Creating CLI commands

Yii has good command-line support and allows creating reusable console commands. Console commands  
are faster to create than web GUIs. If you need to create some kind of utility for your application that will  
be used by developers or administrators, console commands are the right tool.

To show how to create a console command, we’ll create a simple command that will clean up various  
things, such as assets and temp directories.

Подготовка

Создайте новое приложение с помощью диспетчера пакетов Composer, как описано в официальном руководстве по адресу  
<http://www.yiiframework.com/doc-2.0/guide-start-installation.html>.   
По русски <http://yiiframework.domain-na.me/doc/guide/2.0/ru/start-installation>.

How to do it...

Carry out the following steps to create CLI commands:

1. Create the commands/cleanController. php file with the following code:

<?php

namespace app\commands;

use yii\console\Controller;  
use yii\helpers\FileHelper;

/\*\*

\* Removes content of assets and runtime directories.

\*/

class CleanController extends Controller  
{

public $assetPaths = ['@app/web/assets'];  
public $runtimePaths = ['©runtime'];

/\*\*

* Removes temporary assets.

\*/

public function actionAssets()

{

foreach ((array)$this->assetPaths as $path) {

$this->cleanDir($path);

}

$this->stdout('Done' . PHP\_EOL);

}

/\*\*

* Removes runtime content.

\*/

public function actionRuntime()

{

foreach ((array)$this->runtimePaths as $path) {

$this->cleanDir($path);

}

$this->stdout('Done' . PHP\_EOL);

}

private function cleanDir($dir)

{

$iterator = new \DirectoryIterator(\Yii::getAlias($dir));  
foreach($iterator as $sub) {

if(!$sub->isDot() && $sub->isDir()) {

$this->stdout('Removed ' . $sub->getPathname() . PHP\_EOL);

FileHelper::removeDirectory($sub->getPathname());

}

}

}

}

1. Now we can use our own console controller with default settings. Just run the yii shell script:

. /yii

1. Look for own clean commands:

This is Yii version 2.0.7.

The following commands are available:

* asset Allows you to combine...

asset/compress Combines and compresses the asset.

asset/template Creates template of configuration file.

* clean Removes content of assets and runtime directories.

clean/assets Removes temporary assets.

clean/runtime Removes runtime content.

* fixture Manages fixture data loading and unloading.

fixture/load (default) Loads the specified fixture data.  
fixture/unload Unloads the specified fixtures.

1. Right now run asset cleaning:

.yii clean/assets

1. See the process report:

Removed /yii-book.app/web/assets/25f82b8a  
Removed /yii-book.app/web/assets/9b3b2888  
Removed /yii-book.app/web/assets/f4307424  
Done

1. If you want to use this controller in the yii2 -app- advanced application, just specify the custom  
   working paths:

return [

'id' => 'app-console',

'basePath' => dirname( DIR ),

'bootstrap' => ['log'],

'controllerNamespace' => 'console\controllers',

'controllerMap' => [

'clean' => [

'class' => 'console\controllers\CleanController',

'assetPaths' => [

'@backend/web/assets',

'@frontend/web/assets',

],

'runtimePaths' => [

'@backend/runtime',

'@frontend/runtime',

'@console/runtime',

],

],

],

// ...

];

How it works...

All console commands should be extended from the yii\console\controller class. Since all console  
commands are run in yii\console\Application instead of yii\web\Application, we don’t have a way  
to determine the value of the @webroot alias. Also, in the yii2-app-advanced template we have backend,  
frontend, and console subdirectories by default. For this purpose, we are creating configurable public  
properties called assetPaths and runtimePaths.

The console command structure itself is like a typical controller. We are defining several actions we can  
run via yii <console command>/<command action>.

As you can see, there are no views used, so we can focus on programming tasks instead of design,  
markup, and so on. Still, you need to provide some useful output so that users will know what is going on.  
This is done through simple PHP echo statements.

If your command is relatively complex such as message or migrate bundled with Yii, it’s a good decision to  
provide some extra description of the available options and actions. It can be done by overriding the  
getHelp method:

public function getHelp()

{

$out = "Clean command allows you to clean up various temporary data Yii and an  
application are generating.\n\n";

return $out . parent::getHelp();

}

Run the following command:

./yii help clean

You can see the full output as follows:

DESCRIPTION

Clean command allows you to clean up various temporary data Yii and an application are  
generating.

Removes content of assets and runtime directories.

SUB-COMMANDS

* clean/assets Removes temporary assets.
* clean/runtime Removes runtime content.

By default, when we run the shell command:

. /yii

We have seen simplified description of all commands in the output list:

- clean Removes content of assets and runtime directories.

clean/assets Removes temporary assets.

clean/runtime Removes runtime content.

This description will be taken from comments before class and actions:

/\*\*

\* Removes content of assets and runtime directories.

\*/

class CleanController extends Controller  
{

/\*\*

* Removes temporary assets.

\*/

public function actionAssets() { ... }

* Removes runtime content.

\*/

public function actionRuntime() { ... }

}

It is optional to add descriptions for your classes. You must not do it for your own CLI commands.

See also

* The Creating reusable controllers recipe in this chapter
* The Making extensions distribution-ready recipe in this chapter