Course - Laravel Framework

Lịch trình tác vụ

Trước đây, bạn có thể đã viết cấu hình cron cho các tác vụ cần lên lịch trên máy chủ của mình. Tuy nhiên, điều này có thể nhanh chóng trở thành vấn đề vì lịch trình tác vụ của bạn không còn quyền kiểm soát nguồn điện và bạn phải SSH vào máy chủ của mình để xem các mục cron hiện có hoặc cập nhật thêm các chi tiết bổ sung.

Tags: task scheduling, lich trinh tac vu, laravel

Giới thiệu

Trước đây, bạn có thể đã viết cấu hình cron cho các tác vụ cần lên lịch trên máy chủ của mình. Tuy nhiên, điều này có thể nhanh chóng trở thành vấn đề vì lịch trình tác vụ của bạn không còn quyền kiểm soát nguồn điện và bạn phải SSH vào máy chủ của mình để xem các mục cron hiện có hoặc cập nhật thêm các chi tiết bổ sung.

Scheduler của Laravel đưa ra một cách tiếp cận mới trong việc quản lý các tác vụ đã được lên lịch trên máy chủ của bạn. Scheduler cho phép bạn xác định lịch biểu lệnh một cách trôi chảy và rõ ràng trong chính ứng dụng Laravel của bạn. Khi sử dụng scheduler, chỉ cần một mục nhập cron duy nhất trên máy chủ của bạn. Lịch trình tác vụ của bạn được xác định trong phương thức schudule của tập tin `app/Console/Kernel.php`. Để giúp bạn bắt đầu, một ví dụ đơn giản được xác định trong phương thức.

Xác định lịch biểu

Bạn có thể xác định tất cả các tác vụ đã lên lịch của mình trong phương thức lịch biểu của lớp `App\Console\Kernel` của ứng dụng. Để bắt đầu, chúng ta hãy xem một ví dụ. Trong ví dụ này, chúng ta sẽ lên lịch cho một hàm nặc danh mà sẽ được gọi hằng ngày vào lúc nửa đêm. Trong hàm này, chúng ta sẽ thực hiện một truy vấn cơ sở dữ liệu để xóa một bảng nào đó:

```
<?php

namespace App\Console;

use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;
use Illuminate\Support\Facades\DB;

class Kernel extends ConsoleKernel
{
  /**
   * Define the application's command schedule.
   *
   * @param  \Illuminate\Console\Scheduling\Schedule  $schedule
   * @return void
   */
```

```
  protected function schedule(Schedule $schedule)
  {
    $schedule->call(function () {
      DB::table('recent_users')->delete();
    })->daily();
  }
}
```

Ngoài việc lên lịch bằng cách sử dụng các hàm nặc danh, bạn cũng có thể lên lịch cho các đối tượng. Đối tượng có thể truy cập được này là các lớp PHP đơn giản có chứa phương thức `__invoke`:

```
$schedule->call(new DeleteRecentUsers)->daily();
```

Nếu bạn muốn xem tổng quan về các tác vụ đã lên lịch của mình và vào lần tiếp theo chúng được lên lịch chạy, bạn có thể sử dụng lệnh Artisan `schedule:list`:

```
php artisan schedule:list
```

Lệnh artisan cho schedule

Ngoài việc lên lịch với hàm nặc danh, bạn cũng có thể lên lịch với lệnh Artisan và các lệnh hệ thống khác. Ví dụ: bạn có thể sử dụng phương thức command để lên lịch một lệnh Artisan bằng cách sử dụng tên hoặc class của lệnh.

Khi lên lịch với các lệnh Artisan bằng cách sử dụng tên class của lệnh, bạn có thể truyền một mảng các đối số command-line, chúng sẽ được bổ sung cho lệnh khi được gọi:

```
use App\Console\Commands\SendEmailsCommand;

$schedule->command('emails:send Taylor --force')->daily();

$schedule->command(SendEmailsCommand::class, ['Taylor', '--force'])->daily();
```

Lên lịch cho job

Phương thức `job` có thể được sử dụng để lên lịch cho một job đã được xếp hàng đợi. Phương thức này cho phép thuận tiện việc lên lịch các công việc trong một hàng chờ mà không cần sử dụng phương thức call để khai báo hàm nặc danh cho công việc hàng chờ:

```
use App\Jobs\Heartbeat;

$schedule->job(new Heartbeat)->everyFiveMinutes();
```

Đối số thứ hai và thứ ba tùy chọn có thể được đem vào phương thức job chỉ định tên hàng đợi và kết nối hàng đợi sẽ được sử dụng để xếp hàng công việc:

```
use App\Jobs\Heartbeat;

// Dispatch the job to the "heartbeats" queue on the "sqs" connection...
$schedule->job(new Heartbeat, 'heartbeats', 'sqs')->everyFiveMinutes();
```

Lên lịch các lệnh Shell

Phương thức `exec` có thể được sử dụng để đề xuất một lệnh cho hệ điều hành:

```
$schedule->exec('node /home/forge/script.js')->daily();
```

Các tùy chọn của schedule

Chúng ta đã xem qua một vài ví dụ về cách mà bạn có thể cấu hình một tác vụ để chạy trong các khoảng thời gian xác định. Tuy nhiên, có rất nhiều tần suất lịch trình khác nhau mà bạn có thể chỉ định cho một tác vụ:

Phương thức	Mô tả
`->cron('* * * * *');`	Chạy tác vụ theo lịch trình cron tùy chỉnh
`->everyMinute();`	Chạy tác vụ mỗi phút
`->everyTwoMinutes();`	Chạy tác vụ hai phút một lần

Phương thức	Mô tả
`->everyThreeMinutes();`	Chạy tác vụ ba phút một lần
`->everyFourMinutes();`	Chạy tác vụ bốn phút một lần
`->everyFiveMinutes();`	Chạy tác vụ năm phút một lần
`->everyTenMinutes();`	Chạy tác vụ mười phút một lần
`->everyFifteenMinutes();`	Chạy tác vụ cứ sau mười lăm phút một lần
`->everyThirtyMinutes();`	Chạy tác vụ ba mươi phút một lần
`->hourly();`	Chạy tác vụ mỗi giờ
`->hourlyAt(17);`	Chạy tác vụ mỗi giờ lúc 17 phút trước giờ
`->everyTwoHours();`	Chạy tác vụ hai giờ một lần
`->everyThreeHours();`	Chạy tác vụ ba giờ một lần
`->everyFourHours();`	Chạy tác vụ bốn giờ một lần
`->everySixHours();`	Chạy tác vụ sáu giờ một lần
`->daily();`	Chạy tác vụ hàng ngày vào lúc nửa đêm
`->dailyAt('13:00');`	Chạy tác vụ hàng ngày lúc 13:00
`->twiceDaily(1, 13);`	Chạy tác vụ hàng ngày lúc 1:00 & 13:00
`->weekly();`	Chạy tác vụ vào Chủ nhật hàng tuần lúc 00:00
`->weeklyOn(1, '8:00');`	Chạy tác vụ hàng tuần vào Thứ Hai lúc 8:00
`->monthly();`	Chạy tác vụ vào ngày đầu tiên hàng tháng lúc 00:00

Phương thức	Mô tả
`->monthlyOn(4, '15:00');`	Chạy tác vụ hàng tháng vào ngày 4 lúc 15:00
`->twiceMonthly(1, 16, '13:00');`	Chạy tác vụ hàng tháng vào ngày 1 và 16 lúc 13:00
`->lastDayOfMonth('15:00');`	Chạy tác vụ vào ngày cuối cùng của tháng lúc 15:00
`->quarterly();`	Chạy tác vụ vào ngày đầu tiên của mỗi quý lúc 00:00
`->yearly();`	Chạy tác vụ vào ngày đầu tiên hàng năm lúc 00:00
`->yearlyOn(6, 1, '17:00');`	Chạy tác vụ hàng năm vào ngày 1 tháng 6 lúc 17:00
`->timezone('America/New_York');`	Đặt múi giờ cho tác vụ

Các phương thức trên có thể được kết hợp với các ràng buộc bổ sung để tạo ra các lịch biểu được điều chỉnh tinh vi hơn chỉ chạy vào một số ngày nhất định trong tuần. Ví dụ: bạn có thể lên lịch để chạy lệnh hàng tuần vào Thứ Hai:

```
// Run once per week on Monday at 1 PM...
$schedule->call(function () {
    //
})->weekly()->mondays()->at('13:00');

// Run hourly from 8 AM to 5 PM on weekdays...
$schedule->command('foo')
  ->weekdays()
  ->hourly()
  ->timezone('America/Chicago')
  ->between('8:00', '17:00');
```

Dưới đây là danh sách các ràng buộc bổ sung cho lịch trình:

Phương thức	Mô tả

Phương thức	Mô tả
`->weekdays();`	Giới hạn tác vụ trong các ngày của tuần
`->weekends();`	Giới hạn tác vụ vào cuối tuần
`->sundays();`	Giới hạn tác vụ đến Chủ nhật
`->mondays();`	Giới hạn tác vụ đến Thứ Hai
`->tuesdays();`	Giới hạn tác vụ đến Thứ Ba
`->wednesdays();`	Giới hạn tác vụ đến Thứ Tư
`->thursdays();`	Giới hạn tác vụ đến Thứ Năm
`->fridays();`	Giới hạn tác vụ đến Thứ Sáu
`->saturdays();`	Giới hạn tác vụ đến Thứ Bảy
`->days(array|mixed);`	Giới hạn tác vụ trong những ngày cụ thể
`->between($startTime, $endTime);`	Giới hạn tác vụ phải chạy giữa thời gian bắt đầu và kết thúc
`->unlessBetween($startTime, $endTime);`	Giới hạn tác vụ để không chạy giữa thời gian bắt đầu và kết thúc
`->when(Closure);`	Giới hạn tác vụ dựa trên một bài kiểm tra xác thực
`->environments($env);`	Giới hạn tác vụ trong các môi trường cụ thể

Ràng buộc trong ngày

Phương pháp `days` có thể được sử dụng để giới hạn việc thực hiện một tác vụ trong những ngày cụ thể trong tuần. Ví dụ: bạn có thể lên lịch để lệnh chạy hàng giờ vào Chủ Nhật và Thứ Tư:

```
$schedule->command('emails:send')
```

```
    ->hourly()
    ->days([0, 3]);
```

Ngoài ra, bạn có thể sử dụng các hằng số có sẵn trên class `Illuminate\Console\Scheduling\Schedule` khi xác định các ngày mà một tác vụ sẽ chạy:

```
use Illuminate\Console\Scheduling\Schedule;

$schedule->command('emails:send')
    ->hourly()
    ->days([Schedule::SUNDAY, Schedule::WEDNESDAY]);
```

Ràng buộc thời gian

Phương thức `between` có thể được sử dụng để giới hạn việc thực thi một tác vụ dựa trên thời gian trong ngày:

```
$schedule->command('emails:send')
    ->hourly()
    ->between('7:00', '22:00');
```

Tương tự như vậy, phương thức `ifBetween` có thể được sử dụng để loại trừ việc thực thi một tác vụ trong một khoảng thời gian xác định:

```
$schedule->command('emails:send')
    ->hourly()
    ->unlessBetween('23:00', '4:00');
```

Ràng buộc trong kiểm tra xác thực

Phương thức `when` có thể được sử dụng để giới hạn việc thực thi một tác vụ dựa trên kết quả của một hàm test đã cho. Nói cách khác, nếu hàm test đã cho trả về true, tác vụ sẽ thực thi miễn là không có thêm điều kiện ràng buộc nào khác ngăn tác vụ này chạy tiếp:

```
$schedule->command('emails:send')->daily()->when(function () {
```

```
    return true;
});
```

Phương thức `skip` có thể được coi là phương thức nghịch đảo của `when`. Nếu phương thức skip trả về `true`, thì tác vụ đã lên lịch sẽ không được thực thi:

```
$schedule->command('emails:send')->daily()->skip(function () {
    return true;
});
```

Khi sử dụng phương thức ràng buột `when`, thì lệnh đã lên lịch sẽ chỉ thực thi nếu tất cả khi các điều kiện trả về `true`.

Các ràng buộc về môi trường

Phương thức enviroments chỉ có thể được sử dụng để thực thi các tác vụ trên các môi trường nào đó đã cho (như được định nghĩa bởi biến môi trường `APP_ENV`):

```
$schedule->command('emails:send')
    ->daily()
    ->environments(['staging', 'production']);
```

Múi giờ

Bằng cách sử dụng phương thức timezone, bạn có thể chỉ định thời gian của tác vụ đã lên lịch sẽ được diễn giải trong một múi giờ nào đó nhất định:

```
$schedule->command('report:generate')
    ->timezone('America/New_York')
    ->at('2:00')
```

Nếu bạn liên tục chỉ định cùng một múi giờ cho tất cả các tác vụ đã lên lịch của mình, bạn có thể vận dụng phương thức `scheduleTimezone` trong lớp `App\Console\Kernel` của mình. Phương thức này sẽ trả về múi giờ mặc định sẽ được chỉ định cho tất cả các tác vụ đã lên lịch:

```php
/**
 * Get the timezone that should be used by default for scheduled events.
 *
 * @return \DateTimeZone|string|null
 */
protected function scheduleTimezone()
{
  return 'America/Chicago';
}
```

Chú ý: Hãy nhớ rằng một số múi giờ vận dụng thời gian tiết kiệm ánh sáng ban ngày. Khi các thay đổi về thời gian tiết kiệm ánh sáng ban ngày xảy ra, tác vụ đã lên lịch của bạn sẽ có thể chạy hai lần hoặc thậm chí hoàn toàn không chạy. Vì lý do này, chúng tôi khuyên bạn nên tránh lập lịch múi giờ khi có thể.

Ngăn chặn việc chồng chéo tác vụ

Theo mặc định, các tác vụ đã lên lịch sẽ được chạy ngay cả khi phiên bản trước của tác vụ vẫn đang chạy. Để ngăn chặn điều này, bạn có thể sử dụng phương thức `withoutOverlapping`:

```php
$schedule->command('emails:send')->withoutOverlapping();
```

Trong ví dụ này, lệnh Artisan `email:send` sẽ được chạy mỗi phút nếu nó chưa chạy. Phương thức `withoutOverlapping` đặc biệt hữu ích nếu bạn có các tác vụ thay đổi đáng kể trong thời gian thực thi của chúng, ngăn bạn dự đoán chính xác thời gian thực thi của một tác vụ nào đó.

Nếu cần, bạn có thể chỉ định bao nhiêu phút phải trôi qua trước khi khóa "without overlapping"("không chồng chéo") hết hạn. Theo mặc định, khóa sẽ hết hạn sau 24 giờ:

```php
$schedule->command('emails:send')->withoutOverlapping(10);
```

Chạy tác vụ trên một máy chủ

> **Chú ý:** Để sử dụng tính năng này, ứng dụng của bạn phải đang sử dụng driver bộ nhớ cache database, memcached, dynamicodb hoặc redis làm driver bộ nhớ cache mặc định của ứng dụng của bạn. Ngoài ra, tất cả các máy chủ phải được giao tiếp với cùng một máy chủ bộ đệm trung tâm.

Nếu chương trình lên lịch của ứng dụng của bạn đang chạy trên nhiều máy chủ, bạn có thể giới hạn công việc đã lên lịch chỉ được thực thi trên một máy chủ. Ví dụ: giả sử bạn có một nhiệm vụ đã lên lịch tạo một báo cáo mới vào mỗi tối thứ Sáu. Nếu chương trình lên lịch tác vụ đang chạy trên ba máy chủ, thì tác vụ đã lên lịch sẽ chạy trên cả ba máy chủ và tạo báo cáo ba lần. Không tốt!

Để chỉ ra rằng tác vụ chỉ nên chạy trên một máy chủ, hãy sử dụng phương thức `onOneServer` khi xác định tác vụ đã lên lịch. Máy chủ đầu tiên nhận được tác vụ sẽ bảo mật một khóa nguyên tử đối với công việc để ngăn các máy chủ khác chạy cùng một tác vụ cùng một lúc:

```
$schedule->command('report:generate')
  ->fridays()
  ->at('17:00')
  ->onOneServer();
```

Tác vụ chạy nền

Theo mặc định, nhiều tác vụ được lên lịch cùng lúc sẽ thực hiện tuần tự dựa trên thứ tự chúng được xác định trong phương thức `schedule` của bạn. Nếu bạn có các tác vụ chạy lâu, điều này có thể khiến các tác vụ tiếp theo bắt đầu muộn hơn nhiều so với dự kiến. Nếu bạn muốn chạy các tác vụ trong nền để tất cả chúng có thể chạy đồng thời, bạn có thể sử dụng phương thức `runInBackground`:

```
$schedule->command('analytics:report')
  ->daily()
  ->runInBackground();
```

> **Chú ý:** The `runInBackground` method may only be used when scheduling tasks via the `command` and `exec` methods.

Chế độ bảo trì

Các tác vụ đã lên lịch của ứng dụng của bạn sẽ không chạy khi ứng dụng đang ở chế độ bảo trì, vì chúng tôi không muốn các tác vụ của bạn ảnh hưởng đến bất kỳ công việc bảo trì chưa hoàn thành nào mà bạn có thể đang thực hiện trên máy chủ của mình. Tuy nhiên, nếu bạn muốn buộc một tác vụ chạy ngay cả trong chế độ bảo trì, bạn có thể gọi phương thức `evenInMaintenanceMode` khi xác định tác vụ:

```
$schedule->command('emails:send')->evenInMaintenanceMode();
```

Chạy scheduler

Bây giờ chúng ta đã học cách xác định các tác vụ đã lên lịch, hãy thảo luận về cách thực sự chạy chúng trên máy chủ của chúng ta. Lệnh Artisan theo `schedule:run` sẽ đánh giá tất cả các tác vụ đã lên lịch của bạn và xác định xem chúng có cần chạy hay không dựa trên thời gian hiện tại của máy chủ.

Vì vậy, khi sử dụng scheduler của Laravel, chúng ta chỉ cần thêm một mục nhập cấu hình cron duy nhất vào máy chủ của chúng ta để chạy lệnh schedule:run mỗi phút. Nếu bạn không biết cách thêm các mục cron vào máy chủ của mình, hãy xem xét sử dụng một dịch vụ như Laravel Forge có thể quản lý các mục cron cho bạn:

```
* * * * * cd /path-to-your-project && php artisan schedule:run >> /dev/null 2>&1
```

Chạy scheduler cục bộ

Thông thường, bạn sẽ không thêm mục nhập cron của scheduler vào máy phát triển cục bộ của mình. Thay vào đó, bạn có thể sử dụng lệnh artisan `schedule:work`. Lệnh này sẽ chạy ở chế độ foreground và gọi scheduler mỗi phút cho đến khi bạn kết thúc lệnh:

```
php artisan schedule:work
```

Đầu ra nhiệm vụ

Scheduler của Laravel cho một số phương thức thuận tiện để làm việc với đầu ra được tạo ra bởi các tác vụ đã lên lịch. Đầu tiên, bằng cách sử dụng phương thức `sendOutputTo`, bạn có thể gửi đầu ra tới một tập tin để kiểm tra như sau:

```
$schedule->command('emails:send')
  ->daily()
  ->sendOutputTo($filePath);
```

Nếu bạn muốn nối kết quả đầu ra vào một tập tin nhất định, bạn có thể sử dụng phương thức `appendOutputTo`:

```
$schedule->command('emails:send')
  ->daily()
  ->appendOutputTo($filePath);
```

Sử dụng phương thức `emailOutputTo`, bạn có thể gửi email đầu ra đến một địa chỉ email bạn chọn. Trước khi gửi email đầu ra của một nhiệm vụ, bạn nên cấu hình các dịch vụ email của Laravel:

```
$schedule->command('report:generate')
  ->daily()
  ->sendOutputTo($filePath)
  ->emailOutputTo('taylor@example.com');
```

Nếu bạn chỉ muốn gửi email đầu ra nếu lệnh Artisan hoặc hệ thống đã lên lịch kết thúc bằng mã thoát khác 0, hãy sử dụng phương thức `emailOutputOnFailure`:

```
$schedule->command('report:generate')
  ->daily()
  ->emailOutputOnFailure('taylor@example.com');
```

> **Chú ý:** Các phương thức `emailOutputTo`, `emailOutputOnFailure`, `sendOutputTo` và `appendOutputTo` chỉ dành riêng cho các phương thức command và exec.

Hook tác vụ

Sử dụng các phương thức trước và sau, bạn có thể chỉ định mã được thực thi trước và sau khi tác vụ đã lên lịch được thực thi:

```php
$schedule->command('emails:send')
  ->daily()
  ->before(function () {
    // The task is about to execute...
  })
  ->after(function () {
    // The task has executed...
  });
```

Các phương thức `onSuccess` và `onFailure` cho phép bạn chỉ định mã sẽ được thực thi nếu tác vụ đã lên lịch thành công hay không thành công. Lỗi cho biết rằng lệnh Artisan hoặc hệ thống đã lên lịch đã kết thúc bằng mã thoát khác 0:

```php
$schedule->command('emails:send')
  ->daily()
  ->onSuccess(function () {
    // The task succeeded...
  })
  ->onFailure(function () {
    // The task failed...
  });
```

Nếu đầu ra có sẵn từ lệnh của bạn, bạn có thể truy cập nó trong các hook `after`, `onSuccess` hoặc `onFailure` của mình bằng cách nhập gợi ý đối tượng `Illuminate\Support\Stringable` làm đối số `$output` của định nghĩa đóng hook của bạn:

```php
use Illuminate\Support\Stringable;

$schedule->command('emails:send')
  ->daily()
  ->onSuccess(function (Stringable $output) {
    // The task succeeded...
  })
  ->onFailure(function (Stringable $output) {
    // The task failed...
```

```
});
```

Ping URL

Bằng cách sử dụng các phương thức pingBefore và thenPing, scheduler có thể tự động ping một URL nào đó trước hoặc sau khi một tác vụ được thực thi. Phương pháp này hữu ích khi thông báo cho một dịch vụ bên ngoài, chẳng hạn như Envoyer, rằng tác vụ đã lên lịch của bạn đang bắt đầu hoặc đã kết thúc thực thi:

```
$schedule->command('emails:send')
    ->daily()
    ->pingBefore($url)
    ->thenPing($url);
```

Các phương thức `pingBeforeIf` và `thenPingIf` chỉ có thể được sử dụng để ping một URL nhất định nếu một điều kiện nhất định là `true`:

```
$schedule->command('emails:send')
    ->daily()
    ->pingBeforeIf($condition, $url)
    ->thenPingIf($condition, $url);
```

Các phương thức `pingOnSuccess` và `pingOnFailure` chỉ có thể được sử dụng để ping một URL nhất định khi tác vụ thành công hoặc không thành công. Lỗi cho biết rằng lệnh Artisan hoặc hệ thống đã lên lịch đã kết thúc bằng mã thoát khác 0:

```
$schedule->command('emails:send')
    ->daily()
    ->pingOnSuccess($successUrl)
    ->pingOnFailure($failureUrl);
```

Tất cả các phương thức ping đều yêu cầu thư viện Guzzle HTTP. Guzzle thường được cài đặt trong tất cả các dự án Laravel mới theo mặc định, nhưng bạn có thể cài đặt Guzzle vào dự án của mình theo cách thủ công bằng trình quản lý gói Composer nếu nó vô tình bị gỡ bỏ:

```
composer require guzzlehttp/guzzle
```

Event (Sự kiện)

Nếu cần, bạn có thể lắng nghe các sự kiện do người lập lịch cử đi. Thông thường, ánh xạ trình xử lý sự kiện sẽ được xác định trong lớp `App\Providers\EventServiceProvider` của ứng dụng của bạn:

```
/**
 * The event listener mappings for the application.
 *
 * @var array
 */
protected $listen = [
  'Illuminate\Console\Events\ScheduledTaskStarting' => [
    'App\Listeners\LogScheduledTaskStarting',
  ],

  'Illuminate\Console\Events\ScheduledTaskFinished' => [
    'App\Listeners\LogScheduledTaskFinished',
  ],

  'Illuminate\Console\Events\ScheduledBackgroundTaskFinished' => [
    'App\Listeners\LogScheduledBackgroundTaskFinished',
  ],

  'Illuminate\Console\Events\ScheduledTaskSkipped' => [
    'App\Listeners\LogScheduledTaskSkipped',
  ],

  'Illuminate\Console\Events\ScheduledTaskFailed' => [
    'App\Listeners\LogScheduledTaskFailed',
  ],
];
```