

Symfony : services

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en Programmation par contrainte (IA)
Ingénieur en Génie logiciel

`elmouelhi.achref@gmail.com`



1 Introduction

2 Service personnalisé

- Exemple
- Fichier `services.yaml`
- Injection de dépendance
- `ContainerInterface`
- `ContainerBagInterface`
- Clé arguments
- Débogage
- Alias
- Utilisation des interfaces

3 Quelques autres services Symfony prédéfinis

- Session
- Flash Messages

Symfony

Service

- Classe **PHP**
- Singleton
- Réalise une et une seule fonctionnalité
 - Envoi de mail
 - Manipulation d'une base de données...
- Accessible de partout dans notre code
- Injectable dans les classes où on a besoin de l'utiliser
- Pouvant utiliser un ou plusieurs autres services
- Ayant un identifiant = nom de la classe

Un conteneur de service

- Classe **PHP**
- Gestionnaire de services
 - Instanciation de services
 - Renvoi de services

Symfony

Un conteneur de service

- Classe **PHP**
- Gestionnaire de services
 - Instanciation de services
 - Renvoi de services

Pour accéder à un service, il faut passer par le conteneur de service.

Déroulement

- ❶ Le contrôleur demande un service au conteneur de service
- ❷ Le conteneur de service vérifie
 - Si ce service a déjà été instancié, alors il renvoie l'objet au contrôleur
 - Sinon, il l'instancie
 - Ensuite il l'enregistre
- ❸ Enfin il le renvoie

Symfony

Exemple de code d'un conteneur de service

```
class ServiceContainer
{
    private $services = array();

    public function getX()
    {
        if (isset($this->services['x'])) {
            return $this->services['x'];
        }

        $pdo = new PDO('mysql:host=localhost', ...);

        return $this->services['x'] = new X($pdo);
    }
}
```

Symfony

Exemple de code d'un conteneur de service

```
class ServiceContainer
{
    private $services = array();

    public function getX()
    {
        if (isset($this->services['x'])) {
            return $this->services['x'];
        }

        $pdo = new PDO('mysql:host=localhost', ...);

        return $this->services['x'] = new X($pdo);
    }
}
```

Le conteneur prend en charge toute la complexité (gestion de dépendance...)

Symfony

En **Symfony**

- Services prédéfinis : créés par **Symfony**.
- Services personnalisés : créés par le développeur.

Symfony

Créons un service AdresseEmailService **dans** src/Service

```
<?php

namespace App\Service;

class AdresseEmailService
{
    public function getAMfromNomPrenom($nom, $prenom)
    {
        $adresse = $nom . '.' . $prenom . '@symfony.fr';
        return $adresse;
    }
}
```

Pour utiliser le service dans l'action `index` du contrôleur `HomeController`

```
#[Route('/', name: "app_home")]
public function index(AdresseEMailService $adresseEMail): Response
{
    $personne = new Personne();
    $personne->setId(100);
    $personne->setNom("wick");
    $personne->setPrenom("john");
    $email = $adresseEMail->getAMfromNomPrenom($personne->getNom(),
        $personne->getPrenom());
    return $this->render('home/index.html.twig', [
        'email' => $email,
        'personne' => $personne
    ]);
}
```

Pour utiliser le service dans l'action `index` du contrôleur `HomeController`

```
#[Route('/', name: "app_home")]
public function index(AdresseEmailService $adresseEmail): Response
{
    $personne = new Personne();
    $personne->setId(100);
    $personne->setNom("wick");
    $personne->setPrenom("john");
    $email = $adresseEmail->getAMfromNomPrenom($personne->getNom(),
        $personne->getPrenom());
    return $this->render('home/index.html.twig', [
        'email' => $email,
        'personne' => $personne
    ]);
}
```

Les `use` nécessaires

```
use App\Entity\Personne;
use App\Service\AdresseEmail;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
```

Symfony

Dans la vue

```
{% extends 'base.html.twig' %}
```

```
{% block title %}Home
```

```
{% endblock %}
```

```
{% block body %}
```

```
Bonjour {{ personne.nom }}.
```

```
Votre adresse email Symfony est : {{ email }}
```

```
{% endblock %}
```

Symfony

Question

Pourquoi **Symfony** a considéré `AdresseEmailService` comme service ?

© Achref EL MOUELHI ©

Symfony

Question

Pourquoi **Symfony** a considéré AdresseEmailService comme service ?

La réponse est donnée par cette partie de `config/services.yaml`

```
services:
    # default configuration for services in *this* file
    _defaults:
        autowire: true      # Automatically injects dependencies in
                             your services.
        autoconfigure: true # Automatically registers your services as
                             commands, event subscribers, etc.

    # makes classes in src/ available to be used as services
    # this creates a service per class whose id is the fully-qualified
    # class name
    App\:
        resource: '../src/'
```

Symfony

Exercice

- Créer un service `CalculService`.
- Injecter `CalculService` dans l'action `apply` de `CalculController`.
- Déplacer le contenu de l'action `apply` de `CalculController` dans la méthode `getResultat(string $op, int $valeur1, int $valeur2)` de `CalculService`.
- L'action `apply` doit récupérer les paramètres, utiliser le service pour calculer le résultat et le passer à la vue pour affichage.

Un service peut utiliser un autre

```
namespace App\Service;

use Psr\Log\LoggerInterface;

class AdresseEmailService
{
    private $logger;

    public function __construct(LoggerInterface $logger)
    {
        $this->logger = $logger;
    }

    public function getAMfromNomPrenom($nom, $prenom)
    {
        $adresse = $nom . '.' . $prenom . '@symfony.fr';
        $this->logger->info("$adresse a été générée");
        return $adresse;
    }
}
```

Un service peut utiliser un autre

```
namespace App\Service;

use Psr\Log\LoggerInterface;

class AdresseEmailService
{
    private $logger;

    public function __construct(LoggerInterface $logger)
    {
        $this->logger = $logger;
    }

    public function getAMfromNomPrenom($nom, $prenom)
    {
        $adresse = $nom . '.' . $prenom . '@symfony.fr';
        $this->logger->info("$adresse a été générée");
        return $adresse;
    }
}
```

Allez à localhost:8000/home puis vérifiez dans var/log/dev.log l'ajout de la ligne APP wick.john@symfony.fr a été générée [] [].

Symfony

Injection de dépendance

- Le constructeur de notre service prend en paramètre `LoggerInterface`.
- Lorsque le conteneur instancie le service la première fois, il instancie aussi une instance de `logger`.

Symfony

On peut aussi définir l'extension comme paramètre dans la section `parameters` de `config/services.yaml`

```
# This file is the entry point to configure your own
services.

# Files in the packages/ subdirectory configure your
dependencies.

# Put parameters here that don't need to change on
each machine where the app is deployed
# https://symfony.com/doc/current/best_practices/
# configuration.html#application-related-
# configuration

parameters:
    nom: 'wick'
    extension: 'fr'
```

Symfony

Dans AdresseEmailService, utilisons ContainerInterface (dépréciée depuis Symfony 5.1 et supprimée depuis Symfony 6) pour récupérer le paramètre

```
namespace App\Service;

use Psr\Log\LoggerInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

class AdresseEmailService
{
    private $logger;
    private $container;

    public function __construct(LoggerInterface $logger, ContainerInterface $container)
    {
        $this->logger = $logger;
        $this->container = $container;
    }

    public function getAMfromNomPrenom($nom, $prenom)
    {
        if ($this->container->hasParameter('extension')) {
            $extension = $this->container->getParameter('extension');
            $adresse = $nom . '.' . $prenom . '@symfony.' . $extension;
        } else {
            $adresse = $nom . '.' . $prenom . '@symfony.gouv.fr';
        }
        $this->logger->info("$adresse a été générée");
        return $adresse;
    }
}
```

Symfony

Nous pouvons aussi utiliser ContainerBagInterface

```
namespace App\Service;

use Psr\Log\LoggerInterface;
use Symfony\Component\DependencyInjection\ParameterBag\ContainerBagInterface;

class AdresseEmailService
{
    private $logger;
    private $params;

    public function __construct(LoggerInterface $logger, ContainerBagInterface $params)
    {
        $this->logger = $logger;
        $this->params = $params;
    }

    public function getAMfromNomPrenom($nom, $prenom)
    {
        if ($this->params->has('extension')) {
            $extension = $this->params->get('extension');
            $adresse = $nom . '.' . $prenom . '@symfony.' . $extension;
        } else {
            $adresse = $nom . '.' . $prenom . '@symfony.gouv.fr';
        }
        $this->logger->info("$adresse a été générée");
        return $adresse;
    }
}
```

Symfony

Dans la section `services` de `config/services.yaml`, déclarons le service et précisons ce qu'il lui faut comme arguments (il faut respecter l'ordre défini dans le constructeur)

```
parameters:
    nom: 'wick'
    extension: 'fr'

services:
    # contenu précédent

    App\Service\AdresseEmailService:
        arguments: ['@logger', '%extension%']
```

© Achref

Symfony

Dans la section `services` de `config/services.yaml`, déclarons le service et précisons ce qu'il lui faut comme arguments (il faut respecter l'ordre défini dans le constructeur)

```
parameters:
    nom: 'wick'
    extension: 'fr'

services:
    # contenu précédent

    App\Service\AdresseEmailService:
        arguments: ['@logger', '%extension%']
```

Explication

- `@` permet de faire référence à un service.
- `%parametre%` permet de faire référence à un paramètre défini dans la section `parameters`.

Symfony

Avant de tester, modifions le service en injectant l'extension dans son constructeur

```
namespace App\Service;

use Psr\Log\LoggerInterface;

class AdresseEmailService
{
    private $logger;
    private $extension;

    public function __construct(LoggerInterface $logger, string $extension)
    {
        $this->logger = $logger;
        $this->extension = $extension;
    }

    public function getAMfromNomPrenom($nom, $prenom)
    {
        $adresse = $nom . '.' . $prenom . '@symfony.' . $this->extension;
        $this->logger->info("$adresse a été générée");
        return $adresse;
    }
}
```

Symfony

Avec PHP 8 (utilisé par Symfony 6), on peut simplifier

```
namespace App\Service;

use Psr\Log\LoggerInterface;

class AdresseEmailService
{
    public function __construct(private LoggerInterface $logger, private string
        $extension)
    {
    }

    public function getAMfromNomPrenom($nom, $prenom)
    {
        $adresse = $nom . '.' . $prenom . '@symfony.' . $this->extension;
        $this->logger->info("$adresse a été générée");
        return $adresse;
    }
}
```

Symfony

Pour consulter la liste des services disponibles

```
php bin/console debug:container
```

Symfony

Depuis la version 3.4, une nouvelle commande a été introduite qui permet de réaliser le même objectif, pour les services ayant un alias, avec un peu plus de détails

```
php bin/console debug:autowiring
```

© Achref EL MOUELHI ©

Symfony

Depuis la version 3.4, une nouvelle commande a été introduite qui permet de réaliser le même objectif, pour les services ayant un alias, avec un peu plus de détails

```
php bin/console debug:autowiring
```

Une partie du résultat

Autowirable Types

=====

The following classes & interfaces can be used as **type**-hints when autowiring:

App\Kernel (kernel)

Interface **for** annotation readers.

Doctrine\Common\Annotations\Reader (annotations.cached_reader)

Doctrine\Common\Persistence\ManagerRegistry (doctrine)

...

Symfony

Pour consulter la liste de tous les services avec ou sans alias

```
php bin/console debug:autowiring --all
```

© Achref EL MOUELHI ©

Symfony

Pour consulter la liste de tous les services avec ou sans alias

```
php bin/console debug:autowiring --all
```

Une partie du résultat

Autowirable Types

=====

The following classes & interfaces can be used as **type**-hints when autowiring:

App\Controller\HomeController

App\Controller\VehiculeController

App\Kernel (kernel)

App\Service\AdresseEmailService

...

Pour consulter la liste des services qui correspondent à un motif de recherche ([mail ici](#))

```
php bin/console debug:autowiring mail --all
```

© Achref EL MOUELHI ©

Pour consulter la liste des services qui correspondent à un motif de recherche (mail ici)

```
php bin/console debug:autowiring mail --all
```

Une partie du résultat

Autowirable Types

=====

The following classes & interfaces can be used as **type**-hints when autowiring:

(only showing classes/interfaces matching mail)

App\Service\AdresseEmailService

Interface **for** mailers able to send emails synchronous **and/or** asynchronous.

Symfony\Component\Mailer\MailerInterface (mailer.mailer)

Interface **for all** mailer transports.

Symfony\Component\Mailer\Transport\TransportInterface (mailer.default_transport)

Symfony

Pour consulter la liste des paramètres disponibles

```
php bin/console debug:container --parameters
```

Symfony

Alias

- Un service peut avoir un alias.
- Un alias peut aider à trouver le service lorsqu'on le cherche avec les lignes de commande.
- Plus besoin d'ajouter l'option `--all`.

© Achref

Symfony

Alias

- Un service peut avoir un alias.
- Un alias peut aider à trouver le service lorsqu'on le cherche avec les lignes de commande.
- Plus besoin d'ajouter l'option `--all`.

Deux étapes pour créer correctement un alias

- Déclarer un identifiant du service.
- Associer le service à l'alias.

Symfony

Définissons l'alias `app.mail.generator` dans la section `services` de `config/services.yaml`

```
services:
    # contenu précédent de la section services
    app.mail.generator:
        class: App\Service\AdresseEmailService
        arguments: ['@logger', '%extension%']

    App\Service\AdresseEmailService: '@app.mail.generator'
```

© Achref EL M...

Symfony

Définissons l'alias `app.mail.generator` dans la section `services` de `config/services.yaml`

```
services:
    # contenu précédent de la section services
    app.mail.generator:
        class: App\Service\AdresseEmailService
        arguments: ['@logger', '%extension%']

    App\Service\AdresseEmailService: '@app.mail.generator'
```

Remarque

- La première partie permet de déclarer un nouvel identifiant (`app.mail.generator`) pour le service `App\Service\AdresseEmailService`.
- La dernière ligne permet de déclarer `app.mail.generator` comme Alias de la classe `App\Service\AdresseEmailService`.

Symfony

Constats

- Vérifier que le service est accessible.
- Vérifier que le service apparaît lorsqu'on lance la commande `php bin/console debug:autowiring` sans l'option `--all`.

Symfony

Considérons l'interface `EmailInterface` à définir dans `src\Util`

```
<?php

namespace App\Util;

interface EmailInterface
{
    public function getAMfromNomPrenom($nom, $prenom): string;
}
```

Symfony

Faisons hériter notre service de cette nouvelle interface

```
namespace App\Service;

use App\Util\EmailInterface;
use Psr\Log\LoggerInterface;

class AdresseEmailService implements EmailInterface
{
    public function __construct(private LoggerInterface $logger, private string $extension)
    {
    }

    public function getAMfromNomPrenom($nom, $prenom): string
    {
        $adresse = $nom . '.' . $prenom . '@symfony.' . $this->extension;
        $this->logger->info("$adresse a été générée");
        return $adresse;
    }
}
```

Symfony

Utilisons plutôt l'interface dans l'action `index` de `HomeController`

```
namespace App\Controller;

use App\Entity\Personne;
use App\Util\EmailInterface;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class HomeController extends AbstractController
{
    #[Route('/home', name: "app_home")]
    public function index(EmailInterface $adresseEmail): Response
    {
        $p = new Personne();
        $p->setId(100);
        $p->setNom("wick");
        $p->setPrenom("john");
        $email = $adresseEmail->getAMfromNomPrenom($p->getNom(), $p->getPrenom());
        return $this->render('home/index.html.twig', [
            'email' => $email,
            'personne' => $p
        ]);
    }
}
```

Symfony

Modifions l'alias `app.mail.generator` **dans la section** `services` **de** `config/services.yaml`

```
services:
    # contenu précédent de la section services
    app.mail.generator:
        class: App\Service\AdresseEmailService
        arguments: ['@logger', '%extension%']

    App\Util\EmailInterface: '@app.mail.generator'
```

Symfony

Constats

- Vérifier que le service est accessible.
- Vérifier que le service apparaît lorsqu'on lance la commande `php bin/console debug:autowiring` sans l'option `--all`.

Symfony

Services prédéfinis : on en a déjà utilisé au moins deux

- request
- logger

© Achref EL MOUËZ

Symfony

Services prédéfinis : on en a déjà utilisé au moins deux

- request
- logger

Il existe plusieurs autres tels que

- session
- mailer
- ...

Symfony

Le service `session`

- Les outils de session sont également intégrés dans un service (`SessionInterface`).
- Utiliser la méthode `set` pour ajouter une variable dans la session et `get` pour récupérer.
- Dans les vues : `{{ app.session.get('nom_variable') }}`

Exemple d'utilisation de session : code de HomeController

```
namespace App\Controller;

use App\Entity\Personne;
use Symfony\Component\HttpFoundation\Session\SessionInterface;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class HomeController extends AbstractController
{
    #[Route('/home', name: "app_home")]
    public function index(SessionInterface $session): Response
    {
        $personne = new Personne();
        $personne->setId(100);
        $personne->setNom("wick");
        $personne->setPrenom("john");
        $session->set('personne', $personne);
        return $this->render('home/index.html.twig', []);
    }
}
```

Symfony

Code de la vue associée à la méthode `index` de `HomeController`

```
{% extends 'base.html.twig' %}

{% block title %}Home{% endblock %}

{% block body %}

    Bonjour      {{ app.session.get('personne').nom }}.
    <a href="{{ url('app_vehicule') }}">Véhicule</a>

{% endblock %}
```

Symfony

Code de VehiculeController

```
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Session\SessionInterface;

class VehiculeController extends AbstractController
{
    #[Route('/vehicule', name: 'app_vehicule')]
    public function index(SessionInterface $session): Response
    {
        $personne = $session->get('personne');
        return $this->render('vehicule/index.html.twig', [
            'personne' => $personne
        ]);
    }
}
```

Symfony

Code de la vue associée à la méthode `index` de `VehiculeController`

```
{% extends 'base.html.twig' %}

{% block title %}Vehicule{% endblock %}

{% block body %}

    Bonjour {{ app.session.get('personne').nom }}
    {{ personne.prenom }}.
    <a href="{{ url('app_home') }}">Accueil</a>

{% endblock %}
```

Symfony

Flash Messages

- Un messages flash est une variable de session qui ne dure que le temps d'une seule page
- Une fois le message flash est affiché, il sera détruit de la session (donc il disparaîtra après actualisation ou changement de vue)

Modifions le contrôleur HomeController

```
namespace App\Controller;

use App\Entity\Personne;
use Symfony\Component\HttpFoundation\Session\SessionInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;

class HomeController extends AbstractController
{
    #[Route('/home', name: "app_home")]
    public function index(SessionInterface $session)
    {
        $personne = new Personne(100, "wick", "john");
        $this->addFlash(
            'version',
            'Symfony 5'
        );
        $session->set('personne', $personne);
        return $this->render('home/index.html.twig', []);
    }
}
```

Symfony

Dans la vue associée à la méthode `index` du contrôleur `HomeController`

```
{% extends 'base.html.twig' %}

{% block title %}Home{% endblock %}

{% block body %}

    Bonjour
    {{ app.session.get('personne').nom }}.

    {% for message in app.flashes('version') %}
        {{ message }}
    {% endfor %}

    <a href="{{ url('app_vehicule') }}">Véhicule</a>

{% endblock %}
```

Symfony

Dans la vue associée à la méthode `index` du contrôleur `VehiculeController`

```
{% extends 'base.html.twig' %}

{% block title %}Vehicule{% endblock %}

{% block body %}

    Bonjour {{ app.session.get('personne').nom }}
    {{ personne.prenom }}.

    {% for message in app.flashes('version') %}
        {{ message }}
    {% endfor %}
    <a href="{{ url('app_home') }}">Accueil</a>

{% endblock %}
```

Symfony

Pour tester

- Allez sur la route `home` et vérifiez que le flash s'affiche
- Ensuite, allez sur `vehicule` et vérifiez que le flash ne s'affiche plus.