Course - Laravel Framework

Session

Vì các ứng dụng vận hành trên HTTP là không trạng thái, nên các session được đưa ra để lưu trữ thông tin về người dùng cho các lượt request sau đó. Thông tin người dùng đó thường được đặt trong một bộ lưu trữ hoặt phía server có thể truy cập được từ các request tiếp theo sau đó.

Tags: session, laravel

Giới thiệu

Laravel cung cấp nhiều chương trình hỗ trợ session mà sẽ được truy cập thông qua một API thống nhất, rõ ràng. API này hỗ trợ cho các chương trình phổ biến như Memcached, Redis và các database khác.

Cấu hình

Tập tin cấu hình session của ứng dụng của bạn được lưu trữ tại *config/session.php*. Hãy đảm bảo rằng bạn review các cài đặt có trong tập tin này. Mặc định, Laravel được cấu hình để sử dụng driver session trên tập tin, driver này sẽ hoạt động tốt cho nhiều ứng dụng. Nếu ứng dụng của bạn sử dụng cân bằng tải trên nhiều máy chủ web, thì bạn nên chọn một bộ lưu trữ tập trung mà tất cả các máy chủ đều có thể truy cập, chẳng hạn như Redis hoặc database.

Tùy chọn cài đặt driver session sẽ chỉ định nơi mà dữ liệu session sẽ được lưu trữ cho mỗi request. Laravel cung cấp thêm một số driver như:

- `file` - session được lưu trữ trong *bộ nhớ/framework/session*.
- `cookie` - session được lưu trữ trong cookie đã được mã hóa, an toàn.
- `database` - session sẽ được lưu trữ trong cơ sở dữ liệu quan hệ.
- `memcached/redis` - session được lưu trữ trong một trong những kho lưu trữ nhanh, dựa trên bộ nhớ cache.
- `dynamicodb` - session được lưu trữ trong AWS DynamoDB.
- `array` - session được lưu trữ trong một mảng PHP và sẽ không tồn tại.

> Driver mảng chủ yếu được sử dụng trong quá trình test và ngăn không cho dữ liệu đã lưu trữ trong session được tồn tại.

Các yêu cầu đối với driver

Database

Khi sử dụng session driver với database, bạn sẽ cần tạo một bảng để chứa các ghi chép của session. Bạn có thể tìm thấy một khai báo schema mẫu trong bảng dưới đây:

```
Schema::create('sessions', function ($table) {
  $table->string('id')->primary();
  $table->foreignId('user_id')->nullable()->index();
```

```
    $table->string('ip_address', 45)->nullable();
    $table->text('user_agent')->nullable();
    $table->text('payload');
    $table->integer('last_activity')->index();
});
```

Bạn có thể sử dụng lệnh Artisan `session:table` để tạo quá trình chuyển đổi này. Để tìm hiểu thêm về di chuyển cơ sở dữ liệu, bạn có thể tham khảo tài liệu di chuyển đầy đủ:

```
php artisan session:table
```

```
php artisan migrate
```

Redis

Trước khi sử dụng session Redis với Laravel, bạn cần cài đặt phần mở rộng *PhpRedis PHP* với *PECL* hoặc cài đặt package *predis/predis* (~ 1.0) với Composer. Để biết thêm thông tin về cách cấu hình Redis, hãy tham khảo tài liệu Redis của Laravel.

Tương tác với Session

Truy xuất dữ liệu

Có hai cách chính để làm việc với dữ liệu session trong Laravel: hàm `session` toàn cục và đối tượng `Request`. Đầu tiên, chúng ta hãy xem xét việc truy cập session thông qua một đối tượng `Request`, có thể khai kiểu trên hàm nặc danh của route hoặc controller. Hãy nhớ rằng, các thư viện được tự động đưa vào với service container của Laravel:

```
<?php
namespace App\Http\Controllers;
use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
class UserController extends Controller
{
  /**
   * Show the profile for the given user.
   *
```

```
     * @param  Request  $request
     * @param  int  $id
     * @return Response
     */
    public function show(Request $request, $id)
    {
        $value = $request->session()->get('key');

        //
    }
}
```

Khi bạn truy xuất một mục nào đó từ session, bạn cũng có thể truyền một giá trị mặc định làm đối số thứ hai cho phương thức `get`. Giá trị mặc định này sẽ được trả về nếu key được chỉ định không tồn tại trong session. Nếu bạn truyền một hàm nặc danh làm giá trị mặc định cho phương thức `get` và key được yêu cầu không tồn tại, thì hàm này sẽ được thực thi và kết quả của nó sẽ được trả về:

```
$value = $request->session()->get('key', 'default');

$value = $request->session()->get('key', function () {
    return 'default';
});
```

Hàm session

Bạn cũng có thể sử dụng hàm `session` của PHP để truy xuất và lưu trữ dữ liệu trong session. Khi hàm `session` được gọi với một đối số chuỗi duy nhất nó sẽ trả về giá trị của key của session đó. Khi hàm được gọi với một mảng chứa các cặp khóa/giá trị, các giá trị đó sẽ được lưu trữ trong session:

```
Route::get('/home', function () {
    // Retrieve a piece of data from the session...
    $value = session('key');

    // Specifying a default value...
```

```
    $value = session('key', 'default');

    // Store a piece of data in the session...
    session(['key' => 'value']);
});
```

> Có một chút khác biệt thực sự giữa việc sử dụng session với một đối tượng HTTP request và việc sử dụng hàm `session`. Cả hai phương pháp đều có thể kiểm tra được bằng phương thức `assertSessionHas` có trong tất cả các test case của bạn.

Truy xuất tất cả dữ liệu session

Nếu bạn muốn truy xuất tất cả dữ liệu trong sesion, thì bạn có thể sử dụng phương thức `all`:

```
$data = $request->session()->all();
```

Kiểm tra mục nào đó trong session

Để kiểm tra xem một mục nào đó có được lưu trong session hay không, thì bạn có thể sử dụng phương thức `has`. Phương thức `has` trả về `true` nếu mục đang kiểm tra đã được lưu và khi không có giá trị thì trả về `null`:

```
if ($request->session()->has('users')) {
    //
}
```

Để kiểm tra xem một mục nào đó có tồn tại trong session hay không, ngay cả khi giá trị của nó là `null`, thì bạn có thể sử dụng phương thức `esixts`:

```
if ($request->session()->exists('users')) {
    //
}
```

Để kiểm tra xem một mục nào đó không có trong session, thì bạn có thể sử dụng phương

thức `missing`. Phương thức `missing` trả về `true` nếu mục là `null` hoặc nếu mục không được lưu:

```
if ($request->session()->missing('users')) {
  //
}
```

Lưu trữ dữ liệu

Để lưu trữ dữ liệu trong session, bạn thường sẽ sử dụng phương thức `put` của đối tượng `Request` hoặc hàm `session`:

```
// Via a request instance...
$request->session()->put('key', 'value');

// Via the global "session" helper...
session(['key' => 'value']);
```

Lưu vào mảng trong session

Phương thức `push` có thể được sử dụng để đẩy dữ liệu vào một mảng bên trong session. Ví dụ: nếu khóa `user.teams` chứa một mảng tên nhóm, bạn có thể đẩy một giá trị mới vào mảng như sau:

```
$request->session()->push('user.teams', 'developers');
```

Truy xuất và xóa dữ liệu

Phương thức `pull` sẽ truy xuất và xóa một mục nào khỏi session trong một câu lệnh:

```
$value = $request->session()->pull('key', 'default');
```

Tăng và giảm giá trị trong session

Nếu dữ liệu session của bạn chứa một số nguyên mà bạn muốn tăng hoặc giảm, thì bạn có thể sử dụng các phương `increment` và `decrement`:

```
$request->session()->increment('count');
$request->session()->increment('count', $incrementBy = 2);
$request->session()->decrement('count');
$request->session()->decrement('count', $decrementBy = 2);
```

Flash Data

Đôi khi, bạn có thể muốn lưu trữ các mục trong session cho duy nhất một request tiếp theo ngay sau đó. Bạn có thể làm như vậy bằng cách sử dụng phương thức `flash`. Dữ liệu được lưu trữ trong session sử dụng phương thức này sẽ có sẵn ngay lập tức và trong quá trình gửi HTTP request tiếp theo. Sau khi đến được HTTP request tiếp theo, thì dữ liệu đã flash sẽ bị xóa. Dữ liệu flash chủ yếu hữu ích cho các thông điệp status tồn tại trong thời gian ngắn:

```
$request->session()->flash('status', 'Task was successful!');
```

Nếu bạn cần duy trì dữ liệu đã flash của mình vì một số lý do, thì bạn có thể sử dụng phương thức `reflash`, phương thức này sẽ giữ lại tất cả dữ liệu đã flash cho một request bổ sung nào đó. Nếu bạn cần giữ lại dữ liệu flash cần thiết, thì bạn có thể sử dụng phương thức `keep`:

```
$request->session()->reflash();

$request->session()->keep(['username', 'email']);
```

Để duy trì dữ liệu flash của bạn chỉ cho request hiện tại, bạn có thể sử dụng phương pháp `now`:

```
$request->session()->now('status', 'Task was successful!');
```

Xóa dữ liệu

Phương thức `forget` sẽ xóa một mục nào đó ra khỏi session. Nếu bạn muốn xóa mọi dữ liệu khỏi session, bạn có thể sử dụng phương thức `flush`:

```
// Forget a single key...
$request->session()->forget('name');

// Forget multiple keys...
$request->session()->forget(['name', 'status']);

$request->session()->flush();
```

Tạo SessionID

Việc tạo lại session ID thường được thực hiện để ngăn kẻ xâm hại khai thác cuộc tấn công với session cố định trên ứng dụng của bạn.

Laravel tự động tạo lại session ID trong quá trình xác thực nếu bạn đang sử dụng một trong các bộ khởi động ứng dụng Laravel hoặc *Laravel Fortify*; tuy nhiên, nếu bạn cần tạo lại session ID theo cách thủ công, bạn có thể sử dụng phương pháp `regenerate`:

```
$request->session()->regenerate();
```

Nếu bạn cần tạo lại session ID và xóa tất cả dữ liệu khỏi session trong một câu lệnh, bạn có thể sử dụng phương thức `invalidate`:

```
$request->session()->invalidate();
```

Session Blocking

> **Chú ý:** Để sử dụng tính năng Session Blocking, thì ứng dụng của bạn phải sử dụng dirver cache hỗ trợ xử lý tốc độ cao. Hiện tại, những driver cache đó gồm các driver như memcached, dynamicodb, redis và datebase. Ngoài ra, bạn không thể sử dụng cookie driver.

Mặc định, Laravel cho phép các request sử dụng cùng một session thực thi đồng bộ. Ví dụ: nếu bạn sử dụng thư viện JavaScript HTTP để gửi hai HTTP request đến ứng dụng của

mình, thì cả hai sẽ thực thi cùng một lúc. Đối với nhiều ứng dụng, đây không phải là vấn đề; tuy nhiên, mất dữ liệu session có thể xảy ra trong một phần nhỏ các ứng dụng khi thực hiện request đồng bộ đến hai url khác nhau, mà cả hai đều ghi dữ liệu vào session.

Để giảm thiểu điều này, Laravel cung cấp chức năng cho phép bạn giới hạn các request đồng bộ cho một session nào đó. Để bắt đầu, bạn có thể chỉ cần gọi phương thức `block` vào khai báo route của mình. Trong ví dụ này, một yêu cầu đến endpoint `/profile` sẽ có được một ổ khóa session. Trong khi ổ khóa này đang có hiệu lực, thì bất kỳ yêu cầu nào đến điểm cuối `/profile` hoặc `/order` có cùng session ID sẽ đợi cho request đầu tiên kết thúc rồi mới xử lý tiếp:

```
Route::post('/profile', function () {
  //
})->block($lockSeconds = 10, $waitSeconds = 10)

Route::post('/order', function () {
  //
})->block($lockSeconds = 10, $waitSeconds = 10)
```

Phương thức `block` chấp nhận hai đối số tùy chọn. Đối số đầu tiên được phương thức `block` chấp nhận là số giây tối đa mà khóa phiên sẽ được hiệu lực trước khi nó được giải phóng. Tất nhiên, nếu yêu cầu kết thúc thực hiện trước thời điểm này, khóa sẽ được phát hành sớm hơn.

Đối số thứ hai được phương thức block chấp nhận là số giây mà một yêu cầu phải đợi trong khi cố gắng lấy một session lock. `Illuminate\Contracts\Cache\LockTimeoutException` sẽ được ném ra nếu yêu cầu không thể có được session lock trong một số giây nhất định.

Nếu cả hai đối số này đều không được truyền vào, quá trình lock sẽ được nhận trong tối đa 10 giây và các yêu cầu sẽ đợi tối đa 10 giây trong khi cố gắng lấy session lock:

```
Route::post('/profile', function () {
  //
})->block()
```

Bổ sung driver session

Chạy driver

Nếu không có session driver hiện có nào phù hợp với nhu cầu của ứng dụng của bạn, Laravel sẽ giúp bạn viết session driver của riêng mình. Session driver bổ sung của bạn nên thực thi interface `SessionHandlerInterface` có sẵn trong PHP. Interface này chỉ chứa một số phương thức đơn giản. Thực hiện driver đơn giản trên MongoDB như sau:

```
<?php
namespace App\Extensions;
class MongoSessionHandler implements \SessionHandlerInterface
{
  public function open($savePath, $sessionName) {}
  public function close() {}
  public function read($sessionId) {}
  public function write($sessionId, $data) {}
  public function destroy($sessionId) {}
  public function gc($lifetime) {}
}
```

> Laravel không gửi kèm thư mục để chứa các tiện ích mở rộng của bạn. Bạn có thể tự do đặt chúng ở bất cứ đâu bạn thích. Trong ví dụ này, chúng tôi đã tạo một thư mục *Extensions* để chứa `MongoSessionHandler`.

Vì mục đích của các phương pháp này không dễ hiểu, chúng tôi sẽ nhanh chóng trình bày những gì mỗi phương pháp thực hiện:

- Phương thức `open` thường được sử dụng trong hệ thống lưu trữ session dựa trên tập tin. Vì Laravel mặc định đi kèm với session driver trên file, nên bạn sẽ hiếm khi cần đặt bất cứ thứ gì vào phương thức này. Bạn chỉ cần để trống phương thức này.
- Phương thức `close`, giống như phương thức `open`, cũng có thể thường bị bỏ qua. Đối với hầu hết các driver, nó không cần thiết.
- Phương thức `read` sẽ trả về phiên bản chuỗi của dữ liệu session được liên kết với `$sessionId` đã cho. Không cần phải nén hay mã hóa khi truy xuất hoặc lưu trữ dữ liệu session trong driver của bạn, vì Laravel sẽ thực hiện nén cho bạn.
- Phương thức `write` sẽ ghi chuỗi `$data` nào đó có liên kết với `$sessionId` vào một số hệ thống lưu trữ ổn định, chẳng hạn như MongoDB hoặc một hệ thống lưu trữ khác mà bạn biết. Một lần nữa, bạn không nên thực hiện bất kỳ việc nén (serialization) nào bởi Laravel sẽ xử lý việc đó cho bạn.
- Phương thức `destroy` sẽ xóa dữ liệu được liên kết với `$sessionId` khỏi hệ thống

lưu trữ ổn định.
- Phương thức `gc` sẽ hủy tất cả dữ liệu session cũ hơn dấu thời gian UNIX đã cho biến `$lifetime`. Đối với các hệ thống tự hết hạn như Memcached và Redis, phương thức này có thể để trống.

Đăng ký driver

Khi driver của bạn đã được triển khai, bạn đã sẵn sàng đăng ký nó với Laravel. Để thêm driver bổ sung vào hệ thống cấp phát session của Laravel, bạn có thể sử dụng phương thức `extend` được cung cấp bởi facade Session. Bạn nên gọi phương thức `extend` từ phương thức `boot` của một service provider. Bạn có thể thực hiện việc này từ `App\Providers\AppServiceProvider` hiện có hoặc tạo một service provider hoàn toàn mới:

```
<?php

namespace App\Providers;

use App\Extensions\MongoSessionHandler;
use Illuminate\Support\Facades\Session;
use Illuminate\Support\ServiceProvider;

class SessionServiceProvider extends ServiceProvider
{
  /**
   * Register any application services.
   *
   * @return void
   */
  public function register()
  {
    //
  }
  /**
   * Bootstrap any application services.
   *
   * @return void
   */
  public function boot()
```

```
  {
    Session::extend('mongo', function ($app) {
      // Return an implementation of SessionHandlerInterface...
      return new MongoSessionHandler;
    });
  }
}
```

Khi session driver đã được đăng ký, bạn có thể sử dụng driver mongo trong tập tin cấu hình `config/session.php` của mình.