Consola Artisan

Introducción

Artisan es el nombre de la interfaz de línea de comandos incluida en Laravel. Provee un gran número de comandos útiles en el desarrollo de su aplicación. Artisan está basado en el poderoso componente de consola de Symfony. Para ver una lista completa de los comandos, puede usar el comando `list`:

```
php artisan list
```

Cada comando include la opción de ayuda pasando como parámetro "help" que describe con mas detalle los argumentos y las opciones disponibles para ese comando. Para mostrar la ayuda en la consola, simplemente precede el comando con `help`:

```
php artisan help migrate
```

Crear comandos

Adicional a los comandos incluidos en Artisan, usted tiene la opción de crear su propio comando para su aplicación. Debe guardar sus comandos en la carpeta `app/Console/Commands`; pero, puede guardarlos en otra carpeta siempre y cuando pueden ser autocargados según sus configuraciones en su archivo `composer.json`.

Para crear un nuevo comando, debe usar el comando Artisan `make:console`, que generará una plantilla con código básico:

```
php artisan make:console SendEmails
```

El comando de arriba crea una clase en `app/Console/Commands/SendEmails.php`. Cuando se genera el comando, la opción `--command` puede ser usado para asignar el nombre del comando que usaremos en la consola:

```
php artisan make:console SendEmails --
command=emails:send
```

Estructura de un comando

Una vez creado el comando, usted deberá llenar las propriedades `signature` y `description` en la clase, que se usarán para mostrar su comando en la terminal cuando se usa `list`.

El método `handle` será llamado cuando su comando sea ejecutado. Usted puede meter la lógica del comando en ese método. Veremos un ejemplo de comando.

Tenga en cuenta que podemos injectar cuanquiera dependencia en el constructor del comando. EL Contenedor de servicio de Laravel

injectará todas las dependencias pasadas al constructor. Para mejor resusabilidad, mantenga sus comandos ligeros e infocados a sus tareas específicas.

```
<?php

namespace App\Console\Commands;

use App\User;
use App\DripEmailer;
use Illuminate\Console\Command;
use Illuminate\Foundation\Inspiring;

class SendEmails extends Command
{
    /**
     * The name and signature of the console
command.
     *
     * @var string
     */
    protected $signature = 'email:send {user}';

    /**
     * The console command description.
     *
     * @var string
     */
    protected $description = 'Send drip e-mails to a
 user';

    /**
     * The drip e-mail service.
     *
     * @var DripEmailer
     */
    protected $drip;

    /**
     * Create a new command instance.
     *
     * @param  DripEmailer  $drip
     * @return void
     */
    public function __construct(DripEmailer $drip)
    {
        parent::__construct();

        $this->drip = $drip;
    }

    /**
     * Execute the console command.
     *
     * @return mixed
     */
    public function handle()
    {
```

```
        $this->drip->send(User::find($this-
>argument('user')));
    }
}
```

I/O (Entrada/Salida) de Comando

Definiendo entradas

Cuando se crea comando, es muy común recibir entradas por parte del usuario final por medio de argumentos o opciones. Con Laravel es muy conveniente definir la entrada esperada del usuario usando la propriedad `signature`. La propriedad `signature` le permite definir el nombre, los argumentos y las opciones para el comando en un sólo, expresivo síntaxis parecido al de las rutas.

Todos los argumentos y optiones ingresados por el usuario están encerrados en llaves, por ejemplo:

```
/**
 * The name and signature of the console command.
 *
 * @var string
 */
protected $signature = 'email:send {user}';
```

En ese ejemplo, el comando define un argumento mandatorio: `user`. También puede hacer que los argumentos sean opcionales y definir valores por defecto para esos argumentos opcionales:

```
// argumento opcional...
email:send {user?}

// Argumento optional con valor user por defecto...
email:send {user=foo}
```

Las opciones al igual de los argumentos son una forma de entrada por parte del usuario. Sin embargo, tienen el prefijo (`--`) cuando son especificados en la línea de comando. Podemos definir opciones para la signatura en la siguiente manera:

```
/**
 * The name and signature of the console command.
 *
 * @var string
 */
protected $signature = 'email:send {user} {--
queue}';
```

En ese ejemplo, la opción `--queue` puede ser usado cuando llamamos el comando Artisan. Si el usuario pasa la opción `--queue`, el valor será `true`. Si no, el valor será `false`:

```
php artisan email:send 1 --queue
```

También puede especificar que la opción debería ser asignada por el usuario agregando la opción con el sigo `=, indicando que un valor es mandatorio:

```
/**
 * The name and signature of the console command.
 *
 * @var string
 */
protected $signature = 'email:send {user} {--
queue=}';
```

En ese ejemplo, la opción puede ser pasada de la siguiente manera:

```
php artisan email:send 1 --queue=default
```

También puede asignar valores por defecto de las opciones:

```
email:send {user} {--queue=default}
```

Descripciones para las entradas

Puede proveer descripciones para los argumentos separando el parámetro de la descripción usando dos puntos:

```
/**
 * The name and signature of the console command.
 *
 * @var string
 */
protected $signature = 'email:send
```

```
                        {user : The ID of the user}
                        {--queue= : Whether the job
should be queued}';
```

Recuperar entradas

Cuando el comando se está ejecutando, usted querá accesar los valores para los argumentos y opciones pasados a su comando. Para eso, puede utilizar los métodos `argument` y `option`:

Para recuperar el valor de un argumento, utilizar el método `argument`:

```
/**
 * Execute the console command.
 *
 * @return mixed
 */
public function handle()
{
    $userId = $this->argument('user');

    //
}
```

Si necesita recuperar todos los argumentos como un arreglo `array`, llama el método `argument` sin parámetros:

```
$arguments = $this->argument();
```

Se puede recuperar las Opciones al igual como hacemos con los argumentos usando el método `option`:

```
// Retrieve a specific option...
$queueName = $this->option('queue');

// Recuperar todas las opciones...
$options = $this->option();
```

En caso de que el argumento o la opción no exista, `null` será retornado.

Solicitar entradas

Adicional a mostrar salidas, usted puede entradas al usuario durante la ejecución de su comando. El método `ask` le pedirá cosas al usuario, aceptará la entrada y pasará esa entrada al comando:

```
/**
 * Execute the console command.
 *
 * @return mixed
 */
public function handle()
{
    $name = $this->ask('What is your name?');
}
```

El método `secret` es parecido a `ask`, pero la entrada por parte del usuario no será visible en la terminal. Ese método es útil cuando se necesitar solicitar informaciones sensibles como por ejemplo contraseña:

```
$password = $this->secret('What is the password?');
```

Solicitar Confirmación

Si necesita solicitar una simple confirmación al usuarion, puede usar el método `confirm`. Por defecto, ese método retornará `false`. Pero, si el usuario entra `y`, el método retornará `true`.

```
if ($this->confirm('Do you wish to continue?
[y|N]')) {
    //
}
```

Proveer opciones al usuario

El método `anticipate` puede ser usado para proveeer una lista posible de opciones. El usuario puede sin embargo entrar otras opciones que no están en la lista.

```
$name = $this->anticipate('What is your name?',
['Taylor', 'Dayle']);
```

Para proveer una lista de opciones predefinida, debe utilizar el

método `choice`. El usuario escoge el índice de la respuesta, pero el valor de la respuesta será retornado para ser usado. Puede setear el valor retornado por defecto en caso de que el usuario no escoge nada:

```
$name = $this->choice('What is your name?',
['Taylor', 'Dayle'], false);
```

Escribir salidas en la consola

Para enviar informaciones a la consola, utilice los métodos `line`, `info`, `comment`, `question` y `error`. Cada uno de estos métodos imprimirá información en color, por ejemplo rojo para `error`.

Para mostrar un mensaje de tipo informativo al usuario, utilice el método `info`. Generalmente, la información tendrá el color verde en la consola:

```
/**
 * Execute the console command.
 *
 * @return mixed
 */
public function handle()
{
    $this->info('Display this on the screen');
}
```

Para mostrar un mensaje de tipo error al usuario, utilice el método `error`. Generalmente, la información tendrá el color rojo en la consola:

```
$this->error('Something went wrong!');
```

Si quieres mostrar una salida sin formato, usa el método `line`. Éste método no recibe ninguna declaración de color:

```
$this->line('Display this on the screen.');
```

Table Layouts

El método `table` permite presentar informaciones en forma tabular. Lo único que tiene que hacer, es pasar las cabeceras y las filas al método. El ancho y el alto serán calculados dinamicamente dependiendo en la información pasada al método:

```
$headers = ['Name', 'Email'];

$users = App\User::all(['name', 'email'])->toArray();

$this->table($headers, $users);
```

Barras de progreso

Para tareas muy pesadas, es conveniente presentar un indicador de progreso. Usando el objeto output, podemos iniciar, adelantar y parar la barra de progreso. Usted debe definir la cantidad de pasos cuando el indicativo de progreso se inicia, y luego avanzarlo después de cada paso:

```
$users = App\User::all();

$bar = $this->output->createProgressBar(count($users));

foreach ($users as $user) {
    $this->performTask($user);

    $bar->advance();
}

$bar->finish();
```

Para mas informaciones, ver la documentación sobre en componente Barra de progreso de Symfony.

Registrar comandos

Una vez definido el comando, es necesario regirtarlo con Artisan antes de poder utilizarlo. Eso se hace en el archivo `app/Console/Kernel.php`.

Dentro de ese archivo, encontrarás una lista de comando en la

propriedad `commands`. Para registrar el comando, simplemente agregar la clase a la lista. Cuando Artisan se arranca, todos los comandos en esa propriedad serán instanciados por el contenedor de servicio y regristrado con Artisan:

```
protected $commands = [
    'App\Console\Commands\SendEmails'
];
```

Ejecutar comandos con programación

Quizá algunas veces usted necesitará ejecutar un comando fuera del interpretador de línea de comando (CLI). Por ejemplo, quiere ejecutar un comando Artisa desde una ruta o un controlador. Puede utilizar el método `call` desde el facade `Artisan`. El método `call`accepta el nombre del comando como el primer argumento, y un arreglo de parámetros de comando como el segundo argumento. El código de terminación será retornado:

```
Route::get('/foo', function () {
    $exitCode = Artisan::call('email:send', [
        'user' => 1, '--queue' => 'default'
    ]);

    //
});
```

Utilizando el método `queue` del facade `Artisan`, puede hasta agregar sus comandos Artisan para que pueden ser ejecutados en segundo plano por sus ejecutadores de cola:

```
Route::get('/foo', function () {
    Artisan::queue('email:send', [
        'user' => 1, '--queue' => 'default'
    ]);

    //
});
```

Si necesitas especificar el valor de una option que no acepta valores

como cadenas, tal como la opción '--force' del comando `migrate:refresh`, puedes pasar un valor booleano `true` o `false`:

```
$exitCode = Artisan::call('migrate:refresh', [
    '--force' => true,
]);
```

Llamar comandos desde otros comandos

Algunas veces querrá llamar otros comandos desde un comando Artisan existente. Puede usar el método `call` method .Ese método toma en parámetros el nombre del comando y un arreglo de parámetros de comando:

```
/**
 * Execute the console command.
 *
 * @return mixed
 */
public function handle()
{
    $this->call('email:send', [
        'user' => 1, '--queue' => 'default'
    ]);

    //
}
```

Si desea llamar otro comando de consola y eliminar todos sus informaciones de salida, puede utilizar el método `callSilent`. Ese método tiene la misma signatura que el método `call`:

```
$this->callSilent('email:send', [
    'user' => 1, '--queue' => 'default'
]);
```