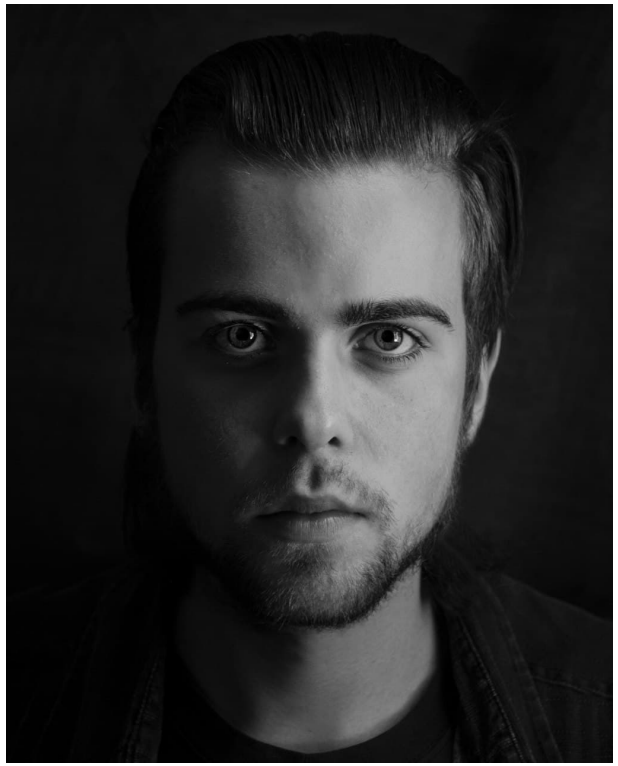
Веб разработка на PHP Symfony

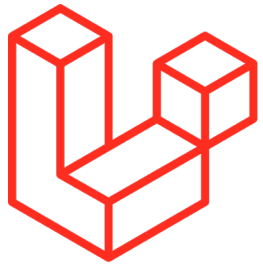
Event Listener

Introduction



Демьян Костельный
Middle PHP Developer

demian-kostelny-613b90151



Тема урока

Event Listener

План урока

1. Понятие Event
2. Разница Event Listener и Event Subscriber
3. Пример использования
4. Дебаг ивентов
5. Приоритет

Понятие Event

Понятие Event

В Symfony уже есть реализованная логика событий, т.е. в приложении есть отдельные системные события, которые отвечают за разные компоненты. К примеру есть **kernel.exception**, которое отвечает за ошибки в системе.

Разница Event Listener и Event Subscriber

Разница Event Listener и Event Subscriber

В Symfony есть два типа классов в которые можно регистрировать события для приложения. Это Event Listener и Event Subscriber. Какая же разница между ними?

В Event Listener можно регистрировать любые события, как и в Event Subscriber - разница лишь в том, что Event Subsciber знает с какими событиями он постоянно работает на данный момент.

Пример использования

Пример использования

Для начала рассмотрим пример использования Event Listener. Для этого сначала создаётся файл в папке `src/EventListener/ExceptionListener.php`:

```php
namespace App\EventListener;

use Symfony\Component\HttpKernel\Event\GetResponseForExceptionEvent;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpKernel\Exception\HttpExceptionInterface;

class ExceptionListener // Создаём класс нашего слушателя событий (Event Listener)
{

    public function onKernelException(GetResponseForExceptionEvent $event)
    {
        // А здесь уже пропишем код для того чтобы получать события
        // через переменную $event. Также можете заметить что данный
        // Event Listener будет для обработки ошибок Kernel
    }

}
```

Пример использования

Дальше вносим код в функцию для получения событий:

```php
public function onKernelException(GetResponseForExceptionEvent $event)
{

    $exception = $event->getException(); // Получаем ошибку из переменной события
    $message = sprintf(
        'Ошибка: %s с кодом %s',
        $exception->getMessage(), // Достаём сообщение ошибки в сообщение
        $exception->getCode() // и также достаём код ошибки
    );

    $response = new Response(); // Создаём объект ответа от сервера
    $response->setContet($message); // Вносим сообщение об ошибке в переменную ошибки

    // Дальше делаем проверку ошибки является ли он объектом класса HttpExceptionInterface,
    // ошибка тогда будет содержать код статуса и детали HTTP заголовка
    if ( $exception instanceof HttpExceptionInterface ) {
        $response->setStatusCode($exception->getStatusCode()); // Достаём код статуса HTTP
        $response->headers->replace($exception->getHeaders()); // Достаём заголовки HTTP из
        ошибки
    } else {
        $response->setStatusCode(Response::HTTP_INTERNAL_SERVER_ERROR);
        // Если ошибка другого типа - то выводим 505
    }

    $event->setResponse($response); // Отправляем объект ответа событию

}
```

Пример использования

После того как мы создали класс, нужно его зарегистрировать в файле `config/services.yaml`:

```yaml
services:
    App\EventListener\ExceptionListener: # Заметьте что регистрируем класс как сервис
        tags:
            - { name: kernel.event_listener, event: kernel.exception }
```

Пример использования

Теперь давайте рассмотрим пример с использованием Event Subscriber. Для начала давайте создадим файл в папке `src/EventSubscriber/EventSubscriber.php`:

```php
namespace App\EventSubscriber;

use Symfony\Component\EventDispatcher\EventSubscriberInterface;
use Symfony\Component\HttpKernel\Event\GetResponseForExceptionEvent;
use Symfony\Component\HttpKernel\KernelEvents;

class ExceptionSubscriber implements EventSubscriberInterface
{
    public static function getSubscribedEvents()
    {
        // Функция вёрнет события, а также их функции с приоритетами которые в числовом значении
        return array(
            KernelEvents::EXCEPTION => array(
                array('processException', 10),
                array('logException', 0),
                array('notifyException', -10),
            )
        );
    }
```

Пример использования

Также в самом классе находятся и все другие функции, которые используют события:

```php
public function processException()
{
    // ...
}

public function logException()
{
    // ...
}

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

Пример использования

И вот, наш Event Subscriber готов. Следующим шагом является настроить его в `config/services.yaml`:

```yaml
services:
    App\EventSubscriber\ExceptionSubscriber:
        tags:
            - { name: kernel.event_subscriber, event: kernel.exception }
```

Дебаг ивентов

Дебаг ивентов

Если нужно узнать какие зарегистрированы события в системе, event listeners и event subscribers - то нужно выполнить следующую команду в консоли:

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

Если нужно узнать какие event listeners и event subscribers зарегистрированы на какое-то отдельное событие, то можно это посмотреть через следующую команду:

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

Приоритеты

Приоритеты

При вызове команд для дебага ивентов выводятся значения приоритетов для вызова функций. К примеру, если какая-то функция среди всех имеет приоритет выше чем другие - то она будет вызвана в первую очередь. Если же две функции будут иметь одинаковый приоритет, например, 0 - то они вызовутся по очереди. А функции, которые имеют наименьший приоритет - будут вызываться последними. В общем, приоритет задаёт порядок вызова функций.

```
Registered Listeners for "kernel.exception" Event
=================================================

 ------- ------------------------------------------------------------------------ ---------- 
  Order   Callable                                                                 Priority  
 ------- ------------------------------------------------------------------------ ---------- 
  #1      App\EventSubscriber\ExceptionSubscriber::processException()              10        
  #2      App\EventSubscriber\ExceptionSubscriber::logException()                  0         
  #3      Symfony\Component\HttpKernel\EventListener\ErrorListener::logKernelException()    0         
  #4      Symfony\Component\HttpKernel\EventListener\ProfilerListener::onKernelException()  0         
  #5      App\EventSubscriber\ExceptionSubscriber::notifyException()               -10       
  #6      Symfony\Component\HttpKernel\EventListener\RouterListener::onKernelException()    -64       
  #7      Symfony\Component\HttpKernel\EventListener\ErrorListener::onKernelException()     -128      
 ------- ------------------------------------------------------------------------ ---------- 
```

Информационный видеосервис для разработчиков программного обеспечения

TestProvider Pearson CyberBionic systematics Microsoft Partner Windows Azure EBA

Проверка знаний

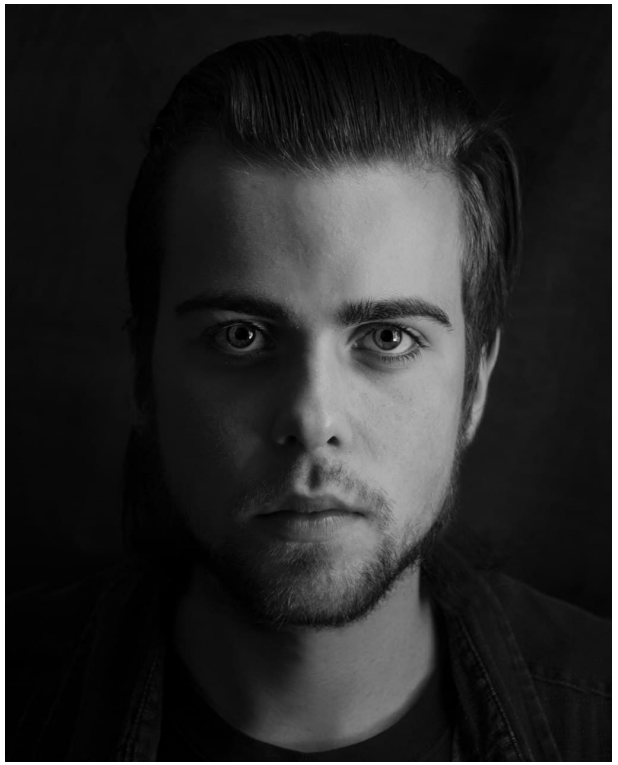
TestProvider.com

TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и для общей оценки знаний IT специалиста.

Проверьте как Вы усвоили данный материал на TestProvider.com

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.

Спасибо за внимание! До новых встреч!



Демьян Костельный
Middle PHP Developer