Course - Laravel Framework

Event - Sự kiện

Các sự kiện của Laravel cung cấp một bản thực thi pattern theo dõi đơn giản, cho phép bạn đăng ký và theo dõi các event khác nhau xảy ra trong ứng dụng của bạn.

Tags: laravel event, laravel

Giới thiệu

Các sự kiện của Laravel cung cấp một bản thực thi biểu mẫu theo dõi đơn giản, cho phép bạn đăng ký và theo dõi các event khác nhau xảy ra trong ứng dụng của bạn. Các class event thường được lưu trữ trong thư mục *app/Events*, trong khi các chương trình theo dõi của chúng được lưu trữ trong thư mục *app/Listeners*. Đừng lo lắng nếu bạn không thấy các thư mục này trong ứng dụng của mình vì chúng sẽ được tự động tạo ra cho bạn khi bạn tạo event và chương trình theo dõi bằng cách sử dụng lệnh Artisan console.

Event là một cách tuyệt vời để phân biệt rạch ròi các khía cạnh khác nhau của ứng dụng của bạn, vì một event duy nhất có thể có nhiều chương trình theo dõi không phụ thuộc vào nhau. Ví dụ: bạn có thể muốn gửi thông báo Slack cho người dùng của mình mỗi khi đơn đặt hàng được giao. Thay vì ghép mã xử lý đơn đặt hàng của bạn với mã token Slack, bạn có thể nêu ra một event *App\Events\OrderShipped* mà chương trình theo dõi có thể nhận và sử dụng để gửi thông báo Slack.

Đăng ký event & chương trình theo dõi

Phần mềm middleware *App\Providers\EventServiceProvider* được kèm với ứng dụng Laravel của bạn sẽ cung cấp một nơi thuận tiện để đăng ký tất cả các chương trình theo dõi event của ứng dụng của bạn. Thuộc tính `listen` chứa một mảng tất cả các event (khóa) và chương trình theo dõi (giá trị) của chúng. Bạn có thể thêm bao nhiêu event vào mảng này tuỳ ý theo yêu cầu của ứng dụng của bạn. Ví dụ: thêm một event `OrderShipped`:

```
use App\Events\OrderShipped;
use App\Listeners\SendShipmentNotification;

/**
 * The event listener mappings for the application.
 *
 * @var array
 */
protected $listen = [
  OrderShipped::class => [
    SendShipmentNotification::class,
  ],
];
```

> Lệnh `event:list` có thể được sử dụng để hiển thị danh sách tất cả các event và chương trình theo dõi được đăng ký bởi ứng dụng của bạn.

Tạo event và chương trình theo dõi

Tất nhiên, việc tạo thủ công các tập tin cho từng event và chương trình theo dõi là rất phức tạp. Thay vào đó, hãy thêm chương trình theo dõi và event vào của bạn `EventServiceProvider` và sử dụng lệnh Artisan `event:generate`. Lệnh này sẽ tạo ra bất kỳ event hoặc chương trình theo dõi nào được liệt kê trong `EventServiceProvider` của bạn mà chưa tồn tại ở dạng vật lý:

```
php artisan event:generate
```

Ngoài ra, bạn có thể sử dụng lệnh Artisan `make:event` và `make:listener` để tạo các event và chương trình theo dõi riêng lẻ:

```
php artisan make:event PodcastProcessed

php artisan make:listener SendPodcastNotification --event=PodcastProcessed
```

Đăng ký event theo cách thủ công

Thông thường, các event nên được đăng ký thông qua mảng `$listen` của `EventServiceProvider`; tuy nhiên, bạn cũng có thể đăng ký class hoặc hàm xử lý event theo cách thức thủ công trong phương thức `boot` của `EventServiceProvider`:

```
use App\Events\PodcastProcessed;
use App\Listeners\SendPodcastNotification;
use Illuminate\Support\Facades\Event;

/**
 * Register any other events for your application.
 *
 * @return void
 */
public function boot()
```

```
{
  Event::listen(
    PodcastProcessed::class,
    [SendPodcastNotification::class, 'handle']
  );

  Event::listen(function (PodcastProcessed $event) {
    //
  });
}
```

Bố trí hàng đợi cho chương trình theo dõi event nặc danh

Khi đăng ký chương trình theo dõi sự kiện dựa trên hàm xử lý theo cách thủ công, bạn có thể bọc hàm của chương trình theo dõi trong hàm `Illuminate\Events\queueable` nhằm cho Laravel biết để thực thi chương trình theo dõi bằng cách sử dụng phương pháp sắp xếp hàng chờ:

```
use App\Events\PodcastProcessed;
use function Illuminate\Events\queueable;
use Illuminate\Support\Facades\Event;

/**
 * Register any other events for your application.
 *
 * @return void
 */
public function boot()
{
  Event::listen(queueable(function (PodcastProcessed $event) {
    //
  }));
}
```

Giống như các tác vụ được sắp xếp hàng chờ, bạn có thể sử dụng các phương thức `onConnection`, `onQueue` và `delay` để tùy chỉnh việc thực thi chương trình theo dõi được xếp hàng chờ:

```
Event::listen(queueable(function (PodcastProcessed $event) {
  //
})->onConnection('redis')->onQueue('podcasts')->delay(now()->addSeconds(10)));
```

Nếu bạn muốn xử lý lỗi chương trình theo dõi được xếp hàng đợi, bạn có thể đưa ra một hàm nặc danh vào trong phương thức `catch` trong khi chỉ định chương trình theo dõi `queueable`. Hàm này sẽ nhận được đối tượng event và đối tượng `Throwable` gây ra lỗi của chương trình theo dõi:

```
use App\Events\PodcastProcessed;
use function Illuminate\Events\queueable;
use Illuminate\Support\Facades\Event;
use Throwable;

Event::listen(queueable(function (PodcastProcessed $event) {
  //
})->catch(function (PodcastProcessed $event, Throwable $e) {
  // The queued listener failed...
}));
```

Sử dụng chương trình theo dõi event với wildcard

Bạn thậm chí có thể đăng ký chương trình theo dõi bằng cách sử dụng tham số ký tự đại diện *, cho phép bạn bắt nhiều event trên cùng một chương trình theo dõi. Chương trình theo dõi nhận tên event làm đối số đầu tiên và toàn bộ mảng dữ liệu event làm đối số thứ hai:

```
Event::listen('event.*', function ($eventName, array $data) {
  //
});
```

Khám phá sự kiện

Thay vì đăng ký các event và chương trình theo dõi theo cách thủ công trong mảng `$listen` của `EventServiceProvider`, bạn có thể bật tính năng tự động phát hiện event.

Khi tính năng khám phá event được bật, Laravel sẽ tự động tìm và đăng ký các event cũng như chương trình theo dõi của bạn bằng cách quét thư mục ứng dụng của bạn */Listeners*. Ngoài ra, mọi event được khai báo minh bạch được liệt kê trong `EventServiceProvider` sẽ vẫn được đăng ký.

Laravel tìm các chương trình xử lý event bằng cách quét các class chương trình theo dõi bằng cách sử dụng các service reflection của PHP. Khi Laravel tìm thấy bất kỳ phương thức nào của class chương trình theo dõi mà sẽ bắt đầu bằng `handle`, Laravel sẽ đăng ký các phương thức đó làm chương trình theo dõi event cho event được khai báo kiểu trong phần tham số của phương thức:

```
use App\Events\PodcastProcessed;

class SendPodcastNotification
{
  /**
   * Handle the given event.
   *
   * @param  \App\Events\PodcastProcessed  $event
   * @return void
   */
  public function handle(PodcastProcessed $event)
  {
    //
  }
}
```

Tính năng khám phá event bị tắt theo mặc định, nhưng bạn có thể bật tính năng này bằng cách ghi đè phương thức `shouldDiscoverEvents` của `EventServiceProvider` trong ứng dụng của bạn:

```
/**
 * Determine if events and listeners should be automatically discovered.
 *
 * @return bool
 */
public function shouldDiscoverEvents()
{
```

```
    return true;
}
```

Theo mặc định, tất cả người nghe trong thư mục ứng dụng của `app/Listeners` bạn sẽ được quét. Nếu bạn muốn xác định các thư mục bổ sung để quét, bạn có thể ghi đè phương thức `discoverEventsWithin` trong `EventServiceProvider`:

```
/**
 * Get the listener directories that should be used to discover events.
 *
 * @return array
 */
protected function discoverEventsWithin()
{
    return [
        $this->app->path('Listeners'),
    ];
}
```

Khám phá event trong thành phẩm

Trong thành phẩm, framework nếu quét tất cả chương trình theo dõi của bạn theo mọi yêu cầu sẽ không hiệu quả. Do đó, trong quá trình xuất bản, bạn nên chạy lệnh Artisan `event:cache` để lưu vào bộ nhớ cache trong một tập tin kê khai của tất cả các event và chương trình theo dõi ứng dụng của bạn. Bản kê khai này sẽ được framework sử dụng để tăng tốc quá trình đăng ký event. Lệnh `event:clear` có thể được sử dụng để phá hủy bộ nhớ cache.

Khai báo event

Một class của event về cơ bản là một container dữ liệu chứa thông tin liên quan đến event. Ví dụ: giả sử một event `App\Events\OrderShipped` nhận được một đối tượng Eloquent ORM:

```
<?php

namespace App\Events;

use App\Models\Order;
```

```php
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;

class OrderShipped
{
  use Dispatchable, InteractsWithSockets, SerializesModels;

  /**
   * The order instance.
   *
   * @var \App\Models\Order
   */
  public $order;

  /**
   * Create a new event instance.
   *
   * @param  \App\Models\Order  $order
   * @return void
   */
  public function __construct(Order $order)
  {
    $this->order = $order;
  }
}
```

Như bạn có thể thấy, class event này không chứa logic. Nó là một container cho ví dụ `App\Models\Order` đã được mua. Trait `SerializesModels` được sử dụng bởi event sẽ nén bất kỳ mô hình Eloquent nào nếu đối tượng event được nén bằng cách sử dụng hàm `serialize` của PHP, chẳng hạn như khi sử dụng chương trình theo dõi được xếp hàng chờ.

Khai báo chương trình theo dõi

Tiếp theo, hãy xem xét chương trình theo dõi cho event ví dụ của chúng ta. Chương trình theo dõi sự kiện sẽ nhận các đối tượng event trong phương thức `handle` của họ. Các lệnh

Artisan `event:generate` và `make:listener` sẽ tự động nhập class event thích hợp và nhập gợi ý event trên phương thức `handle`. Trong phương thức handle, bạn có thể thực hiện bất kỳ hành động nào cần thiết để phản hồi sự kiện:

```php
<?php

namespace App\Listeners;

use App\Events\OrderShipped;

class SendShipmentNotification
{
  /**
   * Create the event listener.
   *
   * @return void
   */
  public function __construct()
  {
    //
  }

  /**
   * Handle the event.
   *
   * @param  \App\Events\OrderShipped  $event
   * @return void
   */
  public function handle(OrderShipped $event)
  {
    // Access the order using $event->order...
  }
}
```

> Chương trình theo dõi event của bạn cũng có thể nhập gợi ý bất kỳ event nào chúng cần vào các constructor của chúng. Tất cả các chương trình xử lý event sẽ được trả lại thông qua service container của Laravel, vì vậy các thư viện sẽ được đưa vào tự động.

Ngừng phán tán một event

Đôi khi, bạn có thể muốn dừng việc phán tán một event cho những chương trình theo dõi khác. Bạn có thể làm như vậy bằng cách trả lại `false` từ phương thức `handle` của chương trình theo dõi của bạn.

Chương trình theo dõi event được xếp hàng chờ

Xếp hàng chương trình theo dõi có thể có lợi nếu chương trình theo dõi của bạn thực hiện một tác vụ kiểu chậm như gửi email hoặc thực hiện một HTTP request. Trước khi sử dụng chương trình theo dõi được xếp hàng chờ, hãy đảm bảo cấu hình hàng chờ của bạn và khởi động chương trình xử lý hàng chờ trên máy chủ hoặc môi trường local của bạn.

Để chỉ định rằng một chương trình theo dõi phải được xếp hàng đợi, hãy thêm interface `ShouldQueue` vào class chương trình theo dõi. Các chương trình theo dõi sẽ được tạo bởi lệnh Artisan `event:generate` và `make:listener`, nó có sẵn namespace để bạn có thể sử dụng nó ngay lập tức:

```
<?php

namespace App\Listeners;

use App\Events\OrderShipped;
use Illuminate\Contracts\Queue\ShouldQueue;

class SendShipmentNotification implements ShouldQueue
{
  //
}
```

Bây giờ, khi một event được xử lý bởi chương trình theo dõi này sẽ được gửi đi, chương trình xử lý sẽ tự động được xếp hàng bởi chương trình điều phối event bằng cách sử dụng hệ thống hàng chờ của Laravel. Nếu không có exception nào được ném ra khi chương trình theo dõi được thực thi bởi hàng chờ, tác vụ được xếp hàng sẽ tự động bị xóa sau khi nó được xử lý hoàn tất.

Điều chỉnh kết nối hàng đợi & tên hàng đợi

Nếu bạn muốn điều chỉnh kết nối hàng chờ, tên hàng chờ hoặc thời gian trễ hàng chờ của

chương trình xử lý event, bạn có thể khai báo thuộc tính `$connection` hoặc `$queue`, `$delay` trên class chương trình theo dõi của mình:

```php
<?php
namespace App\Listeners;
use App\Events\OrderShipped;
use Illuminate\Contracts\Queue\ShouldQueue;

class SendShipmentNotification implements ShouldQueue
{
  /**
   * The name of the connection the job should be sent to.
   *
   * @var string|null
   */
  public $connection = 'sqs';

  /**
   * The name of the queue the job should be sent to.
   *
   * @var string|null
   */
  public $queue = 'listeners';

  /**
   * The time (seconds) before the job should be processed.
   *
   * @var int
   */
  public $delay = 60;
}
```

Nếu bạn muốn xác định kết nối hàng chờ của chương trình theo dõi hoặc tên hàng chờ trong thời gian chạy, bạn có thể khai báo các phương thức `viaConnection` hoặc `viaQueue` trên chương trình theo dõi:

```
/**
```

```
     * Get the name of the listener's queue connection.
     *
     * @return string
     */
    public function viaConnection()
    {
        return 'sqs';
    }

    /**
     * Get the name of the listener's queue.
     *
     * @return string
     */
    public function viaQueue()
    {
        return 'listeners';
    }
```

Chương trình theo dõi được xếp hàng chờ có kèm điều kiện

Đôi khi, bạn có thể cần phải xác định xem một chương trình theo dõi có nên được xếp hàng chờ hay không dựa trên một số dữ liệu mà chỉ có sẵn trong thời gian chạy. Để thực hiện điều này, phương thức `shouldQueue` có thể được thêm vào chương trình theo dõi để xác định xem chương trình theo dõi có nên được xếp hàng chờ hay không. Nếu phương thức `shouldQueue` trả về `false`, chương trình theo dõi sẽ không được thực thi:

```
<?php

namespace App\Listeners;

use App\Events\OrderCreated;
use Illuminate\Contracts\Queue\ShouldQueue;

class RewardGiftCard implements ShouldQueue
{
    /**
     * Reward a gift card to the customer.
     *
```

```
     * @param  \App\Events\OrderCreated  $event
     * @return void
     */
    public function handle(OrderCreated $event)
    {
        //
    }

    /**
     * Determine whether the listener should be queued.
     *
     * @param  \App\Events\OrderCreated  $event
     * @return bool
     */
    public function shouldQueue(OrderCreated $event)
    {
        return $event->order->subtotal >= 5000;
    }
}
```

Tương tác thủ công với hàng chờ

Nếu bạn cần truy cập thủ công các phương thức `delete` và công việc hàng chờ cơ bản của chương trình theo dõi, bạn có thể làm điều đó bằng cách sử dụng trait `Illuminate\Queue\InteractsWithQueue`. Trait này được nhập theo mặc định trên các chương trình theo dõi đã tạo và cung cấp quyền truy cập vào các phương thức sau:

```
<?php

namespace App\Listeners;

use App\Events\OrderShipped;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Queue\InteractsWithQueue;

class SendShipmentNotification implements ShouldQueue
{
```

```
    use InteractsWithQueue;

    /**
     * Handle the event.
     *
     * @param  \App\Events\OrderShipped  $event
     * @return void
     */
    public function handle(OrderShipped $event)
    {
        if (true) {
            $this->release(30);
        }
    }
}
```

Chương trình xử lý event được xếp hàng chờ & transaction cơ sở dữ liệu

Khi các chương trình theo dõi trong hàng chờ được gửi đi trong các transaction cơ sở dữ liệu, chúng có thể được hàng chờ xử lý trước khi transaction cơ sở dữ liệu được cam kết. Khi điều này xảy ra, bất kỳ cập nhật nào bạn đã thực hiện cho các model hoặc record cơ sở dữ liệu trong quá trình transaction cơ sở dữ liệu có thể chưa được phản ánh trong cơ sở dữ liệu. Ngoài ra, bất kỳ model hoặc record cơ sở dữ liệu nào được tạo trong transaction có thể không tồn tại trong cơ sở dữ liệu. Nếu chương trình theo dõi của bạn phụ thuộc vào các model này, các lỗi không mong muốn có thể xảy ra khi công việc điều động chương trình theo dõi được xếp hàng chờ được xử lý.

Nếu tùy chọn cấu hình `after_commit` của kết nối hàng chờ của bạn được đặt thành `false`, bạn vẫn có thể chỉ ra rằng một chương trình theo dõi được xếp hàng cụ thể sẽ được gửi đi sau khi tất cả các transaction cơ sở dữ liệu mở đã được cam kết bằng cách xác định một thuộc tính `$afterCommit` trên class chương trình theo dõi:

```
<?php

namespace App\Listeners;

use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Queue\InteractsWithQueue;

class SendShipmentNotification implements ShouldQueue
```

```
{
  use InteractsWithQueue;

  public $afterCommit = true;
}
```

> Để tìm hiểu thêm về cách giải quyết những vấn đề này, vui lòng xem lại tài liệu liên quan đến các tác vụ được xếp hàng chờ và transaction trong cơ sở dữ liệu.

Xử lý công việc không hoàn thành

Đôi khi chương trình theo dõi được xếp hàng chờ của bạn có thể thất bại. Nếu chương trình theo dõi trong hàng chờ vượt quá số lần thử tối đa do hệ thống hàng chờ của bạn xác định, phương thức `failed` sẽ được gọi trên chương trình theo dõi của bạn. Phương thức `failed` sẽ nhận đối tượng event và event `Throwable` gây ra lỗi:

```
<?php
namespace App\Listeners;
use App\Events\OrderShipped;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Queue\InteractsWithQueue;

class SendShipmentNotification implements ShouldQueue
{
  use InteractsWithQueue;

  /**
   * Handle the event.
   *
   * @param  \App\Events\OrderShipped  $event
   * @return void
   */
  public function handle(OrderShipped $event)
  {
    //
  }
```

```
    /**
     * Handle a job failure.
     *
     * @param  \App\Events\OrderShipped  $event
     * @param  \Throwable  $exception
     * @return void
     */
    public function failed(OrderShipped $event, $exception)
    {
        //
    }
}
```

Chỉ định nỗ lực tối đa của chương trình theo dõi được xếp hàng chờ

Nếu một trong những chương trình theo dõi trong hàng chờ của bạn gặp lỗi, bạn có thể không muốn nó tiếp tục thử lại vô thời hạn. Do đó, Laravel cung cấp nhiều cách khác nhau để chỉ định số lần hoặc thời hạn mà một chương trình theo dõi có thể được thử.

Bạn có thể xác định thuộc tính `$tries` trên class chương trình theo dõi của mình để chỉ định số lần mà chương trình theo dõi có thể được thử trước khi nó được coi là không thành công:

```
<?php

namespace App\Listeners;

use App\Events\OrderShipped;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Queue\InteractsWithQueue;

class SendShipmentNotification implements ShouldQueue
{
    use InteractsWithQueue;

    /**
     * The number of times the queued listener may be attempted.
     *
     * @var int
```

```
     */
    public $tries = 5;
}
```

Để thay thế cho việc xác định số lần chương trình theo dõi có thể được thử trước khi thất bại, bạn có thể xác định thời điểm mà chương trình theo dõi không nên thử nữa. Điều này cho phép chương trình theo dõi được thử bất kỳ số lần nào trong một khung thời gian nhất định. Để xác định thời gian mà một chương trình theo dõi không còn được thử lại nữa, hãy thêm phương thức `retryUntil` vào class chương trình theo dõi của bạn. Phương thức này sẽ trả về một đối tượng `DateTime`:

```
/**
 * Determine the time at which the listener should timeout.
 *
 * @return \DateTime
 */
public function retryUntil()
{
    return now()->addMinutes(5);
}
```

Điều phối sự kiện

Để gửi một event, bạn có thể gọi phương thức tĩnh `dispatch` trên event đó. Phương pháp này được tạo sẵn trên event bởi trait `Illuminate\Foundation\Events\Dispatchable`. Bất kỳ đối số nào được truyền cho phương thức `dispatch` cũng sẽ được truyền vào constructor của event:

```
<?php
namespace App\Http\Controllers;
use App\Events\OrderShipped;
use App\Http\Controllers\Controller;
use App\Models\Order;
use Illuminate\Http\Request;

class OrderShipmentController extends Controller
```

```
{
  /**
   * Ship the given order.
   *
   * @param  \Illuminate\Http\Request  $request
   * @return \Illuminate\Http\Response
   */
  public function store(Request $request)
  {
    $order = Order::findOrFail($request->order_id);

    // Order shipment logic...

    OrderShipped::dispatch($order);
  }
}
```

Khi kiểm tra, có thể hữu ích khi khẳng định rằng một số event nhất định đã được gửi đi mà không thực sự kích hoạt chương trình theo dõi của chúng. Chương trình trợ giúp testing có sẵn trong Laravel khiến nó trở nên dễ dàng.

Chương trình đăng ký event

Tạo ra chương trình đăng ký event

Chương trình đăng ký event là các class có thể đăng ký nhiều event từ chính class chương trình đăng ký, cho phép bạn xác định một số chương trình xử lý event trong một class duy nhất. Chương trình đăng ký nên khai báo phương thức `subscribe`, phương thức này sẽ được truyền vào một đối tượng điều phối event. Bạn có thể gọi phương thức `listen` trên chương trình điều phối đã cho để đăng ký chương trình xử lý event:

```
<?php

namespace App\Listeners;

use Illuminate\Auth\Events\Login;
use Illuminate\Auth\Events\Logout;
```

```php
class UserEventSubscriber
{
  /**
   * Handle user login events.
   */
  public function handleUserLogin($event) {}

  /**
   * Handle user logout events.
   */
  public function handleUserLogout($event) {}

  /**
   * Register the listeners for the subscriber.
   *
   * @param  \Illuminate\Events\Dispatcher  $events
   * @return void
   */
  public function subscribe($events)
  {
    $events->listen(
      Login::class,
      [UserEventSubscriber::class, 'handleUserLogin']
    );

    $events->listen(
      Logout::class,
      [UserEventSubscriber::class, 'handleUserLogout']
    );
  }
}
```

Nếu các phương thức của chương trình xử lý event của bạn được khai báo trong chính chương trình đăng ký, bạn có thể thấy thuận tiện hơn khi trả về một mảng event và tên phương thức từ phương thức `subscribe` của chương trình đăng ký. Laravel sẽ tự động xác định tên class của chương trình đăng ký khi đăng ký chương trình theo dõi event:

```php
<?php
namespace App\Listeners;
use Illuminate\Auth\Events\Login;
use Illuminate\Auth\Events\Logout;

class UserEventSubscriber
{
  /**
   * Handle user login events.
   */
  public function handleUserLogin($event) {}

  /**
   * Handle user logout events.
   */
  public function handleUserLogout($event) {}

  /**
   * Register the listeners for the subscriber.
   *
   * @param  \Illuminate\Events\Dispatcher  $events
   * @return array
   */
  public function subscribe($events)
  {
    return [
      Login::class => 'handleUserLogin',
      Logout::class => 'handleUserLogout',
    ];
  }
}
```

Đăng ký chương trình đăng ký event

Sau khi viết chương trình đăng ký, bạn có thể sẵn sàng đăng ký nó với chương trình điều phối event. Bạn có thể đăng ký chương trình đăng ký bằng cách sử dụng thuộc tính

`$subscribe` trên `EventServiceProvider`. Ví dụ: hãy thêm danh sách `UserEventSubscriber` vào danh sách:

```php
<?php
namespace App\Providers;
use App\Listeners\UserEventSubscriber;
use Illuminate\Foundation\Support\Providers\EventServiceProvider as ServiceProvider;

class EventServiceProvider extends ServiceProvider
{
  /**
   * The event listener mappings for the application.
   *
   * @var array
   */
  protected $listen = [
    //
  ];

  /**
   * The subscriber classes to register.
   *
   * @var array
   */
  protected $subscribe = [
    UserEventSubscriber::class,
  ];
}
```