Course - Laravel Framework

Phân trang

Trong các framework khác, việc phân trang có thể rất khó khăn. Chúng tôi hy vọng cách tiếp cận phân trang của Laravel sẽ là một luồng gió mới. Chức năng phân trang của Laravel được tích hợp với query builder và Eloquent ORM và sẽ cung cấp tính năng phân trang thuận tiện, dễ sử dụng cho các record cơ sở dữ liệu mà không cần phải cấu hình phức tạp.

Tags: pagination, phan trang, laravel

Giới thiệu

Trong các framework khác, việc phân trang có thể rất khó khăn. Chúng tôi hy vọng cách tiếp cận phân trang của Laravel sẽ là một luồng gió mới. Chức năng phân trang của Laravel được tích hợp với query builder và Eloquent ORM và sẽ cung cấp tính năng phân trang thuận tiện, dễ sử dụng cho các record cơ sở dữ liệu mà không cần phải cấu hình phức tạp.

Theo mặc định, HTML được tạo bởi trình phân trang tương thích với khung CSS Tailwind; tuy nhiên, hỗ trợ phân trang Bootstrap cũng có sẵn.

Tailwind JIT

Nếu bạn đang sử dụng các chế độ xem phân trang Tailwind mặc định của Laravel và công cụ Tailwind JIT, bạn nên đảm bảo rằng khóa `content` của tập tin *tailwind.config.js* trong ứng dụng của bạn tham chiếu đến các giao diện hiển thị phân trang của Laravel để các lớp Tailwind của chúng không bị loại bỏ:

```
content: [
  './resources/**/*.blade.php',
  './resources/**/*.js',
  './resources/**/*.vue',
  './vendor/laravel/framework/src/Illuminate/Pagination/resources/views/*.blade.php',
],
```

Cách dùng căn bản

Phân trang kết quả của query builder

Có một số cách để phân trang các mục. Đơn giản nhất là sử dụng phương pháp phân trang trên query builder hoặc truy vấn Eloquent. Phương thức phân trang tự động đảm nhận việc đặt "limit" và "offset" của truy vấn dựa trên trang hiện tại đang được người dùng xem. Theo mặc định, trang hiện tại được phát hiện bởi giá trị của đối số bởi chuỗi truy vấn `page` trên yêu cầu HTTP request. Giá trị này được Laravel tự động phát hiện và cũng được tự động chèn vào các liên kết do trình phân trang tạo ra.

Trong ví dụ này, đối số duy nhất được truyền cho phương thức paginate là số lượng mục mà bạn muốn hiển thị "trên mỗi trang". Trong trường hợp này, chúng ta muốn hiển thị 15 mục trên mỗi trang:

```php
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\DB;

class UserController extends Controller
{
  /**
   * Show all application users.
   *
   * @return \Illuminate\Http\Response
   */
  public function index()
  {
    return view('user.index', [
      'users' => DB::table('users')->paginate(15)
    ]);
  }
}
```

Phân trang đơn giản

Phương thức `paginate` đếm tổng số record được kiểm tra khớp bởi truy vấn trước khi truy xuất record từ cơ sở dữ liệu. Điều này được thực hiện để người phân trang biết tổng cộng có bao nhiêu trang. Tuy nhiên, nếu bạn không có kế hoạch hiển thị tổng số trang trong giao diện người dùng của ứng dụng thì truy vấn đếm record là không cần thiết.

Do đó, nếu bạn chỉ cần hiển thị các liên kết đơn giản "Next" và "Previous" trong giao diện người dùng của ứng dụng, thì bạn có thể sử dụng phương thức `simplePaginate` để thực hiện một truy vấn hiệu quả:

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

Phân trang các kết quả Eloquent

Bạn cũng có thể phân trang với các truy vấn Eloquent. Trong ví dụ này, chúng tôi sẽ phân trang model `App\Models\User` và dự định sẽ hiển thị 15 record trên mỗi trang. Như bạn có thể thấy, cú pháp gần giống với việc phân trang với kết quả của query builder:

```
use App\Models\User;

$users = User::paginate(15);
```

Tất nhiên, bạn có thể gọi phương thức paginate sau khi đặt các ràng buộc khác trên truy vấn, chẳng hạn như mệnh đề `where`:

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

Bạn cũng có thể sử dụng phương thức `simplePaginate` khi phân trang các mô hình Eloquent:

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

Tương tự, bạn có thể sử dụng phương thức `cursorPaginate` để phân trang cursor các model Eloquent:

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

Phân trang cursor là một kỹ thuật phân trang hiệu suất cao thường được sử dụng cho các tập dữ liệu lớn, cuộn vô hạn và ứng dụng nhiều trong các API. Các framework Laravel mới đã bổ sung phương thức `cursorPaginate` vào các Eloquent và query builder, cách sử dụng phân trang cursor như bên dưới.

```
use App\Models\User;

$users = User::orderBy('id')->cursorPaginate(10);
```

Đoạn code chạy giống hệt như `simplePaginate`, ngoại trừ việc nó kích hoạt một truy vấn khác. Truy vấn bên dưới sẽ được kích hoạt bởi phân trang cursor nếu bạn truy cập vào trang thứ 2nd:

```
select * from users where `id` > 10 order by `id` asc limit 10;
```

Sự khác biệt chính giữa phân trang offset và phân trang cursor là phân trang offset sử dụng mệnh đề offset trong khi phân trang cursor sử dụng mệnh đề *where* với toán tử so sánh.

Nhiều đối tượng Paginator trên mỗi trang

Đôi khi bạn có thể cần phải hiển thị hai sự phân trang riêng biệt trên cùng một trang duy nhất. Tuy nhiên, nếu cả hai trường hợp phân trang đều sử dụng tham số chuỗi truy vấn page để lưu trữ trang hiện tại, thì hai trình phân trang sẽ xung đột. Để giải quyết xung đột này, bạn có thể chuyển tên của tham số chuỗi truy vấn mà bạn muốn sử dụng để lưu trữ trang hiện tại của trình phân trang thông qua đối số thứ ba được cung cấp cho các phương thức `paginate`, `simplePaginate` và `cursorPaginate`:

```
use App\Models\User;

$users = User::where('votes', '>', 100)->paginate(
  $perPage = 15, $columns = ['*'], $pageName = 'users'
);
```

Phân trang cursor

Trong khi `paginate` và `simplePaginate` tạo truy vấn bằng mệnh đề SQL "offset", phân trang cursor hoạt động bằng cách xây dựng mệnh đề "where" so sánh giá trị của các cột được sắp xếp có trong truy vấn, cung cấp hiệu suất cơ sở dữ liệu hiệu quả nhất hiện có trong số tất cả các phương thức paginate của Laravel. Phương thức paginate này đặc biệt thích hợp cho các tập dữ liệu lớn và giao diện người dùng cuộn "vô hạn".

Không giống như phân trang dựa trên offset, bao gồm một số trang trong chuỗi truy vấn của URL được tạo bởi trình phân trang, phân trang dựa trên cursor đặt một chuỗi "cursor" trong chuỗi truy vấn. Cursor là một chuỗi được mã hóa chứa vị trí mà truy vấn được phân trang tiếp theo sẽ bắt đầu phân trang và hướng mà nó sẽ phân trang:

```
http://localhost/users?cursor=eyJpZCI6MTUsIl9wb2ludHNUb05leHRJdGVtcyI6dHJ1ZX0
```

Bạn có thể tạo một đối tượng phân trang dựa trên cursor thông qua phương thức `cursorPaginate` được cung cấp bởi query builder. Phương thức này trả về một đối tượng

của `Illuminate\Pagination\CursorPaginator`:

```
$users = DB::table('users')->orderBy('id')->cursorPaginate(15);
```

Khi bạn đã truy xuất một đối tượng phân trang cursor, bạn có thể hiển thị kết quả phân trang như bạn thường làm khi sử dụng các phương thức `paginate` và `simplePaginate`. Để biết thêm thông tin về các phương thức được cung cấp bởi trình phân trang cursor, xin vui lòng tham khảo tài liệu về các phương thức của trình phân trang cursor.

> **Chú ý:** Truy vấn của bạn phải chứa mệnh đề "order by" để tận dụng việc phân trang cursor.

Khác biệt giữa phân trang cursor và offset

Để minh họa sự khác biệt giữa phân trang offset và phân trang cursor, chúng ta hãy xem xét một số truy vấn SQL mẫu. Cả hai truy vấn sau đều sẽ hiển thị "trang thứ hai" của kết quả cho bảng users được sắp xếp theo `id`:

```
# Offset Pagination...
select * from users order by id asc limit 15 offset 15;

# Cursor Pagination...
select * from users where id > 15 order by id asc limit 15;
```

Truy vấn phân trang cursor cung cấp các ưu điểm sau so với phân trang offset:

- Đối với tập dữ liệu lớn, phân trang cursor sẽ mang lại hiệu suất tốt hơn nếu các cột "theo thứ tự" được lập chỉ mục. Điều này là do mệnh đề "offset" quét qua tất cả các dữ liệu đã so khớp trước đó.
- Đối với các tập dữ liệu thường xuyên write, phân trang offset có thể bỏ qua các record hoặc hiển thị các bản sao không cần thiết nếu kết quả đã được thêm vào hoặc bị xóa khỏi trang mà người dùng hiện đang xem.

Tuy nhiên, phân trang cursor có những hạn chế sau:

- Giống như `simplePaginate`, phân trang cursor chỉ có thể được sử dụng để hiển thị các liên kết "Next" và "Previous" và không hỗ trợ tạo liên kết với số trang.
- Nó yêu cầu thứ tự dựa trên ít nhất một cột là *unique* hoặc kết hợp các cột là *unique*. Các cột có giá trị `null` không được hỗ trợ.

- Biểu thức truy vấn trong mệnh đề "order by" chỉ được hỗ trợ nếu chúng được đặt bí danh và cũng được thêm vào mệnh đề "select".

Tạo thủ công đối tượng phân trang

Đôi khi bạn có thể muốn tạo một đối tượng phân trang theo cách thủ công, truyền nó vào một mảng các mục mà bạn đã có trong bộ nhớ. Bạn có thể làm như vậy bằng cách tạo một đối tượng `Illuminate\Pagination\Paginator`, `Illuminate\Pagination\LengthAwarePaginator` hoặc `Illuminate\Pagination\CursorPaginator`, tùy thuộc vào nhu cầu của bạn.

Các class `Paginator` và `CursorPaginator` không cần biết tổng số mục trong tập kết quả; tuy nhiên, do các class này không có các phương thức để lấy chỉ mục của trang cuối cùng. `LengthAwarePaginator` chấp nhận các đối số gần giống như `Paginator`; tuy nhiên, nó yêu cầu đếm tổng số mục trong tập kết quả.

Nói cách khác, `Paginator` tương ứng với phương thức `simplePaginate` trên query builder, `CursorPaginator` thì tương ứng với phương thức `cursorPaginate` và `LengthAwarePaginator` tương ứng với phương thức paginate.

> **Chú ý:** Khi tạo thủ công một đối tượng phân trang, bạn nên "xén" mảng kết quả mà bạn truyền cho đối tượng phân trang theo cách thủ công. Nếu bạn không biết cách thực hiện việc này, hãy tham khảo hàm `array_slice` PHP.

Tùy biến URL phân trang

Theo mặc định, các liên kết được tạo bởi đối tượng phân trang sẽ khớp với URI của yêu cầu hiện tại. Tuy nhiên, phương thức `withPath` của paginator cho phép bạn tùy chỉnh URI được sử dụng bởi paginator khi tạo liên kết. Ví dụ: nếu bạn muốn trình phân trang tạo các liên kết như *http://example.com/admin/users?page=N*, thì bạn nên truyền */admin/users* vào phương thức `withPath`:

```php
use App\Models\User;

Route::get('/users', function () {
  $users = User::paginate(15);
  $users->withPath('/admin/users');
  //
});
```

Thêm bồi các giá trị chuỗi truy vấn

Bạn có thể thêm vào chuỗi truy vấn của các đường link phân trang bằng phương thức `appends`. Ví dụ: để nối thêm *sort=votes* vào mỗi đường link phân trang, thì bạn nên thực hiện lệnh gọi `appends` thêm sau:

```php
use App\Models\User;

Route::get('/users', function () {
  $users = User::paginate(15);
  $users->appends(['sort' => 'votes']);
  //
});
```

Bạn có thể sử dụng phương thức `withQueryString` nếu bạn muốn bồi thêm tất cả các giá trị chuỗi truy vấn của yêu cầu hiện tại vào các đường link phân trang:

```php
$users = User::paginate(15)->withQueryString();
```

Hiển thị các kết quả phân trang

Khi gọi phương thức paginate, bạn sẽ nhận được một đối tượng của `Illuminate\Pagination\LengthAwarePaginator`, trong khi gọi phương thức `simplePaginate` sẽ trả về một đối tượng của `Illuminate\Pagination\Paginator`. Và cuối cùng, việc gọi phương thức `cursorPaginate` sẽ trả về một đối tượng của `Illuminate\Pagination\CursorPaginator`.

Các đối tượng này cung cấp một số phương thức mà chúng có thể mô tả tập hợp kết quả. Ngoài các phương thức có ích này, các đối tượng của trình phân trang còn có các phần tử lặp (iterator) và có thể được lặp lại dưới dạng một mảng. Vì vậy, khi bạn đã truy xuất kết quả, bạn có thể hiển thị kết quả và hiển thị các liên kết trang bằng Blade template như sau:

```
<div class="container">
    @foreach ($users as $user)
        {{ $user->name }}
    @endforeach
</div>

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

Phương thức `links` sẽ hiển thị các liên kết đến phần còn lại của các trang trong tập kết quả. Mỗi liên kết này sẽ chứa biến chuỗi truy vấn page thích hợp. Hãy nhớ rằng HTML được tạo bởi phương thức `links` tương thích với khung CSS Tailwind.

Điều chỉnh đường link phân trang

Khi trình phân trang hiển thị các đường link phân trang, số trang hiện tại được hiển thị cũng như các liên kết cho ba trang trước, sau và hiện tại. Bằng việc dùng phương thức `onEachSide`, bạn có thể kiểm soát số lượng đường link bổ sung được hiển thị trên mỗi bên của trang hiện tại nằm giữa của các liên kết được tạo bởi đối tượng phân trang:

```
{{ $users->onEachSide(5)->links() }}
```

Chuyển đổi các kết quả vào JSON

Các class phân trang của Laravel được thực thi từ contract `Illuminate\Contracts\Support\Jsonable` nên nó có phương thức `toJson`, vì vậy sẽ rất dễ dàng chuyển đổi kết quả phân trang của bạn sang JSON. Bạn cũng có thể chuyển đổi một đối tượng paginator thành JSON bằng cách trả về nó từ action của route hoặc controller:

```
use App\Models\User;

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

```
    return User::paginate();
});
```

JSON từ đối tượng phân trang sẽ bao gồm thông tin meta như total, current_page, last_page và vân vân. Các record kết quả sẽ được sẵn sàng thông qua khóa dữ liệu trong mảng JSON. Dưới đây là một ví dụ về JSON được tạo bằng cách trả về một đối tượng phân trang từ một route:

```
{
  "total": 50,
  "per_page": 15,
  "current_page": 1,
  "last_page": 4,
  "first_page_url": "http://laravel.app?page=1",
  "last_page_url": "http://laravel.app?page=4",
  "next_page_url": "http://laravel.app?page=2",
  "prev_page_url": null,
  "path": "http://laravel.app",
  "from": 1,
  "to": 15,
  "data":[
    {
      // Record...
    },
    {
      // Record...
    }
  ]
}
```

Tùy chỉnh cho phần hiển thị phân trang

Theo mặc định, các giao diện hiển thị được hiển thị để hiển thị các đường link phân trang tương thích với framework CSS Tailwind. Tuy nhiên, nếu bạn không sử dụng Tailwind, bạn có thể tự tạo giao diện hiển thị của riêng mình để hiển thị các đường link này. Khi gọi phương thức `links` trên một đối tượng phân trang, bạn có thể truyền tên giao diện hiển thị làm đối số đầu tiên cho phương thức:

```
{{ $paginator->links('view.name') }}

// Passing additional data to the view...
{{ $paginator->links('view.name', ['foo' => 'bar']) }}
```

Tuy nhiên, cách dễ nhất để tùy chỉnh các giao diện phân trang là xuất chúng sang thư mục *resources/views/vendor* của bạn bằng lệnh `vendor:publish`:

```
php artisan vendor:publish --tag=laravel-pagination
```

Lệnh này sẽ đặt các giao diện hiển thị trong thư mục *resources/views/vendor/pagination* của ứng dụng của bạn. Tập tin *tailwind.blade.php* trong thư mục này tương ứng với giao diện phân trang mặc định. Bạn có thể chỉnh sửa tập tin này để sửa đổi HTML phân trang.

Nếu bạn muốn chỉ định một tập tin khác làm giao diện phân trang mặc định, thì bạn có thể gọi các phương thức `defaultView` và `defaultSimpleView` của đối tượng phân trang trong phương thức boot của class `App\Providers\AppServiceProvider`:

```
<?php

namespace App\Providers;

use Illuminate\Pagination\Paginator;
use Illuminate\Support\Facades\Blade;
use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider
{
  /**
   * Bootstrap any application services.
   *
   * @return void
   */
  public function boot()
  {
    Paginator::defaultView('view-name');
```

```
    Paginator::defaultSimpleView('view-name');
  }
}
```

Sử dụng bootstrap

Laravel bao gồm các giao diện phân trang được xây dựng bằng Bootstrap CSS. Để sử dụng các giao diện này thay vì giao diện Tailwind mặc định, bạn có thể gọi phương thức `useBootstrap` của `Paginator` trong phương thức `boot` của class `App\Providers\AppServiceProvider`:

```
use Illuminate\Pagination\Paginator;

/**
 * Bootstrap any application services.
 *
 * @return void
 */
public function boot()
{
  Paginator::useBootstrap();
}
```

Các phương thức của Paginator/LengthAwarePaginator

Mỗi đối tượng phân trang cung cấp thông tin phân trang bổ sung thông qua các phương thức sau:

Phương thức	Mô tả
`$paginator->count()`	Lấy số lượng mục cho trang hiện tại.
`$paginator->currentPage()`	Lấy số trang hiện tại.
`$paginator->firstItem()`	Lấy kết quả của mục đầu tiên trong kết quả.

Phương thức	Mô tả
`$paginator->getOptions()`	Nhận các tùy chọn phân trang.
`$paginator->getUrlRange($start, $end)`	Tạo một loạt các URL phân trang.
`$paginator->hasPages()`	Xác định xem có đủ mục để chia thành nhiều trang hay không.
`$paginator->hasMorePages()`	Xác định xem có nhiều mục hơn trong kho dữ liệu hay không.
`$paginator->items()`	Nhận các mục cho trang hiện tại.
`$paginator->lastItem()`	Lấy kết quả của mục cuối cùng trong kết quả.
`$paginator->lastPage()`	Lấy số trang của trang cuối cùng có sẵn. (Không khả dụng khi sử dụng `simplePaginate`).
`$paginator->nextPageUrl()`	Lấy URL cho trang tiếp theo.
`$paginator->onFirstPage()`	Xác định xem trình phân trang có trên trang đầu tiên hay không.
`$paginator->perPage()`	Số lượng mục sẽ được hiển thị trên mỗi trang.
`$paginator->previousPageUrl()`	Lấy URL cho trang trước.
`$paginator->total()`	Xác định tổng số mục phù hợp trong kho dữ liệu. (Không khả dụng khi sử dụng `simplePaginate`).
`$paginator->url($page)`	Lấy URL cho một số trang nhất định.
`$paginator->getPageName()`	Lấy tên biến chuỗi truy vấn được sử dụng để lưu trang hiện tại.

Phương thức	Mô tả
`$paginator->setPageName($name)`	Đặt tên biến chuỗi truy vấn được sử dụng để lưu trữ trang hiện tại.

Bảng các phương thức phân trang

Các phương thức trong phân trang cursor

Mỗi đối tượng phân trang con trỏ cung cấp thông tin phân trang bổ sung với các phương thức sau:

Method	Description
`$paginator->count()`	Lấy số lượng mục cho trang hiện tại.
`$paginator->cursor()`	Lấy đối tượng con trỏ hiện tại.
`$paginator->getOptions()`	Nhận các tùy chọn phân trang.
`$paginator->hasPages()`	Xác định xem có đủ mục để chia thành nhiều trang hay không.
`$paginator->hasMorePages()`	Xác định xem có nhiều mục hơn trong kho dữ liệu hay không.
`$paginator->getCursorName()`	Lấy tên biến chuỗi truy vấn dùng để lưu con trỏ.
`$paginator->items()`	Nhận các mục cho trang hiện tại.
`$paginator->nextCursor()`	Nhận đối tượng con trỏ cho tập hợp các mục tiếp theo.
`$paginator->nextPageUrl()`	Lấy URL cho trang tiếp theo.
`$paginator->onFirstPage()`	Xác định xem trình phân trang có trên trang đầu tiên hay không.
`$paginator->perPage()`	Số lượng mục sẽ được hiển thị trên mỗi trang.
`$paginator->previousCursor()`	Lấy đối tượng con trỏ cho tập hợp các mục trước đó.

Method	Description
`$paginator->previousPageUrl()`	Lấy URL cho trang trước.
`$paginator->setCursorName()`	Đặt tên biến chuỗi truy vấn được sử dụng để lưu trữ con trỏ.
`$paginator->url($cursor)`	Lấy URL cho một đối tượng con trỏ nhất định.