Building Better Controllers

Symfony Live Berlin 2014

Benjamin Eberlei <benjamin@qafoo.com>

31.10.2014

Me

- Working at Qafoo

We promote high quality code with trainings and consulting

`http://qafoo.com`

- Doctrine and Symfony
- `http://whitewashing.de`
- Twitter `@beberlei` and `@qafoo`

Outline

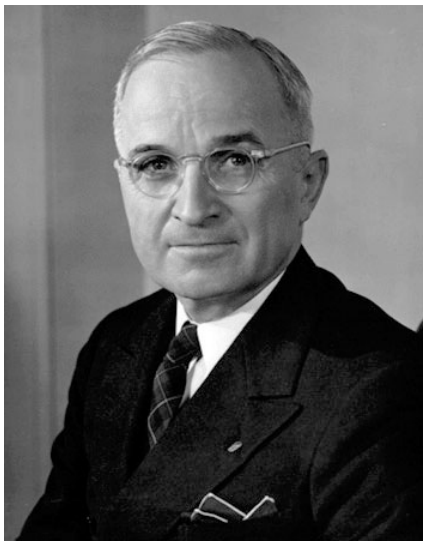
Motivation

Application-Design

- A passion/obsession of mine.
- I occassionally blog about it.
- Mostly Symfony related

Motivation

Give me a one-handed economist! All my economists say, "On the one hand, on the other hand." (Harry S. Truman)



Motivation

Design is about choices/trade-offs

Failure: Overengineering

Success:
Simple design

Some (good!) Choices

Buzzing in the Symfony Community

(My blogging is partially to blame for this, sorry!)

- Domain-Driven-Design
- Command-Query-Responsibility-Seperation
- Event-Sourcing
- Hexagonal Architectures
- Microservices
- Symfony without Symfony

Not even close to 100% of all the choices you can consider.

Consider

A good architecture allows you to defer critical decisions, it doesn't force you to defer them. However, if you can defer them, it means you have lots of flexibility. (Uncle Bob)

Consider

And since your controllers should be thin and contain nothing more than a few lines of glue-code, spending hours trying to decouple them from your framework doesn't benefit you in the long run. The amount of time wasted isn't worth the benefit. (Symfony Best Practices Book)

I don't agree 100%

But there is a lot of truth in it!

- Standard Symfony Controllers usually end up fat!
- Leads us to think we need abstraction from Symfony
- Improving Symfony is easier, requires less time

Goal: Defer complex abstractions

Requirements for Incremental Design

- Testability
- Refactorabillity
- Loose coupling
- Small number of dependencies

Quintessential Symfony Controller Example

```php
public function editAction($id, Request $request)
{
    $entityManager = $this->get('doctrine.orm.default_entity_manager');
    $repository = $entityManager->getRepository('AcmeHelloBundle:Article');

    try {
        $article = $repository->findQuery($id);
    } catch (NoResultException $e) {
        throw new NotFoundHttpException();
    }

    if (!$this->get('security.context')->isGranted('EDIT', $article)) {
        throw new AccessDeniedHttpException();
    }

    $form = $this->createForm(new EditArticleType(), $article);
    $form->handleRequest($request);

    if ($form->isBound() && $form->isValid()) {
        $entityManager->flush();

        $this->get('session')->getFlashBag()->add(
            'notice',
            'Your_changes_were_saved!'
        );

        return $this->redirect($this->generateUrl('Article.show', $id));
    }

    return $this->render(
        'AcmeHelloBundle:Article:edit.html.twig',
        array(
            'form' => $form->createView(),
            'article' => $article,
```

Quintessential Symfony Controller Example

```
1  public function editAction($id, Request $request)
2  {
3      $entityManager = $this
4          ->get('doctrine.orm.default_entity_manager');
5      $repository = $entityManager
6          ->getRepository('AcmeHelloBundle:Article');
7
8      try {
9          $article = $repository->findQuery($id);
10     } catch (NoResultException $e) {
11         throw new NotFoundHttpException();
12     }
13     //..
14 }
```

Quintessential Symfony Controller Example

```
1  public function editAction($id, Request $request)
2  {
3      //..
4
5      $security = $this->get('security.context');
6      if (!$security->isGranted('EDIT', $article)) {
7          throw new AccessDeniedHttpException();
8      }
9
10     //..
11 }
```

Quintessential Symfony Controller Example

```
1  public function editAction($id, Request $request)
2  {
3      //...
4      $form = $this->createForm(
5          new EditArticleType(), $article
6      );
7      $form->handleRequest($request);
8
9      if ($form->isBound() && $form->isValid()) {
10         //  ...
11     }
12
13     //...
14 }
```

Quintessential Symfony Controller Example

```
1  $entityManager->flush();
2
3  $this->get('session')->getFlashBag()->add(
4      'notice',
5      'Your changes were saved!'
6  );
7
8  return $this->redirect(
9      $this->generateUrl('Article.show', $id)
10 );
```

Quintessential Symfony Controller Example

```
1  public function editAction($id, Request $request)
2  {
3      //..
4      return $this->render(
5          'AcmeHelloBundle:Article:edit.html.twig',
6          array(
7              'form' => $form->createView(),
8              'article' => $article,
9          )
10     );
11 }
```

Painful!

Unfortunately the document is made in this style

Problems

- Coupling: 7 services, 3 exceptions and 1 entity class
- Even with impressive typing skills the time lost typing matters.
- Abstracting Forms, Security, Doctrine is alot of work
- Number of bugs are correlated to lines of code
- And this is just a CRUD example (no real behavior involved!)

Solution?

- Best Practices suggest Annotations
- But this just moves the lines/problem elsewere
- I don't trust them, especially not with security.

Solution?

With some simple improvements on top of Symfony we can:

- cut down the number of services to one
- remove all exception code
- cut number of lines by half
- still keep the Symfony style

Two Approaches

- Move code related to response generation into listeners.
- Move code related to request/state into param converters.

Contrary to Best Practices claim, the overhead is neglectible

Template EventListener

Goal: Getting rid of the `"templating"` service dependency:

- Return array from controller
- Automatically convert to rendering a template
- Use naming convention to find the template
- Return `new TemplateView()` to pick a different template.
- Avoids having to use annotations.

Template EventListener

```
1  public function editAction($id /*, ...*/)
2  {
3      return array(
4          'article' => $article,
5          'form' => $form->createView()
6      );
7  }
```

Template EventListener

```
1  use QafooLabs\MVC\TemplateView;
2
3  public function editAction($id /*, ...*/)
4  {
5      return new TemplateView(
6          "AcmeHelloBundle:Article:form.html.twig",
7          array(
8              'article' => $article,
9              'form' => $form->createView()
10         )
11     );
12 }
```

Redirect Listener

Goal: Getting rid of the `"router"` service dependency:

- The router is needed in controllers for redirecting
- Introduce `new RedirectRouteResponse()`

Redirect Listener

```
1  use QafooLabs\MVC\RedirectRouteResponse;
2
3  public function editAction($id /*, ... */)
4  {
5      // ...
6
7      return new RedirectRouteResponse(
8          'Article.show',
9          array('id' => $id)
10     );
11 }
```

Convert Exceptions

Goal: Getting rid of Exception casting and handling in controller.

- Introduce configuration to map exceptions to status codes.
- Example: Doctrine NoResultException is always a 404
- This reuses functionality that was only available for REST APIs before

Convert Exceptions

```
1  qafoo_labs_no_framework:
2      convert_exceptions:
3          "Doctrine\\ORM\\NoResultException": 404
```

Handling Flash-Messages

Goal: Get rid of the `"session"` service dependency.

- Introduce ParamConverter that passes the flash state into controller.
- Typehint for `Flashes`
- Suddenly turns into data object, not a service anymore.
- Applies learning from Request as a service failure

Handling Flash-Messages

```
1  public function editAction(/* ... */, Flashes $flashes)
2  {
3      // ..
4      $flashes->add('notice', 'Your changes were saved');
5      //...
6  }
```

Handling Forms

Goal: Get rid of the `"form.factory"` service dependency.

- Introduce ParamConverter that passes new FormRequest
- `FormRequest` wraps both `Request` and `FormFactory`
- Typehint for `FormRequest`
- Forms turns into data object, not a service anymore.

Handling Forms

```
1  public function editAction(/*.*/, FormRequest $request)
2  {
3      $type = new EditArticleType();
4      if ($formRequest->handle($type, $article)) {
5          // ...
6      }
7
8      return array(
9          // ...
10         'form' => $formRequest->createView()
11     );
12 }
```

Handling Security

Goal: Get rid of the `"security.context"` service dependency.

- Security Token is a data object
- Data should be passed into functions as argument.
- Typehint for `FrameworkContext`
- Does someone have a better name?
- Applies learning from Request as a service failure

Handling Security

```
1  public function editAction(FrameworkContext $context)
2  {
3      // ...
4      $context->assertIsGranted('EDIT', $article);
5      // ...
6  }
```

Controller as a Service

Goal: Get rid of the `"container"` dependency

- Inject only the services that you need
- Controversial: Fabien does not approve
- `http://symfony.com/doc/current/cookbook/controller/service.html`

Controller as a Service

```php
<?php

class ArticleController
{
    /**
     * @var ArticleRepository
     */
    private $repository;

    public function editAction($id, /*.*/)
    {
        $article = $this->repository->findQuery($id);

        //..
    }
}
```

Result

```
1   public function editAction($id, FormRequest $request, FrameworkContext $context,
        Flashes $flashes)
2   {
3       $article = $this->repository->findQuery($id);
4       $context->assertIsGranted('EDIT', $article);
5
6       $type = new EditArticleType();
7       if ($formRequest->handle($type, $article)) {
8           $this->repository->save($article);
9
10          $flashes->add('notice', '...');
11
12          return new RedirectRouteResponse(/*..*/);
13      }
14
15      return array('article' => $article,
16          'form' => $formRequest->createView());
17  }
```

Why?

- No service-layer needed in the beginning
- Treat contoller as service layer
- Refactoring towards services is simple
- Extract code into services
- Very small number of service dependencies
- Testable controllers

Testing?

```
1   public function testEdit()
2   {
3       $repo = $this->getMock(ArticleRepository::class);
4       $repo->expects($this->once())->method('find')
5           ->will($this->returnValue(new Article));
6       $repo->expects($this->once())->method('save');
7
8       $controller = new ArticleController($repo);
9
10      $response = $controller->editAction(
11          $articleId = 10,
12          new ValidFormRequest(),
13          new GrantAllContext(),
14          new FlashMock()
15      );
16
17      $this->assertInstanceOf(RedirectRouteResponse::class, $response);
18  }
```

https://github.com/QafooLabs/QafooLabsNoFrameworkBundle

THANK YOU

Rent a quality expert
qafoo.com