Implementing and executing cron jobs

Sometimes, an application requires some background tasks, such as regenerating a site map or refreshing  
statistics. A common way to implement this is by using cron jobs. When using Yii, there is a way to use a  
command to run as a job.

In this recipe, we will see how to implement both. For our recipe, we will implement writing the current  
timestamp into a t imestamp. txt file under the protected directory.

Getting ready

Create a new yii2 -app-basic application by using the Composer, as described in the official guide at  
[http://www.yiiframework.com/doc-2.0/gurde-startinstallation.html](http://www.yiiframework.com/doc-2.0/guide-startinstallation.html).

How to do it...

Running the Hello command

Let us try to run app\commands\HelloController : : actionIndex as a shell command:

<?php

namespace app\commands;  
use yii\console\Controller;

/\*\*

\* This command echoes the first argument that you have entered.

\*/

class HelloController extends Controller  
{

/\*\*

* This command echoes what you have entered as the message.
* @param string $message the message to be echoed.

\*/

public function actionIndex($message = 'hello world')

{

echo $message . "\n";

}

}

1. Open the shell in your application directory and execute this command:

php yii

Alternatively, you also can call the following and ensure that the shell works:

. /yii

1. Type the following command for the display hello:

./yii help hello

1. The framework must display some information:

DESCRIPTION

This command echoes what you have entered as the message.

USAGE

yii hello [message] [...options...]

- message: string (defaults to 'hello world')  
the message to be echoed.

1. Run the default command action:

./yii hello

Alternatively, run the concrete index action:

./yii hello/index

1. You must now see the default phrase:

Hello world

1. Run the command with any parameter and see the response:

./yii hello 'Bond, James Bond'

Creating your own command

You also can create your own console controllers. For example, create a commands/cronController. php  
file with the sample code:

<?php

namespace app\commands;

use yii\console\Controller;  
use yii\helpers\Console;  
use Yii;

/\*\*

\* Console crontab actions  
\*/

class CronController extends Controller  
{

/\*\*

\* Regenerates timestamp  
\*/

public function actionTimestamp()

{

file\_put\_contents(Yii::getAlias('@app/timestamp.txt'), time());  
$this->stdout('Done!', Console::FG\_GREEN, Console::BOLD);

$this->stdout(PHP\_EOL);

}

}

After all is done, run the command in a shell:

./yii cron/timestamp

Then, check the response text and the existence of a new file, namely timestamp. txt.

Setting the cron schedule

Create /etc/cron. d/myapp on your Linux server and add the following row to run our command at every

midnight:

0 0

\* \* \* www-data /path/to/yii cron/timestamp >/dev/null

How it works...

A console command is defined as a controller class that extends from yii\console\controller. In the  
controller class, you define one or more actions that correspond to the subcommands of the controller.  
Within each action, you write code that implements the appropriate tasks for that particular sub-command.

When running a command, you need to specify the route to the controller action. For example, the route  
migrate/create invokes the sub-command that corresponds to the

MigrateController: :actionCreate() action method. If a route offered during the execution does not  
contain an action ID, the default action will be executed (as with a web controller).

Take care that your console controllers are placed in the directory defined in the c ontrollerNamespace  
option in your web/console. php config.

See also

* For getting more information about Yii2 console commands, refer to  
  <http://www.yiiframework.com/doc-2.0/guide-tutorial-console.html>
* In order to learn more about the Cron daemon, refer to <https://en.wikipedia.org/wiki/Cron>
* The Changing the Yii directory layout recipe for controllerNamespace