DoctrineFixturesBundle

Symfony

Plan

- Installation
- Comment créér un « fixtures »
- Accéder aux services à partir des « fixtures »
- Fractionner les « fixtures » en fichiers séparés
- Partager des objets entre « fixtures »
- Charger les fichiers de « fixtures » dans l’ordre
- Groupe de « fixtures » éxécution de certains « fixtures »
- Specification du comportement de purge

Installation

Les « fixtures » sont utilisés pour charger de « fausses » ensembles de données dans une base de données qui peut être par la suite utilisées pour les tests ou pour fournir des données interressants pendant le developpement.

Ce « bundle » est compatible avec toutenles base de données suporté par l'« ORM » Doctrine.

Dans les applications Symfony 4 ou ultérieurs qui utilisent Symfony Flex, ouvrez une console de commande, positionnez vous dans votre répertoire de projet et éxécuter la commande suivante :

```
composer require --dev orm-fixtures
```

Les données « fixtures » sont des classes PHP où vous créez des objets et les conservez dans la base de données.

Creer une « fixtures »

Apres avoir installé le bundle « DoctrineFixtureBundle », un dossier « DataFixtures » qui contiendra tout nos scripts de remplissage de notre base de données est créé dans le repertoire « src » de notre projet.

On a plusieurs manière de créér nos « fixtures » :

- Soit on créé nos fichiers nous même ;
- Soit on appele une commande

php bin/console make:fixtures AppFixtures

Cette « fixtures » sera créé dans le repertoire « DataFixtures ».

Dans ce fichier vous verez une simple classe qui hérite de la classe Fixture et contient une seule methode : load()

Dans cette methode on mettra tout l'algorithme neccessaire pour générer nos fausses données

Creer une « fixtures »

```php
// src/DataFixtures/AppFixtures.php
namespace App\DataFixtures;

use App\Entity\Product;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Persistence\ObjectManager;

class AppFixtures extends Fixture
{
    public function load(ObjectManager $manager)
    {
        // create 20 products! Bam!
        for ($i = 0; $i < 20; $i++) {
            $product = new Product();
            $product->setName('product '.$i);
            $product->setPrice(mt_rand(10, 100));
            $manager->persist($product);
        }

        $manager->flush();
    }
}
```

Creer une « fixtures »

Une fois vos « fixtures » créés, chargez-les en éxécutant la commande :

```
php bin/console doctrine:fixtures:load
```

- Par défaut, la commande load purge la base de données, supprimant toutes les données de chaque table. Pour ajouter les données de vos « fixtures », ajoutez l'option –append.
- Pour voir d'autres options pour la commande éxécutez :

```
php bin/console make:fixtures:load --help
```

Accès aux services à partir des fixtures

Dans certains cas, vous devrez peut-être accéder aux services de votre application dans une « fixtures ». Aucun problème! Votre classe de fixtures est un service, vous pouvez donc utiliser l'injection de dépendance normale:

```php
// src/DataFixtures/AppFixtures.php
use Symfony\Component\Security\Core\Encoder\UserPasswordEncoderInterface;

class AppFixtures extends Fixture
{
    private $encoder;

    public function __construct(UserPasswordEncoderInterface $encoder)
    {
        $this->encoder = $encoder;
    }

    // ...
    public function load(ObjectManager $manager)
    {
        $user = new User();
        $user->setUsername('admin');

        $password = $this->encoder->encodePassword($user, 'pass_1234');
        $user->setPassword($password);

        $manager->persist($user);
        $manager->flush();
    }
}
```

Fractionner les “fixtures” en fichiers séparés

Dans la plupart des applications, créer tous vos « fixtures » dans une seule classe est très bien. Cette classe peut finir par être un peu longue, mais ça vaut le coup car avoir un seul fichier permet de garder les choses simples.

Si vous décidez de diviser vos « fixtures » en fichiers séparés, Symfony vous aide à résoudre les deux problèmes les plus courants: partager des objets entre « fixtures » et charger les « fixtures » dans l'ordre.

Partager des objets entre “fixtures”

Lorsque vous utilisez plusieurs fichiers de fixtures, vous pouvez réutiliser des objets PHP dans différents fichiers grâce aux références d'objet . Utilisez la méthode addReference() pour donner un nom à n'importe quel objet, puis utilisez la méthode getReference() pour obtenir exactement le même objet via son nom.

La seule mise en garde concernant l'utilisation de références est que les « fixtures » doivent être chargés dans un certain ordre (dans cet exemple, si les GroupFixtures sont chargés avant les UserFixtures, vous verrez une erreur). Par défaut, Doctrine charge les fichiers des « fixtures » dans l'ordre alphabétique, mais vous pouvez contrôler leur ordre comme expliqué dans la section suivante.

Partager des objets entre “fixtures”

```php
// src/DataFixtures/UserFixtures.php
// ...
class UserFixtures extends Fixture
{
    public const ADMIN_USER_REFERENCE = 'admin-user';

    public function load(ObjectManager $manager)
    {
        $userAdmin = new User('admin', 'pass_1234');
        $manager->persist($userAdmin);
        $manager->flush();

        // other fixtures can get this object using the UserFixtures::ADMIN_USER_REFERENCE constant
        $this->addReference(self::ADMIN_USER_REFERENCE, $userAdmin);
    }
}

// src/DataFixtures/GroupFixtures.php
// ...
class GroupFixtures extends Fixture
{
    public function load(ObjectManager $manager)
    {
        $userGroup = new Group('administrators');
        // this reference returns the User object created in UserFixtures
        $userGroup->addUser($this->getReference(UserFixtures::ADMIN_USER_REFERENCE));

        $manager->persist($userGroup);
        $manager->flush();
    }
}
```

Charger les fichiers de fixture dans l'ordre

Au lieu de définir l'ordre exact dans lequel tous les fichiers de « fixtures » doivent être chargés, Doctrine utilise une approche plus intelligente pour s'assurer que certains « fixtures » soient chargés avant d'autres. Implémentez l’interface « DependentFixtureInterface » et ajoutez une nouvelle méthode getDependencies() à votre classe « fixtures ». Cela renverra un tableau des classes « fixtures » qui doivent être chargées avant celle-ci:

Charger les fichiers de fixture dans l'ordre

```php
// src/DataFixtures/UserFixtures.php
namespace App\DataFixtures;

// ...
class UserFixtures extends Fixture
{
    public function load(ObjectManager $manager)
    {
        // ...
    }
}

// src/DataFixtures/GroupFixtures.php
namespace App\DataFixtures;
// ...
use App\DataFixtures\UserFixtures;
use Doctrine\Common\DataFixtures\DependentFixtureInterface;

class GroupFixtures extends Fixture implements DependentFixtureInterface
{
    public function load(ObjectManager $manager)
    {
        // ...
    }

    public function getDependencies()
    {
        return array(
            UserFixtures::class,
        );
    }
}
```

Groupe de “fixtures”: éxécution de certains “fixtures”

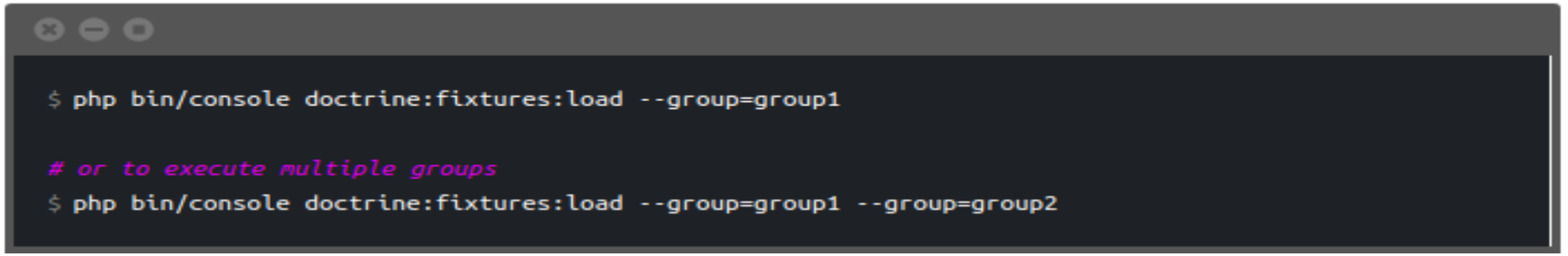
Par défaut, toutes vos classes de fixtures sont exécutées. Si vous ne souhaitez exécuter que certaines de vos classes de fixtures, vous pouvez les organiser en groupes.

La façon la plus simple d'organiser une classe de fixture en groupe est de faire implémenter votre fixture l’interface FixtureGroupInterface

```
1    // src/DataFixtures/UserFixtures.php
2
3  + use Doctrine\Bundle\FixturesBundle\FixtureGroupInterface;
4
5  - class UserFixtures extends Fixture
6  + class UserFixtures extends Fixture implements FixtureGroupInterface
7    {
8        // ...
9
10 +     public static function getGroups(): array
11 +     {
12 +         return ['group1', 'group2'];
13 +     }
14   }
```

Groupe de “fixtures”: éxécution de certains “fixtures”

Pour exécuter tous vos « fixtures » pour un groupe donné, passez l'option –group :



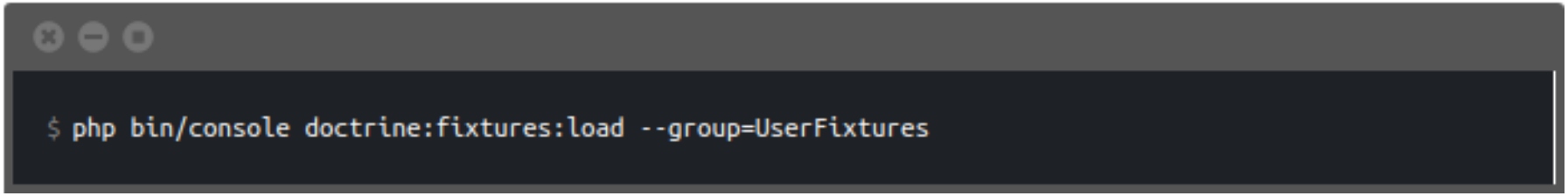
```
$ php bin/console doctrine:fixtures:load --group=group1

# or to execute multiple groups
$ php bin/console doctrine:fixtures:load --group=group1 --group=group2
```

- Alternativement, au lieu d'implémenter le FixtureGroupInterface, vous pouvez également étiqueter votre service avec doctrine.fixture.orm et ajouter un ensemble d'options supplémentaires à un groupe auquel votre « fixtures » doit appartenir.

Groupe de “fixtures”: éxécution de certains “fixtures”

Quels que soient les groupes définis dans le « fixtures » ou la définition de service, le chargeur de « fixtures » ajoute toujours le nom court de la classe en tant que groupe distinct afin que vous puissiez charger un seul « fixtures » à la fois. Dans l'exemple ci-dessus, vous pouvez charger le « fixtures » à l'aide du groupe UserFixtures :



Spécification du comportement de purge

Par défaut, toutes les données existantes étaient purgées à l'aide d'instruction DELETE FROM table. Si vous préférez utiliser l'instruction TRUNCATE table pour la purge, utilisez l'option

--purge-with-truncate

Si vous souhaitez exclure un ensemble de tables de la purge, par exemple parce que votre schéma contient des données semi-statiques préremplies, passez l'option --purge-exclusions. Spécifiez

--purge-exclusions plusieurs fois pour exclure plusieurs tables.

Vous pouvez également personnaliser davantage le comportement de purge et implémenter une purge personnalisée :

Spécification du comportement de purge

```php
// src/Purger/CustomPurger.php
namespace App\Purger;

use Doctrine\Common\DataFixtures\Purger\PurgerInterface;

// ...
class CustomPurger implements PurgerInterface
{
    public function purge() : void
    {
        // ...
    }
}

// src/Purger/CustomPurgerFactory.php
namespace App\Purger;
// ...
use Doctrine\Bundle\FixturesBundle\Purger\PurgerFactory;

class CustomPurgerFactory implements PurgerFactory
{
    public function createForEntityManager(?string $emName, EntityManagerInterface $em, array $excluded =
    {
        return new CustomPurger($em);
    }
}
```

Spécification du comportement de purge

L'étape suivante consiste à enregistrer notre purge personnalisée et à spécifier son alias.

YAML | XML | PHP

```yaml
# config/services.yaml
services:
    App\Purger\CustomPurgerFactory:
        tags:
            - { name: 'doctrine.fixtures.purger_factory', alias: 'my_purger' }
```

Avec l'option --purger, nous pouvons maintenant spécifier d'utiliser my_purger au lieu de la purge par defaut.

Merci

Bibliographie :

https://symfony.com/doc/master/bundles/DoctrineFixturesBundle/index.html