Boozt.com

SERVICE • SELECTION • SURPRISE

Symfony Events vs. Hooks

Antanas Koncius

About me

Backend developer at Boozt.com

14+ years of experience

Latest projects I've worked on:

- Agrosmart - Farmer's accounting system. DDD, Event-Sourcing, CQRS
- Joberate - Predictive analytics, web scraping
- Ome.Health - Automated wellness recommendations based on person's markers (DNA, blood, gut microbiome, fitness)

How this story started

Stumbled into this (pseudo) code

Sets array to event

```
/** @var array $someData - array with already prefilled data */
$event = new BeforePersistEvent($someData);
$this->eventDispatcher->dispatchEvent(BeforePersistEvent::EVENT_NAME,
$event);

$newSomeData = $event->getData();
```

Fetches mutated array from event
and uses in further logic

But Symfony does it too!

```
// Hook to change content of the model data before transformation and
mapping children
if ($dispatcher->hasListeners(FormEvents::PRE_SET_DATA)) {
    $event = new FormEvent($this, $modelData);
    $dispatcher->dispatch(FormEvents::PRE_SET_DATA, $event);
    $modelData = $event->getData();
}
```

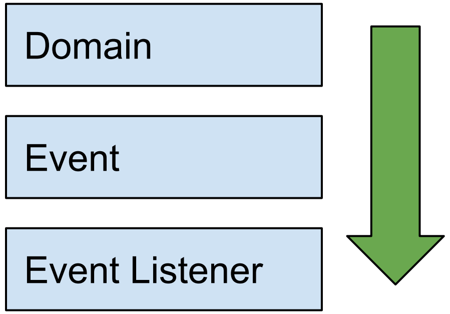
```
$event = new GetResponseEvent($this, $request, $type);
$this->dispatcher->dispatch(KernelEvents::REQUEST, $event);

if ($event->hasResponse()) {
    return $this->filterResponse($event->getResponse(), $request, $type);
}
```

Why is it bad?

- Terminology does not reflect reality
- Introduces additional **unnecessary** layer of complexity
- Makes testing less convenient
- EventDispatcher hides implementation

What is event?

- Notifies that something has happened
- Past tense
- Immutable
- Serializable data structure
- Technology agnostic
- Unidirectional flow of information



What is intention of this code?

Let’s take a look at code one more time:

```
// Hook to change content of the model data before transformation and mapping children
if ($dispatcher->hasListeners(FormEvents::PRE_SET_DATA)) {
    $event = new FormEvent($this, $modelData);
    $dispatcher->dispatch(FormEvents::PRE_SET_DATA, $event);
    $modelData = $event->getData();
}
```

- This code is intended to COLLECT/MUTATE data from external components
- Relies on using references
- Bi-directional flow of information

Maybe it is Hook / Interceptor?

From wikipedia page:

In the field of software development, an **interceptor pattern** is a software design pattern that is used when software systems or frameworks want to offer a way to change, or augment, their usual processing cycle.

Sounds like a good fit!

Pre-persist, form events, kernel events attempt to do exactly that - to offer a way to change their usual processing cycle.

All form events

```
final class FormEvents
{
    const PRE_SUBMIT = 'form.pre_bind';
    const SUBMIT = 'form.bind';
    const POST_SUBMIT = 'form.post_bind';
    const PRE_SET_DATA = 'form.pre_set_data';
    const POST_SET_DATA = 'form.post_set_data';
}
```

Interceptor example

```
interface FormPostSubmitInterceptor {
    public function postSubmit(FormInterface $form, &$viewData);
}

interface FormPreSetInterceptor {
    public function preSetData(FormInterface $form, &$modelData);
}

interface FormPostSetInterceptor {
    public function postSetData(FormInterface $form, &$modelData);
}
```

Using interceptor

```
class Form {
    /** @var FormSubmitInterceptor[] */
    private $formSubmitInterceptors = [];

    public function submit(FormInterface $form, Request $request)
    {
        $formData = $request->get($form->getName());

        /** @var FormSubmitInterceptor $interceptor */
        foreach ($this->formSubmitInterceptors as $interceptor) {
            $interceptor->submit($form, $formData);
        }
        // do stuff with modified $formData
    }
}
```

Configuring interceptors

- Inject interceptors into main component using tags in dependency container
- Symfony supports auto-tagging based on implemented interface

```yaml
App\Domain\EmailSenderInterceptor\BenchmarkInterceptor:
  tags:
    - { name: 'app.email_sender_interceptor.before', priority: 1 }
    - { name: 'app.email_sender_interceptor.after', priority: 1 }
```

Example project: https://github.com/akoncius/interceptor-example

Advantages against symfony events

Simpler tests

- Interceptors are simpler to test because values are provided directly
- Main component also easier to test - just check if it calls ALL interceptors

Explicit code

- Code is clear which interfaces are actually called
- IDE allows ctrl/cmd + click to navigate all interceptors
- Interface serves as a documentation what is expected from interceptor
- You can declare return types, expected exceptions

Disadvantages

Bigger upfront cost

- Write your own tag for dependency container
- Write your own DI “container pass”
- Can end up having a lot of interfaces, depending on “priority” granularity

Final thoughts

- If you are mutating events in listeners - reconsider implementation choices
- Even if renamed EventDispatcher to HookDispatcher, additional layer of “event/hook” does not solve anything

A bit of philosophy

- Think about real intention of code, it will help choosing better suited abstraction
- Precedent is quite important in code base - eventually it will be copy-pasted, so design code carefully