Course - Laravel Framework

Ghi chép logging

Để giúp bạn biết thêm được những gì đang xảy ra trong ứng dụng của mình, Laravel cung cấp các service ghi chép logging cho phép bạn ghi chép lại các thông điệp từ lỗi hệ thống và thậm chí cả Slack để thuận tiện thông báo cho đội nhóm của bạn.

Tags: logging, laravel

Ghi logging của Laravel dựa trên các "channel". Mỗi channel thể hiện một cách cụ thể việc ghi chép cho một dạng thông điệp nào đó. Ví dụ: channel `single` sẽ ghi chép vào một tập tin duy nhất, trong khi channel `slack` gửi thông điệp log đến Slack. Thông điệp log có thể được ghi vào nhiều channel dựa trên mức độ nghiêm trọng của chúng.

Ở phía sau, Laravel sử dụng thư viện `Monolog` vì Laravel muốn giúp bạn dễ dàng cấu hình các chương trình xử lý log này, nó cho phép bạn trộn và kết hợp chúng để tùy chỉnh việc xử lý log trong ứng dụng của bạn.

Cấu hình

Tất cả các cài đặt cấu hình cho chức năng ghi log của ứng dụng của bạn được đặt trong tập tin `config/logging.php`. Tập tin này cho phép bạn cài đặt channel cho việc ghi log của ứng dụng, vì vậy hãy đảm bảo rằng bạn đã đọc kỹ từng channel đang có và các tùy chọn của chúng. Bây giờ, chúng ta sẽ xem qua một số cài đặt phổ biến bên dưới.

Mặc định, Laravel sẽ sử dụng channel `stack` khi ghi các thông điệp log. `stack` được sử dụng để tổng hợp nhiều channel ghi log thành một channel duy nhất. Để biết thêm thông tin về cách xây dựng `stack`, hãy xem phần tài liệu có ở bên dưới.

Cài đặt tên channel

Mặc định, `Monolog` sẽ được khởi tạo với một tên channel phù hợp với môi trường hiện tại, chẳng hạn như `production` hoặc `local`. Để thay đổi giá trị này, hãy thêm tùy chọn `name` vào cài đặt channel của bạn:

```
'stack' => [
  'driver' => 'stack',
  'name' => 'channel-name',
  'channels' => ['single', 'slack'],
],
```

Các driver channel

Mỗi channel log được cung cấp bởi một "driver". Driver xác định cách thức và vị trí thông điệp log thực sự được ghi chép lại. Các driver channel ghi chép sau đây có sẵn trong mọi ứng dụng Laravel. Một mục giành cho tất cả các driver này có trong tập tin *config/logging.php* của ứng dụng của bạn, vì vậy hãy nhớ mở và đọc tập tin này để làm quen

với nội dung của nó:

Tên	Mô tả
`custom`	Driver sẽ gọi một factory cụ thể để tạo channel
`daily`	Driver Monolog đi với `RotatingFileHandler` sẽ hoạt động hàng ngày
`errorlog`	Driver Monolog đi với `ErrorLogHandler`
`monolog`	Driver Monolog có thể sử dụng bất kỳ chương trình xử lý Monolog nào được hỗ trợ
`null`	Driver sẽ loại bỏ tất cả các thông điệp ghi chép
`papertrail`	Driver Monolog đi với `SyslogUdpHandler`
`single`	Một tập tin hoặc một channel ghi nhật ký dựa trên đường dẫn (`StreamHandler`)
`slack`	Driver Monolog đi với `SlackWebhookHandler`
`stack`	Một tủ học để thuận lợi cho việc tạo các channel "multi-channel"
`syslog`	Driver Monolog với SyslogHandler

Bảng tuỳ chọn cho cài đặt channel

> Xem tài liệu về cài đặt channel nâng cao để tìm hiểu thêm về driver monolog và driver tự tạo.

Yêu cầu ban đầu của channel

Cài đặt channel single và daily

Các channel `single` và `daily` có ba tùy chọn cài đặt: `bubble`, `permission`, và `locking`.

Tên	Mô tả	Mặc định
`bubble`	Cho biết liệu thông điệp có nổi lên các channel khác sau khi được xử lý hay không	`true`
`locking`	Cố gắng khóa tập tin ghi chép log trước khi ghi log vào nó	`false`
`permission`	Quyền của tập tin ghi chép log	`0644`

Cài đặt channel `papertrail`

Kênh `papertrail` yêu cầu các tuỳ chọn host và port. Bạn có thể lấy các giá trị này từ Papertrail.

Cài đặt channel slack

Channel Slack yêu cầu tùy chọn cài đặt `url`. URL này phải khớp với một URL cho webhook được gửi vào mà bạn đã cài đặt cho nhóm Slack của mình. Mặc định, Slack sẽ chỉ nhận nhật ký ở mức quan trọng (`critical`) trở lên; tuy nhiên, bạn có thể điều chỉnh lại trong tập tin cấu hình *config/logging.php* của mình bằng cách sửa đổi tùy chọn cài đặt `level` trong mảng cấu hình channel ghi log qua Slack của bạn.

Ghi log cảnh báo ngừng

PHP, Laravel và các thư viện khác thường thông báo cho người dùng của họ rằng một số tính năng của họ đã không còn được dùng nữa và sẽ bị loại bỏ trong một phiên bản trong tương lai. Nếu bạn muốn ghi lại các cảnh báo ngưng sử dụng này, bạn có thể chỉ định channel ghi log các cảnh báo ngưng sử dụng theo sự ưu tiên của riêng mình trong tập tin *config/logging.php* trong dự án:

```
'deprecations' => env('LOG_DEPRECATIONS_CHANNEL', 'null'),

'channels' => [
    ...
]
```

Hoặc, bạn có thể cài đặt channel ghi log có tên là `deprecations`. Nếu tồn tại một channel ghi log với tên này, nó sẽ luôn được sử dụng để ghi log cảnh báo ngưng sử dụng:

```
'channels' => [
  'deprecations' => [
    'driver' => 'single',
    'path' => storage_path('logs/php-deprecation-warnings.log'),
  ],
],
```

Xây dựng stack ghi log

Như đã đề cập trước đây, driver `stack` cho phép bạn kết hợp nhiều channel thành một channel ghi log duy nhất. Để minh họa cách sử dụng `stack`, hãy xem cài đặt ví dụ mà bạn có thể tìm thấy trong một ứng dụng production:

```
'channels' => [
  'stack' => [
    'driver' => 'stack',
    'channels' => ['syslog', 'slack'],
  ],
  'syslog' => [
    'driver' => 'syslog',
    'level' => 'debug',
  ],
  'slack' => [
    'driver' => 'slack',
    'url' => env('LOG_SLACK_WEBHOOK_URL'),
    'username' => 'Laravel Log',
    'emoji' => ':boom:',
    'level' => 'critical',
  ],
],
```

Hãy cùng mổ xẻ cấu hình này. Đầu tiên, hãy lưu ý rằng `stack` của chúng ta liên kết hai channel khác nhau lại với nhau qua tùy chọn `channels` của nó: `syslog` và `slack`. Vì vậy, khi ghi log các thông điệp, cả hai channel này sẽ có cơ hội ghi lại các thông điệp đó. Tuy nhiên, như chúng ta sẽ thấy được ở bên dưới, liệu các kênh này có thực sự ghi lại thông điệp hay không, việc này có thể được xác định bởi mức độ nghiêm trọng hay mức độ nào khác của thông điệp.

Các mức độ ghi log

Hãy để ý đến tùy chọn cài đặt `level` có trên các cài đặt `syslog` và `slack` trong ví dụ trên. Tùy chọn này xác định mức độ tối thiểu nhất mà một thông điệp phải có để được channel ghi chép lại. `Monolog`, một chương trình hỗ trợ các service ghi log của Laravel, đã đưa ra tất cả các cấp độ ghi log được khai báo trong bảng mô tả kỹ thuật RFC số 5424: khẩn cấp (`emergency`), báo động (`alert`), quan trọng (`critical`), lỗi (`error`), cảnh báo (`warning`), thông báo (`notice`), thông tin (`info`) và gỡ lỗi (`debug`).

Vì vậy, hãy tưởng tượng chúng ta ghi chép một thông điệp bằng phương thức `debug`:

```
Log::debug('An informational message.');
```

Với cài đặt mà chúng ta đã đưa ra, channel ghi chép `syslog` sẽ ghi thông điệp vào phần ghi chép cho hệ thống; tuy nhiên, vì thông điệp về lỗi không phải là nghiêm trọng (`critical`) hoặc cao hơn, nên nó sẽ không được gửi đến dịch vụ Slack. Tuy nhiên, nếu chúng ghi chép một thông điệp khẩn cấp (`emergency`), nó sẽ được gửi đến cả syslog và dịch vụ Slack, đó là vì mức độ khẩn cấp cao hơn ngưỡng cấp độ tối thiểu mà chúng ta đã cài đặt vào cả hai channel:

```
Log::emergency('The system is down!');
```

Ghi chép các thông điệp

Bạn có thể ghi chép các thông điệp bằng cách sử dụng facade `Log`. Như đã đề cập trước đây, chương trình ghi chép cung cấp tám mức độ ghi chép được khai báo trong bảng mô tả kỹ thuật RFC số 5424: khẩn cấp (`emergency`), báo động (`alert`), quan trọng (`critical`), lỗi (`error`), cảnh báo (`warning`), thông báo (`notice`), thông tin (`info`) và gỡ lỗi (`debug`):

```
use Illuminate\Support\Facades\Log;

Log::emergency($message);
Log::alert($message);
Log::critical($message);
Log::error($message);
Log::warning($message);
Log::notice($message);
Log::info($message);
Log::debug($message);
```

Bạn có thể gọi bất kỳ phương thức nào trong số này để ghi lại một thông điệp cho cấp độ tương ứng. Mặc định, thông điệp sẽ được ghi chép vào channel ghi chép được thiết lập sẵn trước đó trong tập tin cài đặt `logging` của bạn:

```php
<?php
namespace App\Http\Controllers;
use App\Http\Controllers\Controller;
use App\Models\User;
use Illuminate\Support\Facades\Log;
class UserController extends Controller
{
  /**
   * Show the profile for the given user.
   *
   * @param  int  $id
   * @return \Illuminate\Http\Response
   */
  public function show($id)
  {
    Log::info('Showing the user profile for user: '.$id);
    return view('user.profile', [
      'user' => User::findOrFail($id)
    ]);
  }
}
```

Thông tin cụ thể

Một mảng dữ liệu mô tả bối cảnh nào đó có thể được truyền vào các phương thức ghi chép. Dữ liệu mô tả này sẽ được định dạng và hiển thị với thông điệp ghi chép:

```php
use Illuminate\Support\Facades\Log;

Log::info('User failed to login.', ['id' => $user->id]);
```

Đôi khi, bạn có thể muốn chỉ định một số thông tin bối cảnh cụ thể cần được đưa vào tất cả các mục ghi chép tiếp theo. Ví dụ: bạn có thể muốn ghi lại ID đã được gắn với từng request gửi đến ứng dụng của bạn. Để thực hiện điều này, bạn có thể gọi phương thức `withContext` của facade `Log`:

```php
<?php
namespace App\Http\Middleware;
use Closure;
use Illuminate\Support\Facades\Log;
use Illuminate\Support\Str;
class AssignRequestId
{
  /**
   * Handle an incoming request.
   *
   * @param  \Illuminate\Http\Request  $request
   * @param  \Closure  $next
   * @return mixed
   */
  public function handle($request, Closure $next)
  {
    $requestId = (string) Str::uuid();
    Log::withContext([
      'request-id' => $requestId
    ]);
    return $next($request)->header('Request-Id', $requestId);
  }
}
```

Ghi vào channel chỉ định

Nhiều khi bạn có thể muốn ghi lại một thông điệp vào một channel khác với channel mặc định của ứng dụng của bạn. Bạn có thể sử dụng phương thức `channel` trên facade `Log` để truy xuất và ghi chép vào bất kỳ channel nào được chỉ định trong tập tin cài đặt của bạn:

```php
use Illuminate\Support\Facades\Log;

Log::channel('slack')->info('Something happened!');
```

Nếu bạn muốn tạo một stack ghi chép theo yêu cầu có chứa nhiều channel, bạn có thể sử dụng phương thức `stack`:

```
Log::stack(['single', 'slack'])->info('Something happened!');
```

Các channel yêu cầu

Cũng có thể tạo channel theo yêu cầu bằng cách cung cấp cài đặt trong thời gian chạy mà không cần cài đặt đó có trong tập tin cài đặt ghi chép của ứng dụng của bạn. Để thực hiện điều này, bạn có thể truyền một mảng cấu hình cho phương thức `build` của facade `Log`:

```
use Illuminate\Support\Facades\Log;

Log::build([
  'driver' => 'single',
  'path' => storage_path('logs/custom.log'),
])->info('Something happened!');
```

Bạn cũng có thể muốn đưa channel theo yêu cầu vào stack ghi chép. Điều này có thể được thực hiện bằng cách bố trí đối tượng channel theo yêu cầu của bạn vào trong mảng được truyền cho phương thức `stack`:

```
use Illuminate\Support\Facades\Log;

$channel = Log::build([
  'driver' => 'single',
  'path' => storage_path('logs/custom.log'),
]);

Log::stack(['slack', $channel])->info('Something happened!');
```

Monolog channel

Dựng lại Monolog

Đôi khi bạn có thể cần kiểm soát hoàn toàn cách cài đặt `Monolog` cho một channel hiện có. Ví dụ: bạn có thể muốn chỉ định cài đặt một bản thực thi interface `FormatterInterface` của `Monolog` theo ý mình cho channel `single` được tích hợp sẵn trong Laravel.

Để bắt đầu, hãy xác định mảng `tap` trên cài đặt của channel. Mảng `tap` chứa một danh sách các class có cơ hội tùy chỉnh (hoặc "tap" vào) đối tượng `Monolog` sau khi nó được tạo. Không có vị trí thuận lợi nào để đặt các class này, vì vậy bạn có thể tự tạo một thư mục bên trong dự án của mình để chứa các class này:

```
'single' => [
  'driver' => 'single',
  'tap' => [App\Logging\CustomizeFormatter::class],
  'path' => storage_path('logs/laravel.log'),
  'level' => 'debug',
],
```

Khi bạn đã chỉ định cài đặt `tap` trên channel của mình, bạn đã sẵn sàng khai báo class sẽ dựng lại `Monolog` của bạn. Class này chỉ cần một phương thức duy nhất: `__invoke`, phương thức này nhận một đối tượng `Illuminate\Log\Logger`. Đối tượng `Illuminate\Log\Logger` sẽ ủy quyền cho tất cả các lệnh gọi phương thức đến đối tượng `Monolog` bên dưới:

```
<?php
namespace App\Logging;
use Monolog\Formatter\LineFormatter;
class CustomizeFormatter
{
  /**
   * Customize the given logger instance.
   *
   * @param  \Illuminate\Log\Logger  $logger
   * @return void
   */
  public function __invoke($logger)
  {
    foreach ($logger->getHandlers() as $handler) {
      $handler->setFormatter(new LineFormatter(
        '[%datetime%] %channel%.%level_name%: %message% %context% %extra%'
      ));
    }
  }
```

```
}
```

> Tất cả các class trong "`tap`" của bạn được nhận được bởi service container, vì vậy mọi khai báo thêm thư viện trên contructor sẽ tự động được đưa vào.

Tạo channel xử lý Monolog

`Monolog` có nhiều chương trình xử lý có sẵn và Laravel không sẽ đưa vào các channel có sẵn cho từng chương trình xử lý. Trong một số trường hợp, bạn có thể muốn tạo một channel chỉ đơn thuần là một phiên bản của chương trình xử lý Monolog nào đó mà không hề có driver ghi chép Laravel tương ứng. Có thể dễ dàng tạo các channel này bằng cách dùng driver `monolog`.

Khi sử dụng driver `monolog`, tùy chọn cài đặt `handler` được sử dụng để chỉ định chương trình xử lý nào sẽ được khởi tạo. Theo ý bạn, bất kỳ tham số constructor nào mà chương trình xử lý cần đều có thể được chỉ định bằng sử dụng tùy chọn cài đặt `with`:

```
'logentries' => [
  'driver'  => 'monolog',
  'handler' => Monolog\Handler\SyslogUdpHandler::class,
  'with' => [
    'host' => 'my.logentries.internal.datahubhost.company.com',
    'port' => '10000',
  ],
],
```

Công thức monolog

Khi sử dụng driver monolog, Monolog `LineFormatter` sẽ được sử dụng làm công thức mặc định. Tuy nhiên, bạn có thể tùy chỉnh loại công thức khi truyền vào chương trình xử lý bằng cách sử dụng tùy chọn cài đặt `formatter` và `formatter_with`:

```
'browser' => [
  'driver' => 'monolog',
  'handler' => Monolog\Handler\BrowserConsoleHandler::class,
  'formatter' => Monolog\Formatter\HtmlFormatter::class,
```

```
    'formatter_with' => [
      'dateFormat' => 'Y-m-d',
    ],
],
```

Nếu bạn đang sử dụng chương trình xử lý Monolog có khả năng cung cấp công thức của riêng nó, bạn có thể đặt giá trị của tùy chọn cài dặt công thức thành mặc định:

```
'newrelic' => [
    'driver' => 'monolog',
    'handler' => Monolog\Handler\NewRelicHandler::class,
    'formatter' => 'default',
],
```

Tạo channel tùy chọn với factory

Nếu bạn muốn xác định một channel hoàn toàn theo cách của mình trong đó bạn có toàn quyền kiểm soát việc khởi tạo và cài đặt Monolog, bạn có thể chỉ định loại `custom` driver bạn muốn trong tập tin cài đặt *config/logging.php* của mình. Cài đặt của bạn nên bao gồm một tùy chọn `via` có chứa tên của class mà sẽ được gọi để tạo đối tượng Monolog:

```
'channels' => [
    'example-custom-channel' => [
        'driver' => 'custom',
        'via' => App\Logging\CreateCustomLogger::class,
    ],
],
```

Khi bạn đã định cấu hình `custom` driver cho channel, thì bạn đã sẵn sàng khai báo class mà sẽ tạo đối tượng Monolog của bạn. Class này chỉ cần một phương thức `__invoke` duy nhất và nó sẽ trả về một đối tượng Monolog xử lý ghi chép log. Phương thức sẽ nhận mảng cấu hình channel làm đối số duy nhất của nó:

```
<?php
namespace App\Logging;
use Monolog\Logger;
```

```php
class CreateCustomLogger
{
  /**
   * Create a custom Monolog instance.
   *
   * @param  array  $config
   * @return \Monolog\Logger
   */
  public function __invoke(array $config)
  {
    return new Logger(...);
  }
}
```