Stillat
(https://stillat.com/)

The Laravel Application Console Kernel

December 1, 2016 Laravel (/explore/categories/laravel)

JOHN KOSTER

Each application contains a console kernel. The console kernel is defined in the `app/Console/Kernel.php` (this will be referred to as the application console kernel) file. The kernel class that exists in that file extends Laravel's console kernel (which can be found in the `vendor/laravel/framework/src/Illuminate/Foundation/Console/Kernel.php` file; this kernel will be defined as the framework console kernel). From the applications point of view, the application console kernel is responsible for specifying which custom commands should be made available to users and when to automatically execute various commands and tasks (by using the task scheduler).

It is important to note that while this the application and console kernels are separated into their own sections, they are the same thing, and are only being differentiated on this site to make explaining them easier. The "application" console kernel, located in the `app/Console/Kernel.php` file, is simply a convenient way to expose the features of the "framework" console kernel to your application. Any features, limitations, or otherwise that are discussed in the section "The Framework Console Kernel" can be applied to the application console kernel class.

The application console kernel is mainly used for two purposes in relation to your application:

Registering commands for use;
Scheduling when various commands and tasks should be ran (by using the scheduler).

The kernel is fairly simple by default (the vast majority of the logic is contained within the framework's console kernel):

```
<?php

namespace App\Console;

use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel
{
    /**
     * The Artisan commands provided by your application.
     *
     * @var array
     */
    protected $commands = [
    ];

    /**
     * Define the application's command schedule.
     *
     * @param  \Illuminate\Console\Scheduling\Schedule  $schedule
     * @return void
     */
```

The kernel has an empty `$commands` array and an empty `schedule()` method. The `$commands` property is used to specify which commands are to be loaded when the console application (Artisan) starts. In the introduction to this chapter the following error message was discussed briefly:

```
[Symfony\Component\Console\Exception\CommandNotFoundException]
  Command "inspire" is not defined.
```

The most common cause of this error is the command that a user is attempting to run has not been added to the `$commands` array. The act of adding the name of a class to the `$commands` array is referred to as registering the command with the application.

To register a command with the application, simply add the fully qualified name to the command class to the `$commands` array. In earlier versions of Laravel (targeting older versions of PHP), this was done by specifying the class name as a string; however, starting with PHP version 5.5, the `::class` class constant. It is also important to remember to include the class if it is located in a different namespace (using the `use` PHP keyword). Application commands are, by default, stored within the `app/Console/Commands/` directory (under the `App\Console\Commands` namespace). Alternatively, commands can be registered using the command shorthand syntax. The `commands` method is automatically called by the framework and is a convenient place to register commands using the shorthand syntax.

In the introductory article to this series Writing Custom Laravel Artisan Commands: An Introduction (/blog/2016/12/01/writing-custom-laravel-artisan-commands-an-introduction), the `inspire` Artisan command was used as an example. This command is stored within the `app/Console/Commands/Inspire.php` file and can be accessed by using the `App\Console\Commands\Inspire` class (this is the fully qualified name of the `Inspire` command). The following examples demonstrate how to register the `Inspire` command with the application.

The following example uses the `::class` class constant, and is generally the recommended way of referring to classes:

```php
<?php

namespace App\Console;

use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel
{
    /**
     * The Artisan commands provided by your application.
     *
     * @var array
     */
    protected $commands = [
        // Remember that namespaces are relative so the
        // `Commands\Inspire` class is resolved in
        // relation to the current App\Console
        // namespace to get the final class.
        Commands\Inspire::class
    ];
```

The following example registers the `Inspire` command by supplying the fully qualified class name as a string:

```php
<?php

namespace App\Console;

use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel
{
    /**
     * The Artisan commands provided by your application.
     *
     * @var array
     */
    protected $commands = [
        'App\Console\Commands\Inspire'
    ];


    // Schedule method omitted.

}
```

Scheduling commands and tasks will be discussed in its own article.

The discussion of the Laravel Console kernel continues in the article The Laravel Framework Console Kernel (/blog/2016/12/01/the-laravel-framework-console-kernel) article.

The console kernel exposes many different public methods. This article will...
(/blog/2016/12/01/the-laravel-framework-console-kernel)

Writing Custom Laravel Arti...
It is often very useful to create custom Artisan commands specifically for...
(/blog/2016/12/01/writing-custom-laravel-artisan-commands-an-introduction)

An Introduction to Laravel'...
The Artisan Command Line Environment (CLI) is a terminal based application...
(/blog/2016/11/30/an-introduction-to-laravels-artisan-console)

Creating a Hashing Manager ...
To make it easier to easily work with all the different hashing...
(/blog/2016/11/30/creating-a-hashing-manager-for-our-custom-laravel-hashing-implementations)