonga：インデックス追加