Artisan Console

۱. مقدمه

۲. نوشتن دستورات اختصاصی

- ساختار دستور

۳. ورودی/خروجی دستور (Command I/O)

- تعریف ورودی مورد انتظار
- بازیابی ورودی
- درخواست از کاربر برای ارائه ورودی
- نوشتن و چاپ خروجی

۴. ثبت دستورات

۵. فراخوانی دستورات از طریق کدنویسی

مقدمه

Artisan اسم رابط خط فرمان (command-line interface) است که همراه با فریم ورک Laravel عرضه می شود. این رابط دستورات بسیار کارآمدی برای توسعه ی اپلیکیشن در اختیار برنامه نویس قرار می دهد. این رابط مبتنی بر کامپوننت قدرتمند Symfony Console می باشد.

به منظور مشاهده ی لیستی کامل از تمامی دستورات Artisan، کافی است دستور list را در پنجره ی فرمان درج نموده و اجرا کنید:

```
php artisan list
```

کلیه ی دستورات همچنین همراه با یک صفحه ی help ارائه می شوند که آرگومان ها و تنظیمات (آپشن های) دستور حاضر را نمایش و درباره ی هریک شرح می دهد. جهت مشاهده ی صفحه ی help، کافی است پیش از اسم دستور مورد نظر واژه ی help را تایپ نمایید:

```
php artisan help migrate
```

نوشتن دستورات اختصاصی

علاوه بر دستورات درون ساخته ی کنسول Artisan، می توانید دستورات سفارشی خود را برای کار با برنامه تنظیم نمایید. می توانید دستورات اختصاصی خود را در پوشه ی app/Console/Commands ذخیره نمایید. البته شما آزادید مکان دلخواه خود را برای نگهداری دستورات تعیین کنید، منوط به اینکه دستورات بر اساس تنظیمات composer.json به صورت خودکار بارگذاری شوند (قابل autoload باشند).

به منظور ساخت یک دستور جدید، می توانید دستور make:console آرتیزان را اجرا نمایید. این دستور یک stub (دستور پایه ای) برای شروع کار ایجاد می نماید:

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

دستور فوق یک کلاس در آدرس app/Console/Commands/SendEmails.php ایجاد می کند. به هنگام ایجاد یک دستور جدید، می توانید از آپشن command-- برای تخصیص اسم به دستور استفاده نمایید:

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

DADEHA

ساختار دستور

پس از ایجاد دستور سفارشی خود، بایستی property های signature و description کلاس را مقداردهی نمایید. از مقادیر این property ها به هنگام نمایش دستور مورد نظر در صفحه ی list، استفاده می شود.

متد handle در زمان اجرای دستور ذکر شده، صدا زده می شود. می توانید منطق دستور دلخواه را در این متد جایگذاری نمایید. در زیر به توصیف یک دستور نمونه می پردازیم.

یادآور می شویم که به راحتی می توان dependency مورد نیاز را در تابع constructor دستور مزبور تزریق کرد. service container فریم ورک Laravel تزریق کلیه ی dependency های اعلان نوع (type-hint شده) در constructor را به صورت خودکار انجام می دهد. به منظور بهبود قابلیت استفاده ی مجدد از code، توصیه می شود دستورات کنسول خود را تا حد امکان کم حجم، سبک نگه داشته و آن ها را در مراجعه و کمک گرفتن از سرویس های اپلیکشن جهت انجام عملیات مورد نظر آزاد بگذارید.

آموزشگاه تحلیلگر داده ها

```php
<?php
namespace App\Console\Commands;
use App\User;
use App\DripEmailer;
use Illuminate\Console\Command;
class SendEmails extends Command
{

  protected $signature = 'email:send {user}';


  protected $description = 'Send drip e-mails to a user';

  protected $drip;

  public function __construct(DripEmailer $drip)
  {
    parent::__construct();
    $this->drip = $drip;
  }

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

ورودی/خروجی دستور (Command I/O)

تعریف ورودی مورد انتظار

به هنگام نوشتن دستورات کنسول معمولا از طریق آرگومان ها یا آپشن ها، ورودی هایی از کاربر دریافت می شود. Laravel این امکان را فراهم آورده تا ورودی مورد انتظار از کاربر را به راحتی مشخص نمایید. برای این منظور تنها کافی است خاصیت (property) signature را در دستورهای اختصاصی خود تعریف کنید. خاصیت مذکور به شما اجازه می دهد تا اسم، آرگومان ها و آپشن های دستور را در قالب یک خط کد بهینه و مختصر با ساختار نگارشی و سینتکس route مانند تعریف نمایید.

تمامی آرگومان ها و آپشن هایی ارائه شده توسط کاربر داخل {} محصور می شوند. همان طور که در مثال زیر مشاهده می شود، دستور یک آرگومان الزامی :user می پذیرد:

```
protected $signature = 'email:send {user}';
```

همچنین می توانید آرگومان ها را اختیاری اعلان نموده و مقادیر پیش فرضی برای این آرگومان های اختیاری در نظر بگیرید:

```
// Optional argument...
email:send {user?}
// Optional argument with default value...
email:send {user=foo}
```

آپشن ها نیز مانند آرگومان ها یک نوع user input محسوب می شوند، با یک تفاوت که آپشن ها را باید در خط فرمان با پیشوند (--) نوشت. در زیر نحوه ی تعریف آپشن در signature با یک مثال نمایش داده شده است:

```
protected $signature = 'email:send {user} {--queue}';
```

در مثال حاضر، سویچ queue-- را می توان در زمان فراخوانی دستور Artisan مشخص کرد. چنانچه سویچ گفته شده در بالا ارسال شد، آنگاه مقدار آپشن true خواهد بود. در غیر این صورت، مقدارش false می شود:

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

همچنین می توانید مشخص کنید که آپشن مورد نظر باید توسط کاربر مقداردهی شود. این کار را می توان با قرار دادن علامت تخصیص پس از اسم آپشن به نشانه ی اینکه کاربر باید مقدار را فراهم کند، انجام دهید:

```
protected $signature = 'email:send {user} {--queue=}';
```

در این مثال کاربر می تواند به صورت زیر مقداری را برای آپشن مورد نظر فراهم نماید:

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

همچنین می توان مقادیر پیش فرض به آپشن ها ارسال کرد:

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

برای اینکه بتوان در زمان تعریف آپشن یک میانبر تخصیص داد، بایستی آن را قبل از اسم آپشن درج کرده و پس از آن یک " | " برای تفکیک میانبر از اسم کامل آن آپشن استفاده نمود:

```
email:send {user} {--Q|queue}
```

در صورتی که می خواهید آرگومان ها یا آپشن ها ورودی هایی فقط از نوع آرایه بپذیرند، می بایست از کاراکتر " * " استفاده نمایید:

```
email:send {user*}
email:send {user} {--id=*}
```

شرح ورودی (Input Description)

می توانید به آرگومان های ورودی و آپشن ها شرح (description) تخصیص دهید. برای این منظور کافی است پارامتر را به وسیله ی دو نقطه از شرح آن تفکیک نمایید:

```
protected $signature = 'email:send
                {user : The ID of the user}
                {--queue= : Whether the job should be queued}';
```

بازیابی ورودی

برای دسترسی به مقادیر آرگومان ها و آپشن های دریافت شده توسط دستور مورد نظر، کافی است از دو متد argument (برای بازیابی آرگومان ها) و option (برای بازیابی آپشن ها) استفاده نمایید:

```
public function handle()
{
  $userId = $this->argument('user');

  //
}
```

به منظور بازیابی تمامی آرگومان ها به صورت array (در قالب آرایه)، فقط باید تابع argument را بدون پارامتر صدا بزنید:

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

جهت دسترسی به آپشن ها نیز تنها کافی است تابع option را فراخوانی نمایید. برای بازیابی تمامی آپشن ها به صورت آرایه، لازم است تابع نام برده را بدون پارامتر صدا بزنید:

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

```
// Retrieve all options...
$options = $this->option();
```

در صورت عدم وجود آرگومان یا آپشن مورد نظر، مقدار null برگردانده می شود.

درخواست ورودی از کاربر

علاوه بر نمایش خروجی، می توان از کاربر درخواست کرد ورودی را در حین اجرای دستور ارائه کند. متد ask یک سوال به کاربر نمایش داده و از او می خواهد که ورودی را وارد کند. سپس ورودی کاربر را دریافت کرده و آن را به دستور برمی گرداند:

```
public function handle()
{
  $name = $this->ask('What is your name?');
}
```

متد secret عملکردی نظیر متد ask دارد، با این تفاوت که ورودی کاربر زمانی که آن را در پنجره ی فرمان تایپ می کند، برای او قابل مشاهده نخواهد بود. این متد برای شرایطی که کاربر اطلاعات حساس همچون گذرواژه را وارد می کند بسیار کارآمد تلقی می شود:

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

درخواست confirmation از کاربر

برای درخواست تائیدیه از کاربر لازم است متد confirm را صدا بزنید. به صورت پیش فرض، این متد مقدار بولی false را برمی گرداند. حال اگر کاربر در پاسخ به سوال مقدار y را وارد کرد، متد مقدار true را برمی گرداند.

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

انتخاب دادن به کاربر

می توان با استفاده از متد anticipate قابلیت autocompletion برای انتخاب های احتمالی کاربر فراهم نمود. با این وجود کاربر می تواند هر پاسخی را صرف نظر پیشنهاداتی که توسط auto-completion ارائه می شود، انتخاب نماید:

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

اگرکه باید تعداد محدودی انتخاب از پیش تعیین شده در اختیار کاربر قرار دهید، در آن صورت می توانید از متد choice استفاده نمایید. بدین صورت که کاربر اندیس پاسخ مورد نظر را انتخاب کرده، ولی مقدار پاسخ به شما برگردانده می شود. در صورتی که کاربر هیچ انتخابی نکرد، می توانید یک مقدار پیش فرض تعیین کنید که به عنوان خروجی برگردانده شود:

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

نوشتن و چاپ خروجی

برای ارسال و چاپ خروجی در پنجره ی کنسول، می توانید از توابع line، info، comment، question و error استفاده نمایید. هر یک از این متدها از رنگ های ANSI برای اهداف خاص خود استفاده می کنند.

جهت نمایش پیغام هایی به کاربر، کافی است از متد info استفاده نمایید. این پیغام ها معمولا به رنگ سبز در صفحه ی کنسول به نمایش در می آیند:

```
public function handle()
{
  $this->info('Display this on the screen');
}
```

به منظور نمایش یک پیغام خطا، بایستی متد error را صدا بزنید. متن پیغام خطا معمولا به رنگ قرمز نمایش داده می شود:

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

اگر می خواهید یک خروجی ساده در کنسول نمایش دهید می بایست متد line را فراخوانی کنید. این متد هیچ رنگ منحصربفردی را دریافت نمی کند:

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

طرح بندی به صورت جدول (table layout)

متد table این زمینه را فراهم می آورد تا چندین سطر / ستون از داده ها را به درستی (در قالب یک جدول) فرمت بندی نمایید. کافی است سرستون ها (header) و سطرها را به این متد به عنوان آرگومان ارسال کنید. طول و عرض جدول بر اساس داده های مورد نظر به صورت داینامیک محاسبه شده و اعمال می شوند:

```
$headers = ['Name', 'Email'];
$users = App\User::all(['name', 'email'])->toArray();
$this->table($headers, $users);
```

Progress bar

می توانید برای کارهایی که به طول می انجامند یک نوار برای نشان دادن پیشرفت برنامه نمایش دهید. با بهره گیری از شی output می توانید progress bar را راه اندازی نموده، پیشرفت فرایند را نمایش دهید و همچنین آن را در پایان خاتمه دهید. بایستی تعداد مراحل را پیش از آغاز پیشرفت فرایند تعریف نموده و سپس آن را پس از اتمام هر مرحله یک گام جلو ببرید:

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

```
$bar = $this->output->createProgressBar(count($users));
foreach ($users as $user) {
  $this->performTask($user);
  $bar->advance();
}

$bar->finish();
```

ثبت دستورات

پس از اینکه دستور اختصاصی مورد نظر آماده ی بهره برداری شد، بایستی آن را در Artisan ثبت کرده تا برای استفاده در آینده در دسترس باشد. این عملیات داخل فایل app/Console/Kernel.php صورت می پذیرند.

داخل فایل نام برده در بالا، لیستی از تمامی دستورات موجود را در property (خاصیت) commands مشاهده می کنید. به منظور ثبت دستور، کافی است اسم کلاس را به لیست دستورات اضافه نمایید. زمانی که Artisan راه اندازی (boot) می شود، تمامی دستورات لیست شده در این property توسط service container در اختیار شما قرار گرفته و در Artisan به طور کامل به ثبت می رسند:

```
protected $commands = [
  Commands\SendEmails::class
];
```

فراخوانی دستورات از طریق کدنویسی

گاهی لازم است یک دستور Artisan را خارج از رابط خط فرمان (CLI) صدا زده و اجرا نمایید. به عنوان مثال ممکن است لازم باشد دستور Artisan مد نظر را از یک route یا controller فراخوانی و اجرا نمایید. برای نیل به این هدف کافی است متد call را در facade (فاساد) Artisan بکار ببرید. متد call اسم دستور را به عنوان آرگومان اول و آرایه ای از پارامترهای دستور مورد نظر را به عنوان آرگومان دوم می پذیرد. کد خروجی به صورت زیر بازگردانی می شود:

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

با استفاده از متد queue در فاساد Artisan، همچنین می توانید دستورات Artisan را (برای اجرا به ترتیب معین) در صف قرار داده تا بدین وسیله توسط queue worker ها در پس زمینه پردازش شوند:

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

چنانچه باید مقدار یک آپشن را که مقادیر از نوع رشته نمی پذیرد، مشخص نمایید، مانند فلگ force-- داخل دستور migrate:refresh، در آن صورت می توانید یک مقدار بولی true یا false به عنوان پارامتر ارسال نمایید:

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

```
]);
```

فراخوانی یک دستور از دستوری دیگر

گاهی لازم می شود دستورات دیگری را از دستور Artisan جاری صدا بزنید. برای این منظور کافی است تابع call را فراخوانی نمایید. تابع مزبور اسم دستور و آرایه ای از پارامترهای دستور مورد نظر را به عنوان آرگومان می پذیرد:

```
public function handle()
{
  $this->call('email:send', [
     'user' => 1, '--queue' => 'default'
  ]);
  //
}
```

چنانچه می خواهید یک دستور کنسول دیگری را صدا زده ولی تمامی خروجی های آن را سرکوب نمایید، کافی است تابع callSilent را فراخوانی نمایید. متد نام برده از نظر تعداد ورودی و نوع آن ها (signature) با تابع call یکسان است:

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

DADEHA TAHLIL TITUTE
آموزشگاه تحلیلگر داده ها