Desarrollo de aplicaciones con Symfony 4

Extendiendo twig

Extendiendo twig

Ya vimos en temas anteriores que twig trae por defecto una serie de filtros y funciones para facilitarnos el trabajo:

Filtros: https://twig.symfony.com/doc/2.x/filters/index.html

Funciones: https://twig.symfony.com/doc/2.x/functions/index.html

Extendiendo twig

¿Que diferencia hay entre una función y un filtro?

Un **filtro** es una forma de transformar los datos para mostrarlos.

```
{{ issue.tags|join(‘,’) }}
```

Una **función** se usa cuando necesitas calcular cosas para representar el resultado.

```
{{ dump(issue) }}
```

Extendiendo twig

Por lo general, escribe una función cuando necesita hacer cosas más pesadas que simplemente transformar la visualización de un contenido de una manera simple.

Extendiendo twig: Nuestro primer filtro

Supongamos que desea crear un nuevo filtro llamado price que formatee un número en dinero:

```
{{ product.price | price }}
```

```
{# y que se le puedan pasar argumentos #}

{{ product.price | price(2, ',', '.') }}
```

Extendiendo twig: Nuestro primer filtro

Crearíamos una clase que extienda AbstractExtension

```
// src/Twig/PriceExtension.php

namespace App\Twig;

use Twig\Extension\AbstractExtension;

use Twig\TwigFilter;

class PriceExtension extends AbstractExtension

{

 //...

}
```

Extendiendo twig: Nuestro primer filtro

```
public function getFilters()
{
    return [
        new TwigFilter('price', [$this, 'formatPrice']),
    ];
}
```

AvaiBook BOOKING

Symfony

Extendiendo twig: Nuestro primer filtro

```
public function formatPrice($number, $decimals = 0, $decPoint = '.', $thousandsSep = ',')

{

    $price = number_format($number, $decimals, $decPoint, $thousandsSep);

    $price = '$'.$price;


    return $price;

}
```

AvaiBook BOOKING

Symfony

Extendiendo twig: Nuestro primer filtro

<< Veámoslo en phpStorm>>

Extendiendo twig: Nuestra primera función

En lugar de implementar getFilters(), haremos lo propio con getFunctions()

```
public function getFunctions()
    {
        return [
            new TwigFunction('area', [$this, 'calculateArea']),
        ];
    }
```

AvaiBook BOOKING

Symfony

Extendiendo twig: Nuestra primera función

```
public function calculateArea(int $width, int $length)
{
    return $width * $length;
}
```

AvaiBook BOOKING

Symfony

Extendiendo twig: Nuestra primera función

<< Veámoslo en phpStorm>>

Extendiendo twig

Todo esto ha funcionado correctamente al tener:

_defaults:

    autowire: true

    autoconfigure: true

Si lo tuviésemos desactivado, deberíamos registrar nuestro servicio y etiquetarlo como twig.extension.

Extendiendo twig

Para comprobar que nuestro filtro ha sido activado correctamente tenemos:

```
$ php bin/console debug:twig --filter=price
```

o

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

Creando Lazy-Loaded Twig Extensions

Incluir el código de los filtros / funciones personalizados en la clase de la extensión Twig es la forma más sencilla de crear extensiones. Sin embargo, Twig debe inicializar todas las extensiones antes de representar cualquier plantilla, incluso si la plantilla no usa una extensión.

Creando Lazy-Loaded Twig Extensions

Si las extensiones no definen dependencias (es decir, si no inyectamos servicios en ellas), el rendimiento no se verá afectado. Sin embargo, si las extensiones definen muchas dependencias complejas (por ejemplo, aquellas que realizan conexiones de base de datos), la pérdida de rendimiento puede ser significativa.

Creando Lazy-Loaded Twig Extensions

Por este motivo Twig permite desacoplar la definición de extensión de su implementación. Siguiendo el mismo ejemplo que antes, el primer cambio consistiría en eliminar el método formatPrice () de la extensión y actualizar el código PHP definido en getFilters ():

```
new TwigFilter('price', [AppRuntime::class, 'formatPrice']),
```

Creando Lazy-Loaded Twig Extensions

```
// src/Twig/AppRuntime.php

namespace App\Twig;

use Twig\Extension\RuntimeExtensionInterface;

class AppRuntime implements RuntimeExtensionInterface

{

    //...

}
```

AvaiBook BOOKING

Symfony

Creando Lazy-Loaded Twig Extensions

```
public function __construct()

{

    // Si se necesita inyectar dependencias.

}
```

AvaiBook BOOKING

Symfony

Creando Lazy-Loaded Twig Extensions

El método construct() si se necesita inyectar dependencias:

```
public function __construct()
{
    // Si se necesita inyectar dependencias.
}
```

AvaiBook BOOKING

Symfony

Creando Lazy-Loaded Twig Extensions

Y finalmente el método formatPrice:

```
public function formatPrice(...)
{
    // ...
}
```

AvaiBook BOOKING

Symfony

Extendiento twig

Más información:

https://symfony.com/doc/current/templating/twig_extension.html

¿Preguntas?