Desarrollo de aplicaciones con Symfony 4

Servicios

Servicios

Nuestra aplicación está llena de objetos útiles (por ejemplo un objeto "Mailer" nos servirá para enviar correos). Casi todo lo que su aplicación "hace" en realidad lo hace uno de estos objetos. ¡Y cada vez que instalas un nuevo paquete, se están instalando nuevos “objetos útiles”!

En Symfony, estos objetos útiles se denominan servicios y cada servicio vive dentro de un objeto muy especial llamado **Service Container**. El contenedor le permite centralizar la forma en que se construyen los objetos. Te hace la vida más fácil, promueve una arquitectura sólida y es súper rápido.

Servicios

Veamos un ejemplo:

```
/**
 * @Route("/products")
 */
public function list(LoggerInterface $logger)
{
    $logger->info('Look! I just used a service');

    // ...
}
```

AvaiBook BOOKING

Symfony

Servicios

¿Qué otros servicios están disponibles? Puedes verlo ejecutando:

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

Creando Servicios

```
// src/Service/MessageGenerator.php
namespace App\Service;
class MessageGenerator {
    public function getHappyMessage() {
        $messages = [
            'You did it! You updated the system! Amazing!',
            'That was one of the coolest updates I\'ve seen all day!',
            'Great work! Keep going!',
        ];
        $index = array_rand($messages);
        return $messages[$index];
    }
}
```

Creando Servicios

```
use App\Service\MessageGenerator;

public function new(MessageGenerator $messageGenerator)
{
    $message = $messageGenerator->getHappyMessage();
    $this->addFlash('success', $message);
    // ...
}
```

Creando Servicios

```
use App\Service\MessageGenerator;

public function new(MessageGenerator $messageGenerator)
{
    $message = $messageGenerator->getHappyMessage();
    $this->addFlash('success', $message);
    // ...
}
```

Servicios

¿Cómo funciona exactamente?

Veamos el archivo # config/services.yaml

```
services:
    _defaults:
        autowire: true      # Automatically injects dependencies in your services.
        autoconfigure: true # Automatically registers your services as commands, event subscribers, etc
App\:
        resource: '../src/*'
        exclude: '../src/{Entity,Migrations,Tests,Kernel.php}'
```

AvaiBook BOOKING

Symfony

Inyectando otros servicios en nuestros servicios

¡Muy sencillo!

```
class MessageGenerator
{
    private $logger;
    public function __construct(LoggerInterface $logger)
    {
        $this->logger = $logger;
    }
```

AvaiBook BOOKING

Symfony

Inyectando otros servicios en nuestros servicios

```
private $messageGenerator;
private $mailer;

public function __construct(MessageGenerator $messageGenerator,
\Swift_Mailer $mailer)
{
    $this->messageGenerator = $messageGenerator;
    $this->mailer = $mailer;
}
```

Pasando argumentos de forma manual

```
private $adminEmail;


public function __construct(MessageGenerator $messageGenerator,
\Swift_Mailer $mailer, $adminEmail)
{
   // ...
   $this->adminEmail = $adminEmail;
}
```

Pasando argumentos de forma manual

```
# config/services.yaml
services:
    # ...
    # same as before
    App\:
        resource: '../src/*'
        exclude: '../src/{Entity,Migrations,Tests}'

    # explicitly configure the service
    App\Service\MessageGenerator:
        arguments:
            $adminEmail: 'manager@example.com'
```

Usando parámetros

```
# config/services.yaml
parameters:
    admin_email: manager@example.com

services:
    # ...

     App\Service\MessageGenerator:
        arguments:
            $adminEmail: '%admin_email%'
```

AvaiBook BOOKING

Symfony

Usando parámetros

También podremos acceder a estos parámetros en nuestro controlador:

```
public function new()
{
    // ...
    // this shortcut ONLY works if you extend the base AbstractController
    $adminEmail = $this->getParameter('admin_email');

    // this is the equivalent code of the previous shortcut:
    // $adminEmail = $this->container->get('parameter_bag')->get('admin_email');
}
```

Pasando servicios de forma manual

```
# config/services.yaml
services:
    # ... same code as before

    # explicitly configure the service
    App\Service\MessageGenerator:
        arguments:
            # the '@' symbol is important: that's what tells the container
            # you want to pass the *service* whose id is 'monolog.logger.request',
            # and not just the *string* 'monolog.logger.request'
            $logger: '@monolog.logger.request'
```

Argumentos por nombre o tipo

Veámoslo mejor en:

https://symfony.com/doc/current/service_container.html#binding-arguments-by-name-or-type

Obteniendo el container como servicio

Si algún servicio o controlador necesita muchos parámetros de contenedor, hay una alternativa más fácil a vincularlos a todos con la opción services._defaults.bind.

```
use Symfony\Component\DependencyInjection\ParameterBag\ParameterBagInterface;
    private $params;
    public function __construct(ParameterBagInterface $params)
    {
        $this->params = $params;
    }
//…
$sender = $this->params->get('mailer_sender');
```

AvaiBook BOOKING

Symfony

Public vs Private

Por defecto en symfony cada servicio definido público: false de forma predeterminada.

¿Qué significa esto? Cuando un servicio es público, puede acceder a él directamente desde el objeto contenedor, al que se puede acceder desde cualquier controlador que extienda el Controlador:

Public vs Private

```
use App\Service\MessageGenerator;
// ...
public function new()
{
    // there IS a public "logger" service in the container
    $logger = $this->container->get('logger');

    // this will NOT work: MessageGenerator is a private service
    $generator = $this->container->get(MessageGenerator::class);
}
```

Public vs Private

¿Como hacemos nuestro servicio público?

```
# config/services.yaml
services:
    # ... same code as before

    # explicitly configure the service
    App\Service\MessageGenerator:
        public: true
```

¿Preguntas?