

L'Event Dispatcher

Table des matières

I. Contexte	3
II. Fondamentaux de l'Event Dispatcher de Symfony	3
A. Fondamentaux de l'Event Dispatcher de Symfony.....	3
B. Exercice : Quiz.....	6
III. Créer des listeners et des subscribers	7
A. Listeners.....	7
B. Exercice : Quiz.....	12
IV. Essentiel	12
V. Auto-évaluation	13
A. Exercice	13
B. Test.....	13
Solutions des exercices	14

I. Contexte

Durée : 1 h

Environnement de travail :

- PC ou Mac avec 8 Go de ram minimum
- Ide : VSCode ou autre

Prérequis :

- Notion de PHP
- Notion de POO

Contexte

L'Event Dispatcher de Symfony est un composant clé du framework Symfony qui permet de gérer les événements et les listeners dans une application web en répondant à ces événements et en exécutant des actions spécifiques.

Le système de gestion d'événements de Symfony est basé sur le design pattern Mediator et Observer, dans lequel il y a deux parties principales : l'objet observé (ou sujet) et les observateurs (ou listeners). Le sujet est responsable de la génération des événements, tandis que les listeners écoutent ces événements et y répondent en exécutant des actions spécifiques.

L'Event Dispatcher est articulé autour de trois composants principaux : l'EventDispatcher, l'Event, et le Listener. Dans ce cours, nous apprendrons comment configurer Event Dispatcher, comment définir des événements, mais également comment ajouter des listeners.

II. Fondamentaux de l'Event Dispatcher de Symfony

A. Fondamentaux de l'Event Dispatcher de Symfony

Définition

Événements ou Event

Un événement est un signal qui est émis par une partie de votre application pour indiquer qu'une action a eu lieu. Les événements peuvent être déclenchés à n'importe quel moment pendant le cycle de vie de votre application. Par exemple, un événement peut être déclenché lorsqu'un utilisateur s'inscrit sur votre site, lorsqu'un nouvel article est publié, ou lorsqu'un message est envoyé à partir de votre application.

On appelle aussi Event un objet qui contient des informations sur un événement spécifique. Il est transmis aux listeners qui écoutent cet événement et peut contenir des informations sur l'état de l'application au moment où l'événement a été déclenché.

Exemple

Voici une instance de kernel.request

```
1 $event = new Symfony\Component\HttpFoundation\RequestEvent();
```

Kernel.request est un événement qui est déclenché à chaque fois qu'une nouvelle requête HTTP est reçue par l'application. Les écouteurs d'événements pour cet événement peuvent effectuer des actions telles que la vérification de l'authentification de l'utilisateur ou la mise en cache de certaines données pour améliorer les performances.

Définition Listeners

Un listener est une méthode qui écoute un événement spécifique et qui est exécuté lorsque cet événement est déclenché. Les listeners sont des fonctions que vous écrivez dans votre application Symfony pour réagir aux événements déclenchés par l'Event Dispatcher. Chaque listener est attaché à un ou plusieurs événements et est responsable de la réponse à l'événement en exécutant des actions spécifiques.

Exemple

Voici un exemple de listener qui récupère la requête en cours à l'aide de l'objet RequestEvent qui est un événement ou Event. Il vérifie ensuite si l'utilisateur est connecté en utilisant la méthode `$this->getUser()` qui est fournie par Symfony. Si l'utilisateur n'est pas connecté, le listener crée une réponse d'erreur 401 et l'assigne à l'événement à l'aide de la méthode `setResponse()`. Cela permet de stopper le traitement de la requête en cours et de renvoyer la réponse d'erreur à la place.

```
1 $listener = function(RequestEvent $event) {
2     // On récupère la requête en cours
3     $request = $event->getRequest();
4
5     // On vérifie si l'utilisateur est connecté
6     $user = $this->getUser();
7     if (!$user) {
8         // On crée une réponse d'erreur 401 (Unauthorized)
9         $response = new Response('Unauthorized', Response::HTTP_UNAUTHORIZED);
10        // On assigne la réponse à l'événement pour stopper le traitement de la requête
11        $event->setResponse($response);
12    }
13};
```

Définition Event Dispatcher

L'Event Dispatcher est l'objet central qui gère la communication entre les événements et les listeners. Il reçoit les événements déclenchés dans votre application et les transmet aux listeners appropriés. L'Event Dispatcher est responsable de la gestion des listeners et de leur enregistrement auprès des événements.

Exemple

```
1 use Symfony\Component\HttpKernel\Event\RequestEvent;
2 use Symfony\Component\HttpFoundation\Response;
3
4 // ...
5
6 // On crée un objet Event Dispatcher
7 $dispatcher = new Symfony\Component\EventDispatcher\EventDispatcher();
8
9 $listener = function(RequestEvent $event) {
10    // On récupère la requête en cours
11    $request = $event->getRequest();
12
13    // On vérifie si l'utilisateur est connecté
14    $user = $this->getUser();
15    if (!$user) {
16        // On crée une réponse d'erreur 401 (Unauthorized)
17        $response = new Response('Unauthorized', Response::HTTP_UNAUTHORIZED);
18        // On assigne la réponse à l'événement pour stopper le traitement de la requête
19        $event->setResponse($response);
20    }
21}
```

```

21 };
22
23 // On enregistre le listener pour l'événement kernel.request
24 $dispatcher->addListener('kernel.request', $listener);

```

Le code crée un objet EventDispatcher pour gérer les événements et les listeners dans l'application. Ensuite, il définit une fonction anonyme (\$listener) qui sera exécutée lors de l'événement kernel.request, comme expliqué dans l'exemple du listener.

Puis, le code enregistre le listener pour l'événement kernel.request à l'aide de la méthode \$dispatcher->addListener(). Cela permet à Symfony d'appeler la fonction de vérification d'authentification lors de chaque requête HTTP reçue par l'application.

Events de Symfony

Symfony fournit plusieurs événements par défaut que les développeurs peuvent utiliser dans leurs applications. Ces événements sont déclenchés à différents moments du cycle de vie de l'application et permettent aux développeurs de Symfony d'intercepter et de modifier le comportement de l'application à ces moments clés.

Voici les Events du cycle de vie du Kernel. Les listeners associés à ces événements peuvent effectuer des actions en réponse à ces événements. Par exemple enregistrer des informations dans un journal ou modifier la réponse ou encore rediriger l'utilisateur vers une autre page :

- kernel.request : déclenché lorsque la requête HTTP est reçue par l'application.
- kernel.controller : déclenché avant que le contrôleur ne soit exécuté pour traiter la requête.
- kernel.controller_arguments : déclenché après que le noyau a déterminé le contrôleur à utiliser pour traiter la requête, mais avant que celui-ci ne soit appelé.
- kernel.view : déclenché lorsque le contrôleur a terminé de traiter la requête et a retourné une réponse, mais avant que cette réponse ne soit envoyée au client.
- kernel.response : déclenché après que la réponse a été envoyée au client.
- kernel.finish_request : déclenché après que toutes les étapes de traitement de la requête ont été effectuées, y compris l'envoi de la réponse au client.
- kernel.terminate : déclenché après que la réponse a été envoyée au client et que toutes les connexions ont été fermées.
- kernel.exception : déclenché lorsqu'une exception se produit pendant le traitement de la requête.

Debugger d'événements

Symfony fournit un débogueur qui vous permet de connaître la liste de tous les Events de votre application et des listeners qui y sont attachés.

```
1 $ php bin/console debug:event-dispatcher
```

Vous y trouverez, bien sûr, ceux du Kernel, mais aussi ceux qui seront mis en place par les Packages que vous installerez, par exemple ceux de Security, mais aussi ceux que vous créez (nous verrons cela plus tard). C'est pour cela qu'à chaque nouveau package installé, il peut être intéressant de regarder quels ont été les nouveaux events ajoutés et de comprendre leurs intérêts. On en a pour preuve les événements Doctrine, Form, etc.

Registered Listeners Grouped by Event

"Symfony\Component\Security\Http\Event\CheckPassportEvent" event

Order	Callable	Priority
#1	Symfony\Component\Security\Http\Event\Listener\UserProviderListener::checkPassport()	1824
#2	Symfony\Component\Security\Http\Event\Listener\CookieTokenListener::checkPassport()	332
#3	Symfony\Component\Security\Http\Event\Listener\CheckCredentialsListener::checkPassport()	0

Exemple d'Event listé par le débogueur de Symfony

Dans l'exemple ci-dessus, on voit que `CheckPassportEvent` possède 3 EventListeners et qu'ils ont un numéro dans la colonne `Priority` (voir paragraphe « **Priority** »).

Vous pouvez aussi choisir un seul événement pour connaître toutes les méthodes (listeners) associées, en ajoutant le nom de l'événement comme dans l'exemple ci-dessous.

```
1 $ php bin/console debug:event-dispatcher kernel.exception
```

Priority

Lorsqu'un événement est déclenché, les écouteurs (listeners) sont appelés dans un ordre de priorité spécifique.

La priorité ne détermine pas l'ordre d'exécution des écouteurs dans différents objets, mais plutôt l'ordre d'exécution des méthodes des écouteurs dans un même objet. Si vous avez plusieurs écouteurs pour un même événement et que vous souhaitez contrôler l'ordre dans lequel ils sont appelés, vous devrez les enregistrer dans le bon ordre lors de leur ajout au service container de Symfony.

L'ordre de priorité est déterminé par la méthode `getSubscribedEvents()` dans la classe du listener.

Cette méthode retourne un tableau associatif qui contient les noms des événements écoutés en clé et les méthodes à exécuter en valeur. Chaque méthode peut également avoir une priorité, qui est un nombre entier représentant son ordre d'exécution parmi les autres méthodes écoutant le même événement. Par défaut, les écouteurs ont une priorité de 0, mais cela peut être changé en ajoutant une clé `"priority"` avec une valeur entière à la méthode `getSubscribedEvents()` pour chaque événement écouté.

Exemple Méthode `getSubscribedEvents()`

Voici un exemple de la méthode `getSubscribedEvents()` avec des priorités spécifiées :

```
1 public static function getSubscribedEvents()
2 {
3     return [
4         'event.name' => [
5             ['methodName1', 10],
6             ['methodName2', 0],
7             ['methodName3', -10],
8         ],
9     ];
10 }
```

Dans cet exemple, les méthodes `methodName1`, `methodName2` et `methodName3` sont toutes écoutées pour l'événement `"event.name"`. La méthode `methodName1` a une priorité de 10, la méthode `methodName2` a une priorité de 0 (par défaut) et la méthode `methodName3` a une priorité de -10. Lorsque l'événement est déclenché, les méthodes sont appelées dans l'ordre suivant : `methodName1`, `methodName2`, `methodName3`.

B. Exercice : Quiz

[solution n°1 p.15]

Question 1

Qu'est-ce que l'Event Dispatcher de Symfony ?

- ☐ Un outil pour gérer les événements et les listeners associés dans une application web
- ☐ Un composant pour créer des modèles MVC dans une application Symfony
- ☐ Un outil pour gérer les sessions utilisateur dans une application Symfony

Question 2

Qu'est-ce qu'un événement dans Symfony ?

- ☐ Un signal émis par une partie de votre application pour indiquer qu'une action a eu lieu
- ☐ Un objet qui contient des informations sur un événement spécifique
- ☐ Une méthode qui écoute un événement spécifique et qui est exécutée lorsque cet événement est déclenché

Question 3

Qu'est-ce qu'un listener dans Symfony ?

- ☐ Un objet dans lequel sont stockées des informations
- ☐ Un signal servant à savoir qu'une action va avoir lieu
- ☐ Une classe ayant des méthodes associées à un event

Question 4

Quels sont les trois composants principaux de l'Event Dispatcher ?

- ☐ Dispatcher, Listener, Contrôleur
- ☐ EventDispatcher, Event, Listener
- ☐ Event, Contrôleur, Listener

Question 5

Comment est déterminé l'ordre d'exécution des méthodes des écouteurs dans un même objet ?

- ☐ Par l'ordre dans lequel les écouteurs ont été enregistrés dans le service container de Symfony
- ☐ Par la méthode `getSubscribedEvents()` dans la classe du listener
- ☐ Par la priorité spécifiée pour chaque méthode dans la méthode `getSubscribedEvents()`

III. Créer des listeners et des subscribers

A. Listeners

Imaginons maintenant que nous voulions écouter si une requête est une requête Ajax.

Nous avons deux choses à faire. Configurer le fichier `services.yaml` afin que Symfony sache quels listeners doivent être appelés pour notre événement et créer une classe `AjaxListener` par exemple avec une méthode qui gèrera notre demande.

Méthode Configurer le fichier services.yaml

Voici comment il faut le configurer :

```
1 # config/services.yaml
2 services:
3     App\EventListener\AjaxListener:
4         tags:
5             - { name: kernel.event_listener, event: kernel.request }
```

Dans cet exemple, nous déclarons le service `App\EventListener\AjaxListener` et nous lui ajoutons un tag `kernel.event_listener` (qui est le listener que nous voulons récupérer pour `AjaxListener`) avec le paramètre `event`. Le paramètre `event` indique l'événement auquel le listener doit être attaché, dans ce cas `kernel.request`.

Méthode Créer la classe AjaxListener

Voici comment ce que vous pourriez écrire par la suite dans votre fichier PHP :

```
1 <?php
2
3 namespace App\EventListener;
4
5 use Symfony\Component\HttpFoundation\Response;
6 use Symfony\Component\HttpFoundation\JsonResponse;
7 use Symfony\Component\HttpKernel\Event\RequestEvent;
8
9 class AjaxListener
10 {
11     public function onKernelRequest(RequestEvent $event)
12     {
13         $request = $event->getRequest();
14
15         if ($request->isXmlHttpRequest()) {
16             // La requête est une requête Ajax
17             // Ajouter un en-tête pour indiquer que la réponse est une réponse JSON
18             $response = new JsonResponse();
19             $response->headers->set('Content-Type', 'application/json');
20
21             // Définir un en-tête de sécurité pour empêcher les attaques CSRF
22             $response->headers->set('X-CSRF-Token', $request->attributes->get('_csrf_token'));
23
24             // Définir la réponse de l'événement
25             $event->setResponse($response);
26         }
27     }
28 }
```

Dans cet exemple, la méthode `onKernelRequest` vérifie si la requête est une requête Ajax à l'aide de la méthode `isXmlHttpRequest` de l'objet `Request`. Si c'est le cas, la méthode crée une `JsonResponse` et définit :

- Un en-tête pour indiquer que la réponse est une réponse JSON
- Un en-tête de sécurité pour empêcher les attaques CSRF (Cross-Site Request Forgery)
- La réponse de l'événement à l'aide de la méthode `setResponse` de l'objet `RequestEvent`

Cela vous permet d'ajouter des en-têtes de sécurité spécifiques aux réponses Ajax sans avoir à les répéter dans chaque contrôleur qui renvoie une réponse JSON.

Listeners Vs Subscribers

Un « *listener* » (ou « *événement listener* » en français) et un « *subscriber* » (ou « *événement subscriber* ») sont deux concepts similaires dans Symfony, mais avec quelques différences importantes.

Il s'agit d'un objet qui écoute un événement spécifique déclenché par l'application. Un subscriber, quant à lui, est un objet qui s'abonne à plusieurs événements à la fois. Il est capable de traiter plusieurs événements en même temps, en exécutant une ou plusieurs méthodes en réponse à chacun d'eux. Les subscribers doivent implémenter l'interface "EventSubscriberInterface" pour spécifier les événements auxquels ils s'abonnent et les méthodes à exécuter en réponse.

En résumé, la principale différence entre un listener et un subscriber est que le premier écoute un seul événement à la fois, tandis que le second peut en écouter plusieurs en même temps. Les listeners sont généralement utilisés pour des tâches simples et ponctuelles, tandis que les subscribers sont plus adaptés pour des tâches plus complexes et/ou pour traiter plusieurs événements liés entre eux. Cependant, dans certains cas, les deux peuvent être utilisés pour réaliser la même tâche et le choix dépendra des besoins spécifiques de l'application et de la préférence du développeur.

Méthode Implanter un subscriber

Bien que l'on puisse créer « *à la main* » un subscriber, le mieux c'est d'utiliser les outils que Symfony nous fournit, à savoir MakerBundle que vous avez certainement dû utiliser plusieurs fois avant d'en arriver à ce cours. Alors, utilisez la ligne de commande ci-dessous dans le terminal :

```
1 $ php bin/console make:subscriber
```

Celui-ci va vous demander de choisir un nom pour la classe subscriber à créer. Notez que vous auriez aussi pu directement mettre le nom que vous vouliez donner à votre subscriber à la suite de votre ligne de commande.

Ici, vous allez choisir `LogSubscriber`. Le terminal va ensuite vous demander de choisir à quel Event vous souhaitez relier ce Subscriber alors choisissez `Kernel.request`.

C'est fait, vous avez maintenant créé `src/EventSubscriber/LogSubscriber.php` et avez deux méthodes dans `logSubscriber` qui héritent de `EventSubscriberInterface`

- `getSubscribedEvents()` qui a relié pour vous `KernelEvents` à l'autre méthode,
- `onKernelRequest()` qu'il vous faut maintenant définir.

Exemple

Voici un exemple de comment on pourrait implémenter `onKernelRequest()` :

```
1 <?php
2
3 namespace App\EventSubscriber;
4
5 use Symfony\Component\EventDispatcher\EventSubscriberInterface;
6 use Symfony\Component\HttpKernel\Event\RequestEvent;
7 use Symfony\Component\HttpKernel\KernelEvents;
8 use Psr\Log\LoggerInterface;
9 use Symfony\Component\HttpFoundation\RequestStack;
10
11 class LogSubscriber implements EventSubscriberInterface
12 {
13     public function __construct(
14         private LoggerInterface $logger,
15         private RequestStack $requestStack
16     ) {
17     }
18     public function onKernelRequest(RequestEvent $event)
```

```

19 {
20     $request = $event->getRequest();
21     $this->logger->info(sprintf('Request from %s', $request->getClientIp()));
22 }
23
24 public static function getSubscribedEvents(): array
25 {
26     return [
27         KernelEvents::REQUEST => 'onKernelRequest',
28     ];
29 }
30 }

```

La méthode `onKernelRequest()` de cet exemple est destinée à être appelée chaque fois que l'événement `KernelEvents::REQUEST` est déclenché. Ainsi lors de requête HTTP, l'objet `Request` utilise la méthode `getClientIp()` de cet objet pour enregistrer une ligne de journalisation informant de l'adresse IP du client qui a effectué la requête.

Il nous faut également configurer le fichier `services.yaml`. Tout comme pour le listener, il convient de noter le chemin d'accès du subscriber et dans les tags de mettre le nom du subscriber voulu. Il n'y a pas besoin d'associer d'Événement avec les Subscribers.

```

1 # config/services.yaml
2 services:
3     App\EventSubscriber\LogSubscriber:
4         tags:
5             - { name: kernel.event_subscriber }

```

Arguments d'un subscriber

Lorsque vous déclarez un Subscriber dans le fichier `services.yaml` de votre application Symfony, vous pouvez spécifier plusieurs arguments pour configurer son fonctionnement. Voici les principaux arguments que vous pouvez utiliser :

- **tag** : cet argument permet de définir le tag associé à votre Subscriber. Le tag est utilisé par Symfony pour repérer les Subscriber et les enregistrer automatiquement dans l'objet `EventDispatcher` de l'application.
- **eventSubscriber** : cet argument permet de définir une classe qui implémente l'interface `EventSubscriberInterface`. Cette interface définit deux méthodes : `getSubscribedEvents()`, qui retourne un tableau d'événements auxquels le Subscriber est associé, et `onEvent()`, qui est appelée lorsqu'un événement se produit. Si vous définissez cet argument, Symfony va automatiquement appeler les méthodes `getSubscribedEvents()` et `onEvent()` de la classe spécifiée pour enregistrer les Listeners associés à chaque événement.
- **arguments** : cet argument permet de définir des arguments personnalisés à injecter dans le constructeur de votre Subscriber. Vous pouvez utiliser des expressions de service pour injecter des services ou des paramètres de configuration de votre application.

Exemple

Voici un exemple de déclaration de Subscriber dans le fichier `services.yaml` avec plusieurs arguments :

```

1 # config/services.yaml
2 services:
3     App\EventListener\MySubscriber:
4         arguments:
5             - '@my_service'
6             - '%my_parameter%'

```

```
7     tags:
8         - { name: kernel.event_subscriber }
```

Cet exemple définit un Subscriber de la classe `App\EventListener\MySubscriber`, qui a deux arguments à injecter dans son constructeur : le service `my_service` et le paramètre `my_parameter` de l'application. Le Subscriber est également associé à l'événement `kernel` grâce au tag `kernel.event_subscriber`.

Exemples pratiques d'utilisation de l'Event Dispatcher

L'Event Dispatcher de Symfony est un outil puissant pour personnaliser le comportement de votre application en réponse aux événements qui se produisent. Voici quelques exemples pratiques d'utilisation de l'Event Dispatcher avec des listeners et des subscribers :

- **Logging** : enregistrer des événements importants dans un journal (log). Par exemple, à chaque fois qu'un utilisateur se connecte ou échoue à se connecter à votre application.
- **Notification** : envoyer des notifications en temps réel à vos utilisateurs lorsqu'un événement se produit. Par exemple, lorsqu'un nouveau message est publié sur le fil d'actualités d'un utilisateur.
- **Modification de données** : modifier des données en réponse à un événement. Par exemple, modifier un message avant qu'il ne soit enregistré en BDD.
- **Sécurité** : implémenter des mesures de sécurité dans votre application. Par exemple, écouter l'événement `kernel.request` pour vérifier les autorisations d'un utilisateur.
- **Débogage** : déboguer votre application en affichant des informations sur les événements qui se produisent. Par exemple, enregistrer chaque fois qu'une requête échoue ou qu'un utilisateur soumet un formulaire invalide.
- **Envoyer un e-mail** : par exemple, lors d'une nouvelle inscription sur votre site, déclencher un événement qui envoie un e-mail de confirmation à l'utilisateur.
- **Enregistrer une activité d'utilisateur** : par exemple, créer un événement qui déclenche une fonction qui enregistre cette activité dans une base de données ou un journal.
- **Gérer les erreurs** : par exemple, déclencher un événement d'erreur qui envoie une notification à l'administrateur du site.
- **Modifier le comportement du formulaire** : lorsqu'un utilisateur soumet un formulaire, vous pouvez modifier le comportement du formulaire. Par exemple, en créant un événement qui déclenche une fonction qui ajoute un champ supplémentaire.
- **Intégration avec d'autres services** : par exemple, vous pouvez créer un événement qui déclenche une fonction qui envoie les détails de la commande à un système de gestion de commande externe.

L'Event Dispatcher est un composant clé de Symfony qui permet de gérer les événements dans une application. Il offre de nombreuses possibilités pour personnaliser le comportement de l'application en fonction des événements qui se produisent. Dès lors que plusieurs événements et actions seront amenés à se suivre, l'utilisation du composant Workflow deviendra nécessaire.

FormEvent

Les événements de formulaire sont des événements déclenchés lorsqu'une action est effectuée sur un formulaire Symfony. Ces événements sont un moyen puissant de personnaliser le comportement d'un formulaire à chaque étape du processus de soumission.

Dans Symfony, il existe de nombreux événements de formulaire pré-définis que les développeurs peuvent écouter et personnaliser en fonction de leurs besoins.

B. Exercice : Quiz

[solution n°2 p.16]

Question 1

Quelle est la différence entre un listener et un subscriber ?

- ☐ Il n'y a pas de différence, ce sont deux termes différents pour désigner la même chose
- ☐ Un subscriber est un type particulier de listener qui peut s'abonner à plusieurs types d'événements
- ☐ Un listener est un type particulier de subscriber qui ne peut s'abonner qu'à un seul type d'événement

Question 2

Comment déclare-t-on un service de listener dans le fichier services.yaml ?

- ☐ En utilisant la fonction event_listener avec le nom et l'événement du listener
- ☐ En utilisant le tag kernel.event_listener avec le nom et l'événement du listener
- ☐ En utilisant le tag event_listener avec le nom et l'événement du listener

Question 3

Comment peut-on créer un subscriber dans Symfony ?

- ☐ En créant une classe normale qui hérite de EventSubscriberInterface et en l'enregistrant dans le fichier services.yaml
- ☐ En utilisant la commande make:subscriber dans la console Symfony
- ☐ En créant une classe qui hérite de SubscriberInterface et en l'enregistrant dans le fichier services.yaml

Question 4

Quelle est la méthode de l'interface EventSubscriberInterface qui doit être implémentée dans un subscriber ?

- ☐ getSubscribedEvents()
- ☐ handleEvents()
- ☐ onEvent()

Question 5

Quel est l'exemple pratique d'utilisation de l'Event Dispatcher pour enregistrer des événements dans un journal ?

- ☐ Logging
- ☐ Notification
- ☐ Modification de données

IV. Essentiel

Dans ce cours sur l'Event Dispatcher de Symfony, nous avons vu les concepts de base liés aux événements et aux listeners ainsi que le rôle de l'Event Dispatcher dans la communication entre eux.

Les événements(Events) sont des signaux émis par une partie de l'application pour indiquer qu'une action a eu lieu, et les listeners sont des méthodes qui écoutent les événements et y répondent en exécutant des actions spécifiques. L'Event Dispatcher est l'objet central qui gère la communication entre les événements et les listeners, il reçoit les événements et les transmet aux listeners appropriés.

Symfony fournit plusieurs événements par défaut que les développeurs peuvent utiliser dans leurs applications pour intercepter et modifier le comportement de l'application à des moments clés. Les événements du cycle de vie du Kernel, tels que `kernel.request`, etc. sont déclenchés à différents moments du cycle de vie de l'application. Les listeners associés à ces événements peuvent effectuer des actions en réponse à ces événements, telles que l'enregistrement d'informations dans un journal, la modification de la réponse ou la redirection de l'utilisateur vers une autre page.

En conclusion, ce cours sur les fondamentaux de l'Event Dispatcher de Symfony est un élément clé pour comprendre le fonctionnement des événements et des listeners dans les applications Symfony. Les développeurs peuvent utiliser ces concepts pour intercepter et modifier le comportement de l'application à des moments clés. La compréhension de la priorité des listeners est également importante pour contrôler l'ordre d'exécution des méthodes et garantir que l'application fonctionne comme prévu.

V. Auto-évaluation

A. Exercice

Dans notre application web, lorsqu'un utilisateur s'inscrit, nous souhaitons lui envoyer un e-mail de bienvenue pour le remercier de son inscription et lui donner quelques informations sur les fonctionnalités de l'application.

Question 1

[solution n°3 p.17]

Modifier la class `RegistrationFormType` créer avec `make:registration-form` afin qu'à l'écoute de l'inscription un e-mail soit automatiquement envoyé.

Question 2

[solution n°4 p.18]

Sachant que `prePersist` est un événement de `doctrine.event.listener` qui permet de modifier une entité avant qu'elle ne soit enregistrée, créez dans `services.yaml` un événement qui utilise `prePersist`.

B. Test

Exercice 1 : Quiz

[solution n°5 p.18]

Question 1

Quelle interface doit être implémentée pour spécifier les événements auxquels un subscriber s'abonne dans Symfony ?

- ☐ EventInterface
- ☐ SubscriberInterface
- ☐ EventSubscriberInterface

Question 2

Dans l'exemple ci-dessous, quelle méthode est prioritaire pour l'événement "event.name" ?

```
1 public static function getSubscribedEvents()  
2 {  
3     return [  
4         'event.name' => [  
5             ['methodName1', 128],  
6             ['methodName2', -10],  
7             ['methodName3', 256],  
8         ],  
9     ];  
10 }
```

- ☐ methodName1
- ☐ methodName2
- ☐ methodName3

Question 3

Quel est le rôle de l'Event Dispatcher dans la gestion des listeners ?

- ☐ Il est responsable de leur enregistrement auprès des événements
- ☐ Il est responsable de leur création et de leur suppression
- ☐ Il est responsable de leur exécution lorsque l'événement est déclenché

Question 4

Avec un listener il est possible de modifier des données ?

- ☐ Vrai
- ☐ Faux

Question 5


Si vous avez deux événements ou plus qu'allez-vous utiliser ?

- ☐ Un Listener
- ☐ Un Subscriber
- ☐ Peu importe, les deux ont la même utilité

Solutions des exercices


Exercice p. 6 Solution n°1**Question 1**

Qu'est-ce que l'Event Dispatcher de Symfony ?

- ☒ Un outil pour gérer les événements et les listeners associés dans une application web
- ☐ Un composant pour créer des modèles MVC dans une application Symfony
- ☐ Un outil pour gérer les sessions utilisateur dans une application Symfony
-  L'Event Dispatcher vous permet de déclencher un événement qui sera écouté et amènera de futures actions dans l'application.


Question 2

Qu'est-ce qu'un événement dans Symfony ?

- ☒ Un signal émis par une partie de votre application pour indiquer qu'une action a eu lieu
- ☐ Un objet qui contient des informations sur un événement spécifique
- ☐ Une méthode qui écoute un événement spécifique et qui est exécutée lorsque cet événement est déclenché
-  Lorsqu'un événement est déclenché, toutes les méthodes d'écoute enregistrées pour cet événement sont appelées.


Question 3

Qu'est-ce qu'un listener dans Symfony ?

- ☐ Un objet dans lequel sont stockées des informations
- ☐ Un signal servant à savoir qu'une action va avoir lieu
- ☒ Une classe ayant des méthodes associées à un event
-  Un listener grâce à son écoute permet de définir des actions à exécuter en réponse à des événements spécifiques.


Question 4

Quels sont les trois composants principaux de l'Event Dispatcher ?

- ☐ Dispatcher, Listener, Contrôleur
- ☒ EventDispatcher, Event, Listener
- ☐ Event, Contrôleur, Listener
-  Le contrôleur n'est pas un composant de l'Event Dispatcher.

Question 5


Comment est déterminé l'ordre d'exécution des méthodes des écouteurs dans un même objet ?

- ☐ Par l'ordre dans lequel les écouteurs ont été enregistrés dans le service container de Symfony
- ☐ Par la méthode `getSubscribedEvents()` dans la classe du listener
- ☒ Par la priorité spécifiée pour chaque méthode dans la méthode `getSubscribedEvents()`
-  Plus la valeur donnée est grande, plus l'événement est prioritaire.

Exercice p. 12 Solution n°2


Question 1

Quelle est la différence entre un listener et un subscriber ?

- ☐ Il n'y a pas de différence, ce sont deux termes différents pour désigner la même chose
- ☐ Un subscriber est un type particulier de listener qui peut s'abonner à plusieurs types d'événements
- ☒ Un listener est un type particulier de subscriber qui ne peut s'abonner qu'à un seul type d'événement
-  Un listener est un type de subscriber qui ne peut écouter qu'un seul type d'événement spécifique.


Question 2

Comment déclare-t-on un service de listener dans le fichier `services.yaml` ?

- ☐ En utilisant la fonction `event_listener` avec le nom et l'événement du listener
- ☒ En utilisant le tag `kernel.event_listener` avec le nom et l'événement du listener
- ☐ En utilisant le tag `event_listener` avec le nom et l'événement du listener
-  Effectivement il faut utiliser le tag et préciser dans `event` l'événement que l'on veut suivre.

Question 3

Comment peut-on créer un subscriber dans Symfony ?

- ☐ En créant une classe normale qui hérite de `EventSubscriberInterface` et en l'enregistrant dans le fichier `services.yaml`
- ☒ En utilisant la commande `make:subscriber` dans la console Symfony
- ☐ En créant une classe qui hérite de `SubscriberInterface` et en l'enregistrant dans le fichier `services.yaml`
-  `MakerBundle` est un outil puissant qui peut également nous aider à créer un subscriber, sans avoir à toucher le fichier `services.yaml`.

Question 4

Quelle est la méthode de l'interface `EventSubscriberInterface` qui doit être implémentée dans un subscriber ?

- ☒ `getSubscribedEvents()`
- ☐ `handleEvents()`
- ☐ `onEvent()`

- Q Cette méthode retourne un tableau de clé-valeur, la clé est l'événement auquel vous souhaitez vous abonner et la valeur est le nom de la méthode que vous souhaitez appeler lors de la réception de cet événement.

Question 5

Quel est l'exemple pratique d'utilisation de l'Event Dispatcher pour enregistrer des événements dans un journal ?

- ☒ Logging
- ☐ Notification
- ☐ Modification de données

- Q Le logging est un mécanisme de journalisation qui permet de capturer et d'enregistrer les événements importants ou les erreurs.

p. 13 Solution n°3

```

1 class RegistrationFormType extends AbstractType
2 {
3
4     public function buildForm(FormBuilderInterface $builder, array $options): void
5     {
6         $builder
7             ->add('e-mail')
8             ->add('agreeTerms', CheckboxType::class, [
9                 'mapped' => false,
10                'constraints' => [
11                    new IsTrue([
12                        'message' => 'Vous devez accepter les conditions.',
13                    ]),
14                ],
15            ])
16            ->add('plainPassword', PasswordType::class, [
17
18                'mapped' => false,
19                'attr' => ['autocomplete' => 'new-password'],
20                'constraints' => [
21                    new NotBlank([
22                        'message' => 'Entrez un mot de passe',
23                    ]),
24                    new Length([
25                        'min' => 6,
26                        'minMessage' => 'votre mot de passe doit contenir au moins {{ limit }}
27                caracteres',
28                        'max' => 4096,
29                    ]),
30                ],
31            ]);
32
33        $builder->addEventListener(
34            FormEvents::SUBMIT,
35            function (FormEvent $event) use ($options) {
36                $user = $event->getData();
37                $form = $event->getForm();
38
39                $mailer = $options['mailer'];
40                $message = (new Email())

```

```

40         ->from('masociete@example.com')
41         ->to($user->getEmail())
42         ->subject('Bienvenue')
43         ->text('Merci de vous être inscrit sur notre site');
44
45         $mailer->send($message);
46     }
47 );
48 }
49
50 public function configureOptions(OptionsResolver $resolver): void
51 {
52     $resolver->setRequired('mailer');
53     $resolver->setDefaults([
54         'data_class' => User::class,
55     ]);
56 }
57 }

```

p. 13 Solution n°4

```


1 # config/services.yaml
2
3 services:
4     App\EventListener\MyDoctrineEventListener:
5         tags:
6             - { name: doctrine.event_listener, event: prePersist }

```

Exercice p. 13 Solution n°5

Question 1

Quelle interface doit être implémentée pour spécifier les événements auxquels un subscriber s'abonne dans Symfony ?

- ☐ EventInterface
- ☐ SubscriberInterface
- ☒ EventSubscriberInterface
-  Cette interface apporte 2 méthodes, dont getSubscribedEvents().

Question 2

Dans l'exemple ci-dessous, quelle méthode est prioritaire pour l'événement "event.name" ?


```

1 public static function getSubscribedEvents()
2 {
3     return [
4         'event.name' => [
5             ['methodName1', 128],
6             ['methodName2', -10],
7             ['methodName3', 256],
8         ],
9     ];

```

10 }


- ☐ methodName1
- ☐ methodName2
- ☒ methodName3

 La priorité qui représente l'ordre d'exécution est attribuée à la plus grande valeur numérique.

Question 3

Quel est le rôle de l'Event Dispatcher dans la gestion des listeners ?


- ☒ Il est responsable de leur enregistrement auprès des événements
- ☐ Il est responsable de leur création et de leur suppression
- ☐ Il est responsable de leur exécution lorsque l'événement est déclenché

 Son rôle est de recevoir des événements et de les distribuer à tous les écouteurs (listeners) qui ont été enregistrés pour écouter cet événement.

Question 4

Avec un listener il est possible de modifier des données ?


- ☒ Vrai
- ☐ Faux

 Vrai. Un listener écoute un ou des événements, mais peut aussi agir, notamment en modifiant des données.

Question 5

Si vous avez deux événements ou plus qu'allez-vous utiliser ?

- ☐ Un Listener
- ☒ Un Subscriber
- ☐ Peu importe, les deux ont la même utilité

 Le Subscriber vous permet d'abonner un objet à plusieurs événements simultanément.