**Tìm hiểu về pagination trong laravel**

This post has been more than **4 years** since it was last updated.

**1. Giới thiệu**

Trong các framework khác, pagination có thể khá là đau đầu, còn Laravel thì làm cho nó trở nên đơn giản hơn rất nhiều nó kết hợp với `query builder` và `eloquent ORM` và cung cấp phân trang của dữ liệu trong database một cách thuận tiện. Laravel có thể nhanh chóng tạo một khoảng thông minh của các links dựa trên trang hiện tại và mã HTML sinh ra thì tương thích với `Bootstrap CSS framework`.

**2. Sử dụng cơ bản**

* Phân trang kết quả từ Query Builder Có vài cách để phân trang. Đơn giản nhất là sử dụng hàm paginate ở trong query builder hoặc trong Eloquent query. Hàm paginate cung cấp bởi Laravel sẽ tự động xử lý việc tạo ra limit và vị trí trang dựa trên trang hiện tại đang được xem bởi người dùng. Mặc định, trang hiện tại được nhận biết thông qua giá trị ?page trên query string trên HTTP request. Dĩ nhiên là giá trị này được tự động nhận biết bởi Laravel, và cũng được tự động thêm vào các link sinh ra bởi paginator.

Trước tiên, hãy cùng nhau xem việc gọi hàm paginate trên một query. Trong ví dụ này, đối số duy nhất tryền vào hàm paginate là số items bạn muốn hiển thị trên từng page. Ở đây, chúng ta ví dụ đặt chỉ số là 15 item trên một page:

<?php

namespace App\Http\Controllers;

use DB;

use App\Http\Controllers\Controller;

class UserController extends Controller

{

/\*\*

\* Show all of the users for the application.

\*

\* @return Response

\*/

public function index()

{

$users = DB::table('users')->paginate(15);

return view('user.index', ['users' => $users]);

}

}

Chú ý: Hiện tại, việc phân trang sử dụng groupBy chưa thể thực thi hiệu quả bởi Laravel. Nếu bạn cần sử dụng groupBy với một tập kết quả phân trang, thì khuyến khích các bạn thực hiện query database và tạo một paginator thủ công.

* **Phân trang đơn giản**

Nếu bạn chỉ cần hiển thị hai link đơn giản "Next" và "Previous" trên pagination view, bạn có thể sử dụng hàm simplePaginate để thực hiện một query hiệu quả hơn. Cách này rất hữu dụng với một tập dữ liệu lớn nếu bạn không cần hiển thị một link cho mỗi số trang khi thực hiện render:

$users = DB::table('users')->simplePaginate(15);

* **Phân trang kết quả từ Eloquent**

Bạn cũng có thể phân trang kết quả từ Eloquent. Trong ví dụ này, chúng ta sẽ phân trang User model với 15 item một trang. Như bạn thấy, cú pháp gần như giống hệt với phân trang kết quả từ query builder:

$users = App\User::paginate(15);

Bạn cũng có thể gọi paginate sau khi thiết lập rằng buộc trên query, ví dụ như mệnh đề where:

$users = User::where('votes', '>', 100)->paginate(15);

Bạn cũng có thể sử dụng simplePaginate khi phân trang với Eloquent:

$users = User::where('votes', '>', 100)->simplePaginate(15);

* **Tạo thủ công một Paginator**

Đôi khi bạn muốn tạo một đối tượng xử lý phân trang riêng, truyền vào cho nó một mảng các items. Bạn có thể thực hiện bằng cách tạo một đối tượng từ Illuminate\Pagination\Paginator hoặc từ Illuminate\Pagination\LengthAwarePaginator, phục thuộc vào yêu cầu của bạn.

Class Paginator không quan tâm tổng số items trên tập kết quả; tuy nhiên, chính vì thế mà class không có phương thức để lấy được index của trang cuối cùng. Class LengthAwarePaginator nhận đối số tương tự với Paginator, nhưng lại cần biết tổng số items có trong tập kết quả.

Nói một cách khác, Paginator tương ứng với hàm simplePaginate trên query builder và Eloquent, trong khi LengthAwarePaginator lại tương ứng với hàm paginate.

Khi tự tạo một đối tượng paginator thủ công, bạn nên tự "cắt" mảng của tập kết quả truyền vào cho paginator. Nếu bạn không chắc làm như thế nào, hãy tham khảo hàm array\_slice của PHP.

* **Hiển thị kết quả lên views**

Khi bạn gọi hàm paginate hay simplePaginate trên query builder hay Eloquent, bạn sẽ nhận được một đối tượng paginator. Khi gọi hàm paginate, bạn sẽ nhận được một đối tượng của Illuminate\Pagination\LengthAwarePaginator. Khi gọi hàm simplePaginate, bạn sẽ nhận được một đối tượng của Illuminate\Pagination\Paginator. Những đối tượng này cung cấp vài phương thức mô tả tập kết quả. Ngoài những phương thức này, các đối tượng paginator đều là các iterators và có thể được lặp như một mảng.

Khi đã nhận được kết quả, bạn có thể thực hiện hiển thị kết quả và render các link vào page sử dụng Blade:

<div class="container">

@foreach ($users as $user)

{{ $user->name }}

@endforeach

</div>

{!! $users->links() !!}

Hàm links sẽ render các link cho tới hết các trang trong tập kết quả. Mỗi link này đều chứa sẵn một tham số ?page với giá trị đúng. Hãy nhớ là, mã HTML sinh ra bởi hàm links tương thích với Bootstrap CSS framework.

* **Tuỳ chọn The Paginator URI** Hàm setPath cho phép bạn tuỳ chọn URI sử dụng bởi paginator khi sinh ra links. Ví dụ, nếu bạn muốn paginator sinh ra links theo kiểu này <http://example.com/custom/url?page=N>, bạn chỉ cần truyền vào custom/url vào hàm setPath:

Route::get('users', function () {

$users = App\User::paginate(15);

$users->setPath('custom/url');

//

});

* **Thêm vào link phân trang**

Bạn có thể thêm vào query string của link phân trang sử dụng hàm appends. Ví dụ, để thêm vào &sort=votes vào mỗi link, bạn nên thực hiện thế này:

{!! $users->fragment('foo')->links() !!}

-\*\* Các phương thức helper bổ sung\*\*

Bạn có thể truy cập các thông tin khác trong phân trang thông qua các phương thức sau trong paginator:

$results->count()

$results->currentPage()

$results->firstItem()

$results->hasMorePages()

$results->lastItem()

$results->lastPage() (Not available when using simplePaginate)

$results->nextPageUrl()

$results->perPage()

$results->previousPageUrl()

$results->total() (Not available when using simplePaginate)

$results->url($page)

* **Chuyển kết quả sang JSON**

Các lớp kết quả phân trang của Laravel triển khai từ contract Illuminate\Contracts\Support\JsonableInterface và mở ra hàm toJson, do đó, rất dễ dàng để có thể chuyển kết quả thành JSON.

Bạn cũng có thể convert một đối tượng paginator sang JSON bằng cách return nó từ một route hay controller action:

Route::get('users', function () {

return App\User::paginate();

});

JSON tạo ra từ paginator sẽ chứa các thông tin meta như total, current\_page, last\_page, và nhiều nữa. Các đối tượng kết quả đều có trong khoá data của mảng JSON. Đây là một ví dụ về JSON tạo bởi paginator từ một route:

{

"total": 50,

"per\_page": 15,

"current\_page": 1,

"last\_page": 4,

"next\_page\_url": "http://laravel.app?page=2",

"prev\_page\_url": null,

"from": 1,

"to": 15,

"data":[

{

// Result Object

},

{

// Result Object

}

]

}