Desarrollo de aplicaciones con Symfony 4

Eventos

Eventos

Durante la ejecución de una aplicación Symfony, se activan muchas notificaciones de eventos. Nuestra aplicación puede escuchar estas notificaciones y responderlas ejecutando código personalizado.

Symfony activa varios eventos relacionados con el kernel al procesar la solicitud HTTP, los paquetes de terceros también pueden enviar eventos, e incluso puede enviar eventos personalizados desde nuestro código.

Creando un Event Listener

<< Ver en: >>

https://symfony.com/doc/current/event_dispatcher.html#creating-an-event-listener

Creando un Event Suscriber

Otra forma de escuchar eventos es a través de un event suscriber, que es una clase que define uno o más métodos que escuchan uno o varios eventos. La principal diferencia con los events listener es que los event suscriber siempre saben qué eventos están escuchando.

Creando un Event Suscriber

<< Ver en: >>

https://symfony.com/doc/current/event_dispatcher.html#creating-an-event-subscriber

¿Listeners o Subcribers?

Los listeners y los subscribers pueden usarse en una aplicación indistintamente. La decisión de usar uno de los dos es normalmente a gusto del desarrollador. Sin embargo, hay algunas ventajas para cada uno:

Los subscribers son más fáciles de reutilizar porque los eventos se conocen en la clase en lugar de en la definición del service. Esta es la razón por la que Symfony utiliza subscribers internamente.

Los listeners son más flexibles porque los bundles pueden activarlos o desactivarlos incondicionalmente dependiendo de algún valor de configuración.

Listar eventos disponibles

Para mostrar todos los eventos y sus oyentes, ejecute:

```
$ php bin/console debug:event-dispatcher
```

Y para mostrar uno en particular:

```
$ php bin/console debug:event-dispatcher kernel.exception
```

Custom Events: El componente EventDispacher

El componente EventDispatcher proporciona herramientas que permiten que los componentes de su aplicación se comuniquen entre sí mediante el envío de eventos y escuchándolos.

Es decir, permite crear tus propios eventos.

Custom Events: Event

Es importante saber que cada evento se identifica con un nombre único (por ejemplo, kernel.response). También se crea una instancia de evento y se pasa a todos los listeners y subscribers. Como veremos más adelante, el objeto Event en sí mismo a menudo contiene datos sobre el evento que se está enviando.

Custom Events: Event

El nombre único del evento puede ser cualquier cadena, pero es recomendable seguir algunas convenciones:

- Utilizar sólo letras minúsculas, números, puntos (.) y guiones bajos (_).
- Prefije los nombres con un espacio de nombres seguido de un punto (por ejemplo, issue., user. *);
- Termine los nombres con un verbo que indique qué acción se ha realizado (por ejemplo, issue.saved).

Custom Events: El Dispacher

El dispacher es el objeto central del EventDispacher. El es el encargado de lanzar los eventos y notificar los listeners y suscribers

```
use Symfony\Component\EventDispatcher\EventDispatcher;
```

```
$dispatcher = new EventDispatcher();
```

Custom Events: Creando el evento

```php
<?php
namespace App\Event;
use Symfony\Component\EventDispatcher\Event;
use App\Entity\Issue;
class IssueEvent extends Event{
    const SAVED = 'issue.save';
    private $issue;
    public function __construct(Issue $issue)  {
        $this->issue = $issue;
    }
    public function getIssue() {
        return $this->issue;
    }
}
```

Custom Events: Lanzando el evento

El la parte de nuestro código donde queramos lanzar el evento que acabamos de crear, simplemente lo instanciamos y lo lanzamos

```
use Symfony\Component\EventDispatcher\EventDispatcher;
```

```
$event = new IssueEvent($issue);
$dispatcher->dispatch(IssueEvent::SAVED, $event);
```

Custom Events: Creando el Subscriber

```
class IssueSubscriber implements EventSubscriberInterface
{
    public static function getSubscribedEvents()
    {
        return [
                IssueEvent::SAVED => 'sendMail',
        ];
    }
    public function sendMail(IssueEvent $event)
    {
        // ...
    }
}
```

AvaiBook BOOKING

Symfony

Custom Events: Detener el flujo / propagación del evento

En algunos casos, puede tener sentido que un oyente evite que se llame a otros oyentes. En otras palabras, el oyente debe poder decirle al operador que detenga toda propagación del evento a los oyentes futuros (es decir, que no notifique a más oyentes). Esto se puede lograr desde un oyente a través del método stopPropagation ():

```
$event->stopPropagation();
```

Custom Events: Más información

https://symfony.com/doc/current/components/event_dispatcher.html

¿Preguntas?