Symfony DI in 2017

oder: Wie ich Autowiring lieben lernte.

Alexander M. Turek

Dependency Injection Container

- Haupt-Erweiterungspunkt für Frameworks- und Anwendungsfunktionalität.
- Stellt sog. Services bereit, welche…
  - … erst bei Bedarf erzeugt werden und…
  - … zur weiteren Verwendung vorgehalten werden.
- Container wird kompiliert!

Dependency Injection

```php
class MyService
{
    public function doSometing()
    {
        $helper = new MyHelper();
        $helper->helpMe();
    }
}
```

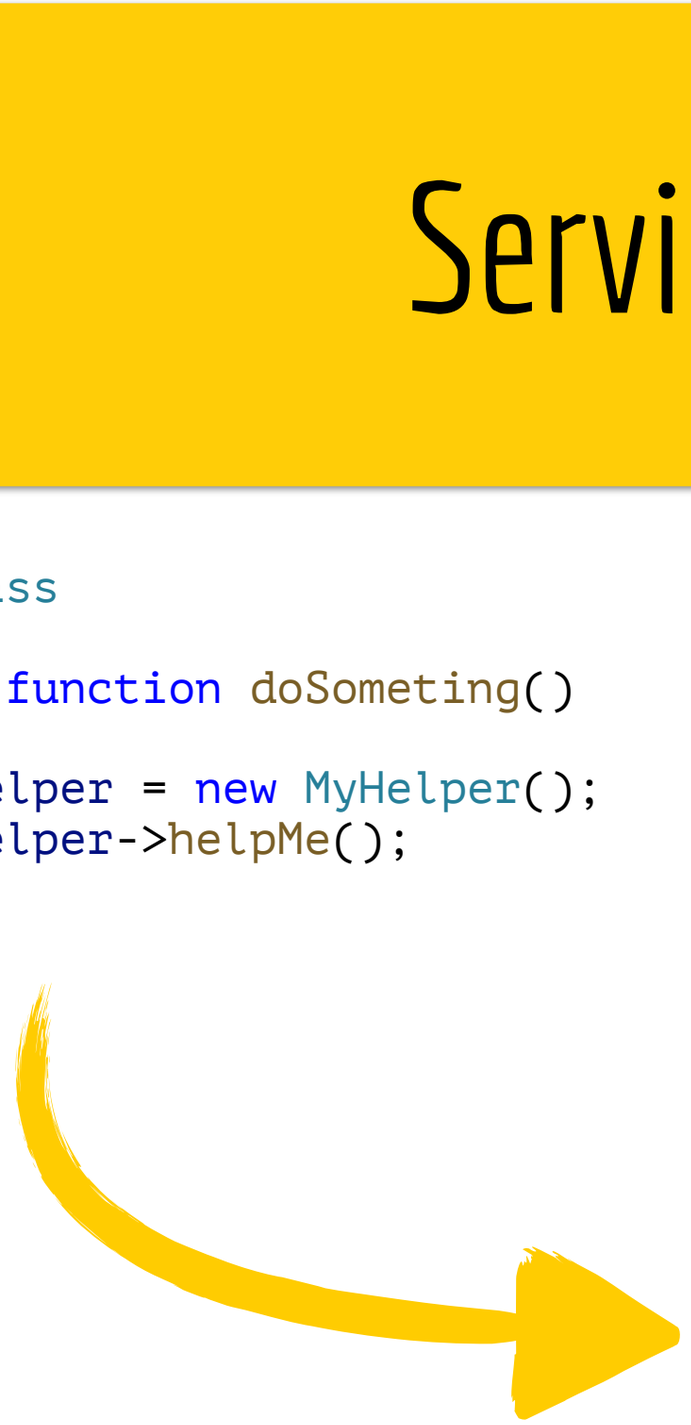
```php
class MyService
{
    private $helper;

    public function __construct(MyHelper $helper)
    {
        $this->helper = $helper;
    }

    public function doSometing()
    {
        $this->helper->helpMe();
    }
}
```

Service Location

```php
class MyClass
{
    public function doSometing()
    {
        $helper = new MyHelper();
        $helper->helpMe();
    }
}
```



```php
class MyClass
{
    public function doSometing()
    {
        $helper = Locator::locate('my-helper');
        $helper->helpMe();
    }
}
```

Der Service Container

- So haben wir ihn kennengelernt:

```php
class MyController extends Controller
{
    public function myAction()
    {
        $myService = $this->get('my_service');
        $myService->doSomething();

        // ...
    }
}
```

services.yml

```php
class MyService
{
    private $helper;

    public function __construct(MyHelper $helper)
    {
        $this->helper = $helper;
    }

    public function doSometing()
    {
        $this->helper->helpMe();
    }
}
```

```yaml
imports:
    - { resource: services/*.yml }

services:
  my_service:
    class: Acme\MyApplication\Util\MyService
    arguments:
      - '@my_helper'

  my_helper:
    class: Acme\MyApplication\Util\MyHelper
    public: false
```

Public/Private Services

- Public Services können über `Container::get()` abgerufen werden.
- Private Services können nur als Abhängigkeit anderer Services verwandt werden.

Kompilierung und Optimierung

- Compiler baut Factory-Methoden für alle Services.
- Private Services, die nicht konsumiert werden, werden eliminiert.
- Private Services, die einmalig konsumiert werden, werden eingebettet.

Private by Default

- Änderungen in Symfony 4:
  - Viele interne Services des Frameworks sind nun als private deklariert.
  - Services, die keine Sichtbarkeit definieren, sind nun automatisch private!
  - Migrationspfad: Sichtbarkeit immer deklarieren.

```yaml
services:
  my_request_listener:
    class: Acme\MyApplication\EventListener\MyRequestListener
    public: false
    arguments:
      - '@security.token_storage'
      - '@router'
    calls:
      - [ setLogger, [ '@logger' ]]
    tags:
      - { name: kernel.event_subscriber }

  my_view_listener:
    class: Acme\MyApplication\EventListener\MyViewListener
    public: false
    arguments:
      - '@twig'
      - '@router'
      - '%kernel.project_dir%'
      - '%kernel.debug%'
    tags:
      - { name: kernel.event_subscriber }

my_response_listener:
  class: Acme\MyApplication\EventListener\MyExceptionListener
  public: false
  arguments:
    - '@twig'
    - '@router'
    - '%kernel.debug%'
  tags:
    - { name: kernel.event_subscriber }
```

Verbosity!

„Wenn man Konfiguration über Konvention stellt und unerwartete Magie vermeiden möchte, dann müssen Konfigurationsdateien wohl so aussehen.“

Ich, 2016

```yaml
services:
  my_request_listener:
    class: Acme\MyApplication\EventListener\MyRequestListener
    public: false
    arguments:
      - '@security.token_storage'
      - '@router'
    calls:
      - [ setLogger, [ '@logger' ]]
    tags:
      - { name: kernel.event_subscriber }

  my_view_listener:
    class: Acme\MyApplication\EventListener\MyViewListener
    public: false
    arguments:
      - '@twig'
      - '@router'
      - '%kernel.project_dir%'
      - '%kernel.debug%'
    tags:
      - { name: kernel.event_subscriber }

my_response_listener:
  class: Acme\MyApplication\EventListener\MyExceptionListener
  public: false
  arguments:
    - '@twig'
    - '@router'
    - '%kernel.debug%'
  tags:
    - { name: kernel.event_subscriber }
```

```yaml
services:
  _defaults:
    public: false
    tags:
      - { name: kernel.event_subscriber }

  my_request_listener:
    class: Acme\MyApplication\EventListener\MyRequestListener
    arguments:
      - '@security.token_storage'
      - '@router'
    calls:
      - [ setLogger, [ '@logger' ]]

  my_view_listener:
    class: Acme\MyApplication\EventListener\MyViewListener
    arguments:
      - '@twig'
      - '@router'
      - '%kernel.project_dir%'
      - '%kernel.debug%'

my_response_listener:
  class: Acme\MyApplication\EventListener\MyExceptionListener
  arguments:
    - '@twig'
    - '@router'
    - '%kernel.debug%'
```

_defaults

- Symfony 3.3
- Ermöglicht Einstellungen für alle Services derselben Konfigurationsdatei vorzunehmen.
- Alle Einstellungen können in den einzelnen Services wieder überschrieben werden.

```yaml
_defaults:
  public: false
  tags:
    - { name: kernel.event_subscriber }
```

Naming Things

„Wenn man grundsätzliche dieselbe Klasse in mehreren Konfigurationen verwenden können möchte, dann kann ein Service nicht allein über den FQCN definiert werden.“

Ebenfalls ich, 2016

```yaml
services:
  _defaults:
    public: false
    tags:
      - { name: kernel.event_subscriber }

  my_request_listener:
    class: Acme\MyApplication\EventListener\MyRequestListener
    arguments:
      - '@security.token_storage'
      - '@router'
    calls:
      - [ setLogger, [ '@logger' ]]

  my_view_listener:
    class: Acme\MyApplication\EventListener\MyViewListener
    arguments:
      - '@twig'
      - '@router'
      - '%kernel.project_dir%'
      - '%kernel.debug%'

my_response_listener:
  class: Acme\MyApplication\EventListener\MyExceptionListener
  arguments:
    - '@twig'
    - '@router'
    - '%kernel.debug%'
```

```yaml
services:
  _defaults:
    public: false
    tags:
      - { name: kernel.event_subscriber }

  Acme\MyApplication\EventListener\MyRequestListener:
    arguments:
      - '@security.token_storage'
      - '@router'
    calls:
      - [ setLogger, [ '@logger' ]]

  Acme\MyApplication\EventListener\MyViewListener:
    arguments:
      - '@twig'
      - '@router'
      - '%kernel.project_dir%'
      - '%kernel.debug%'

Acme\MyApplication\EventListener\MyExceptionListener:
  arguments:
    - '@twig'
    - '@router'
    - '%kernel.debug%'
```

Service ID vs. FQCN

- Klassennamen mit Namespace (FQCN) sind nun als Service-IDs zulässig.
- Symfony 3.3
- In diesem Fall ist das Attribut „class“ optional.
- Best Practise: Öffentliche Services mit expliziter Service-ID versehen.

```yaml
services:
  _defaults:
    public: false

my_service:
  class: Acme\MyApplication\Util\MyService
  public: true
  arguments:
    - '@Acme\MyApplication\Util\MyHelper'

Acme\MyApplication\Util\MyHelper: ~
```

Type-Hints

```php
class MyRequestListener implements EventSubscriberInterface, LoggerAwareInterface
{
    use LoggerAwareTrait;

    private $tokenStorage;
    private $router;

    public function __construct(
        TokenStorageInterface $tokenStorage,
        UrlGeneratorInterface $router
    ) {
        $this->tokenStorage = $tokenStorage;
        $this->router = $router;

        $this->setLogger(new NullLogger());
    }

    // ...

}
```

```yaml
services:
  _defaults:
    public: false
    tags:
      - { name: kernel.event_subscriber }

  Acme\MyApplication\EventListener\MyRequestListener:
    arguments:
      - '@security.token_storage'
      - '@router'
    calls:
      - [ setLogger, [ '@logger' ]]

  Acme\MyApplication\EventListener\MyViewListener:
    arguments:
      - '@twig'
      - '@router'
      - '%kernel.project_dir%'
      - '%kernel.debug%'

Acme\MyApplication\EventListener\MyExceptionListener:
  arguments:
    - '@twig'
    - '@router'
    - '%kernel.debug%'
```

```yaml
services:
  _defaults:
    public: false
    tags:
      - { name: kernel.event_subscriber }

  Acme\MyApplication\EventListener\MyRequestListener:
    autowire: true
    calls:
      - [ setLogger, [ '@logger' ]]

  Acme\MyApplication\EventListener\MyViewListener:
    arguments:
      - '@twig'
      - '@router'
      - '%kernel.project_dir%'
      - '%kernel.debug%'

  Acme\MyApplication\EventListener\MyExceptionListener:
    arguments:
      - '@twig'
      - '@router'
      - '%kernel.debug%'
```

Autowiring

- Symfony 3.3
- Compiler versucht, unspezifizierte Konstruktor-Parameter über den Type-Hint zu verdrahten.
- Regel: Wenn es einen Service gibt, dessen ID dem Type-Hint entspricht, wird dieser als Abhängigkeit gesetzt.

Interface als Alias

```
$ ./bin/console debug:container 'Symfony\Component\Routing\Generator\UrlGeneratorInterface'

 // This service is an alias for the service router.default

Information for Service "router.default"
========================================

 ---------------- ------------------------------------------
  Option           Value
 ---------------- ------------------------------------------
  Service ID       router.default
  Class            Symfony\Bundle\FrameworkBundle\Routing\Router
  Tags             monolog.logger (channel: router)
  Calls            setConfigCacheFactory
  Public           no
  Synthetic        no
  Lazy             no
  Shared           yes
  Abstract         no
  Autowired        no
  Autoconfigured   no
 ---------------- ------------------------------------------
```

```yaml
services:
  _defaults:
    public: false
    tags:
      - { name: kernel.event_subscriber }

  Acme\MyApplication\EventListener\MyRequestListener:
    autowire: true
    calls:
      - [ setLogger, [ '@logger' ]]

  Acme\MyApplication\EventListener\MyViewListener:
    arguments:
      - '@twig'
      - '@router'
      - '%kernel.project_dir%'
      - '%kernel.debug%'

  Acme\MyApplication\EventListener\MyExceptionListener:
    arguments:
      - '@twig'
      - '@router'
      - '%kernel.debug%'
```

```yaml
services:
  _defaults:
    public: false
    autowire: true
    tags:
      - { name: kernel.event_subscriber }

  Acme\MyApplication\EventListener\MyRequestListener:
    calls:
      - [ setLogger, [ '@logger' ]]

  Acme\MyApplication\EventListener\MyViewListener:
    arguments:
      $projectDir: '%kernel.project_dir%'
      $debug:      '%kernel.debug%'

  Acme\MyApplication\EventListener\MyExceptionListener:
    arguments:
      $debug:      '%kernel.debug%'
```

Named Parameters

- Symfony 3.3
- Früher mussten stets alle Parameter spezifiziert werden.
- Jetzt können gezielt einzelne Konstruktor-Parameter spezifiziert werden.
- Dadurch wird u.a. partielles Autowiring ermöglicht.

```yaml
services:
  _defaults:
    public: false
    autowire: true
    tags:
      - { name: kernel.event_subscriber }

  Acme\MyApplication\EventListener\MyRequestListener:
    calls:
      - [ setLogger, [ '@logger' ]]

  Acme\MyApplication\EventListener\MyViewListener:
    arguments:
      $projectDir: '%kernel.project_dir%'
      $debug:      '%kernel.debug%'

  Acme\MyApplication\EventListener\MyExceptionListener:
    arguments:
      $debug:      '%kernel.debug%'
```

```yaml
services:
  _defaults:
    public: false
    autowire: true
    bind:
      $projectDir: '%kernel.project_dir%'
      $debug:      '%kernel.debug%'
    tags:
      - { name: kernel.event_subscriber }

  Acme\MyApplication\EventListener\MyRequestListener:
    calls:
      - [ setLogger, [ '@logger' ]]

  Acme\MyApplication\EventListener\MyViewListener: ~
  Acme\MyApplication\EventListener\MyResponseListener: ~
```

Bind (I)

- Symfony 3.4
- Erlaubt Autowiring von Konstruktor-Argumenten über den Namen des Arguments.
- Hilfreich, wenn mehrere Services in der gleichen Datei dasselbe Argument definieren.

```yaml
services:
  _defaults:
    public: false
    autowire: true
    bind:
      $projectDir: '%kernel.project_dir%'
      $debug:      '%kernel.debug%'
    tags:
      - { name: kernel.event_subscriber }

  Acme\MyApplication\EventListener\MyRequestListener:
    calls:
      - [ setLogger, [ '@logger' ]]

  Acme\MyApplication\EventListener\MyViewListener: ~
  Acme\MyApplication\EventListener\MyExceptionListener: ~
```

```yaml
services:
  _defaults:
    public: false
    autowire: true
    bind:
      $projectDir: '%kernel.project_dir%'
      $debug:      '%kernel.debug%'
    tags:
      - { name: kernel.event_subscriber }

  _instanceof:
    Psr\Log\LoggerAwareInterface:
      calls:
        - [ setLogger, [ '@logger' ]]

  Acme\MyApplication\EventListener\MyRequestListener: ~
  Acme\MyApplication\EventListener\MyViewListener: ~
  Acme\MyApplication\EventListener\MyExceptionListener: ~
```

_instanceof

- Symfony 3.3
- Einstellungen gelten für alle Services, die in derselben Datei definiert werden, sofern sie ein bestimmtes Interface implementieren oder Instanz einer bestimmten Klasse sind.
- Call kann für einzelnen Services überschrieben werden.

```yaml
services:
  _defaults:
    public: false
    autowire: true
    bind:
      $projectDir: '%kernel.project_dir%'
      $debug:      '%kernel.debug%'
    tags:
      - { name: kernel.event_subscriber }

  _instanceof:
    Psr\Log\LoggerAwareInterface:
      calls:
        - [ setLogger, [ '@logger' ]]

  Acme\MyApplication\EventListener\MyRequestListener: ~
  Acme\MyApplication\EventListener\MyViewListener: ~
  Acme\MyApplication\EventListener\MyExceptionListener: ~
```

```yaml
services:
  _defaults:
    public: false
    autowire: true
    autoconfigure: true
    bind:
      $projectDir: '%kernel.project_dir%'
      $debug:      '%kernel.debug%'

  _instanceof:
    Psr\Log\LoggerAwareInterface:
      calls:
        - [ setLogger, [ '@logger' ]]

  Acme\MyApplication\EventListener\MyRequestListener: ~
  Acme\MyApplication\EventListener\MyViewListener: ~
  Acme\MyApplication\EventListener\MyExceptionListener: ~
```

Autoconfiguration (I)

- Symfony 3.3
- Wenn ein Service bestimmte Interfaces implementiert oder Klassen erweitert, wird er automatisch mit einem Tag versehen.
- Kann Spuren von schwarzer Magie enthalten.

```yaml
services:
  _defaults:
    public: false
    autowire: true
    autoconfigure: true
    bind:
      $projectDir: '%kernel.project_dir%'
      $debug:      '%kernel.debug%'

  _instanceof:
    Psr\Log\LoggerAwareInterface:
      calls:
        - [ setLogger, [ '@logger' ]]

  Acme\MyApplication\EventListener\MyRequestListener: ~
  Acme\MyApplication\EventListener\MyViewListener: ~
  Acme\MyApplication\EventListener\MyExceptionListener: ~
```

```yaml
services:
  _defaults:
    public: false
    autowire: true
    bind:
      $projectDir: '%kernel.project_dir%'
      $debug:      '%kernel.debug%'
    tags:
      - { name: kernel.event_subscriber }

  _instanceof:
    Psr\Log\LoggerAwareInterface:
      calls:
        - [ setLogger, [ '@logger' ]]

  Acme\MyApplication\EventListener\:
    resource: '%kernel.project_dir%/src/EventListener/*'
```

PSR-4 Service Discovery

- Alle PHP-Dateien in einem Verzeichnis werden rekursiv als Service registriert.
- Benamung nach PSR-4 wird angenommen.
- Automatisch registrierte Services können überschrieben werden.
- Best Practise: Nur für private Services verwenden.

```php
class MyExceptionListener implements EventSubscriberInterface
{
    private $twig;
    private $router;
    private $debug;

    public function __construct(
        Environment $twig,
        UrlGeneratorInterface $router,
        bool $debug = false
    ) {
        $this->twig = $twig;
        $this->router = $router;
        $this->debug = $debug;
    }

    public static function getSubscribedEvents()
    {
        return [KernelEvents::EXCEPTION => 'onKernelException'];
    }

    public function onKernelException(GetResponseForExceptionEvent $e)
    {
        if (!$e->getException() instanceof MyException) {
            return;
        }

        // ...
    }
}
```

```php
class MyExceptionListener implements EventSubscriberInterface
{
    private $container;
    private $debug;

    public function __construct(ContainerInterface $container, bool $debug = false)
    {
        $this->container = $container;
        $this->debug = $debug;
    }

    public static function getSubscribedEvents()
    {
        return [KernelEvents::EXCEPTION => 'onKernelException'];
    }

    public function onKernelException(GetResponseForExceptionEvent $e)
    {
        if (!$e->getException() instanceof MyException) {
            return;
        }

        // ...
    }

    private function getTwig(): Environment
    {
        return $this->container->get('twig');
    }

    private function getRouter(): UrlGeneratorInterface
    {
        return $this->container->get('router');
    }
}
```

```yaml
services:
  _defaults:
    public: false
    autowire: true
    autoconfigure: true
    bind:
      $projectDir: '%kernel.project_dir%'
      $debug:      '%kernel.debug%'
    tags:
      - { name: kernel.event_subscriber }

  _instanceof:
    Psr\Log\LoggerAwareInterface:
      calls:
        - [ setLogger, [ '@logger' ]]

  Acme\MyApplication\EventListener\:
    resource: '%kernel.project_dir%/src/EventListener/*'

  Acme\MyApplication\EventListener\MyViewListener:
    arguments:
      $container: '@event.service_locator'

Acme\MyApplication\EventListener\MyExceptionListener:
  arguments:
    $container: '@event.service_locator'

event.service_locator:
  class: Symfony\Component\DependencyInjection\ServiceLocator
  tags:
    - { name: container.service_locator }
  arguments:
      -
        twig: '@twig'
        router: '@router'
```

Service Locator

- Erlaubt es eine Teilmenge der verfügbaren Services als Container zur Verfügung zu stellen.
- Kann private Services verfügbar machen.
- Entkoppelung via PSR-11.

```yaml
services:
  _defaults:
    public: false
    autowire: true
    autoconfigure: true
    bind:
      $projectDir: '%kernel.project_dir%'
      $debug:      '%kernel.debug%'
    tags:
      - { name: kernel.event_subscriber }

  _instanceof:
    Psr\Log\LoggerAwareInterface:
      calls:
        - [ setLogger, [ '@logger' ]]

  Acme\MyApplication\EventListener\:
    resource: '%kernel.project_dir%/src/EventListener/*'

  Acme\MyApplication\EventListener\MyViewListener:
    arguments:
      $container: '@event.service_locator'

Acme\MyApplication\EventListener\MyExceptionListener:
  arguments:
    $container: '@event.service_locator'

event.service_locator:
  class: Symfony\Component\DependencyInjection\ServiceLocator
  tags:
    - { name: container.service_locator }
  arguments:
      -
        twig: '@twig'
        router: '@router'
```

```yaml
services:
  _defaults:
    public: false
    autowire: true
    autoconfigure: true
    bind:
      $projectDir:                    '%kernel.project_dir%'
      $debug:                         '%kernel.debug%'
      Psr\Container\ContainerInterface: '@event.service_locator'
    tags:
      - { name: kernel.event_subscriber }

  _instanceof:
    Psr\Log\LoggerAwareInterface:
      calls:
        - [ setLogger, [ '@logger' ]]

  Acme\MyApplication\EventListener\:
    resource: '%kernel.project_dir%/src/EventListener/*'

  event.service_locator:
    class: Symfony\Component\DependencyInjection\ServiceLocator
    tags:
      - { name: container.service_locator }
    arguments:
        -
          twig: '@twig'
          router: '@router'
```

Bind (II)

- Symfony 3.4
- Definiert lokale Alias für das Autowiring von Interfaces/Klassen in derselben Datei.

```yaml
services:
  _defaults:
    public: false
    autowire: true

  my_import:
    class: Acme\MyApplication\Import\MyImport
    public: true
    arguments:
      $mappers:
        - '@Acme\MyApplication\Import\Mapper\MasterDataMapper'
        - '@Acme\MyApplication\Import\Mapper\PricingMapper'
        - '@Acme\MyApplication\Import\Mapper\MarketingCampaignMapper'
        - '@Acme\MyApplication\Import\Mapper\RelatedProductsMapper'

Acme\MyApplication\Import\Mapper\:
  resource: '%kernel.project_dir%/src/Import/Mapper/*'
```

```yaml
services:
  _defaults:
    public: false
    autowire: true

  my_import:
    class: Acme\MyApplication\Import\MyImport
    public: true
    arguments:
      $mappers: []

Acme\MyApplication\Import\Mapper\:
  resource: '%kernel.project_dir%/src/Import/Mapper/*'
  tags:
    - { name: acme.import_mapper }
```

```php
namespace Acme\MyApplication\DependencyInjection;

use Symfony\Component\DependencyInjection\Compiler\CompilerPassInterface;
use Symfony\Component\DependencyInjection\ContainerBuilder;
use Symfony\Component\DependencyInjection\Reference;

class ImportCompilerPass implements CompilerPassInterface
{
    public function process(ContainerBuilder $container)
    {
        $mappers = [];
        $taggedServices = array_keys(
            $container->findTaggedServiceIds('acme.import_mapper')
        );

        foreach ($taggedServices as $id) {
            $mappers[] = new Reference($id);
        }

        $container->findDefinition('my_import')
            ->replaceArgument('$mappers', $mappers);
    }
}
```

Compiler Pass

- Erlaubt Eingriffe in die Kompilierung des Containers.
- Wird vom Framework selbst stark genutzt:
  - Event Subscribers
  - Autowiring
  - Container-Optimierung

```yaml
services:
  _defaults:
    public: false
    autowire: true

  my_import:
    class: Acme\MyApplication\Import\MyImport
    public: true
    arguments:
      $mappers: []

Acme\MyApplication\Import\Mapper\:
  resource: '%kernel.project_dir%/src/Import/Mapper/*'
  autoconfigure: true
```

```php
namespace Acme\MyApplication;

use Acme\MyApplication\DependencyInjection\ImportCompilerPass;
use Acme\MyApplication\Import\MapperInterface;
use Symfony\Component\DependencyInjection\ContainerBuilder;
use Symfony\Component\HttpKernel\Kernel;

class AppKernel extends Kernel
{
    // ...

    protected function build(ContainerBuilder $container)
    {
        $container->addCompilerPass(new ImportCompilerPass());
        $container->registerForAutoconfiguration(MapperInterface::class)
            ->addTag('acme.import_mapper');
    }
}
```

Autoconfiguration (II)

- Eigene Regeln für die Autokonfiguration können auf dem Container vor dem Kompilierschritt registriert werden.
- Achtung: Hier eingestellte Regeln gelten für alle Bundles (die das Feature verwenden)!
- Nur für eigene Interfaces registrieren.

```yaml
services:
  _defaults:
    public: false
    autowire: true

  my_import:
    class: Acme\MyApplication\Import\MyImport
    public: true
    arguments:
      $mappers: []

Acme\MyApplication\Import\Mapper\:
  resource: '%kernel.project_dir%/src/Import/Mapper/*'
  autoconfigure: true
```

```yaml
services:
  _defaults:
    public: false
    autowire: true

  my_import:
    class: Acme\MyApplication\Import\MyImport
    public: true
    arguments:
      $mappers: !tagged acme.import_mapper

Acme\MyApplication\Import\Mapper\:
  resource: '%kernel.project_dir%/src/Import/Mapper/*'
  tags:
    - { name: acme.import_mapper }
```

!tagged

- Symfony 3.4
- Erschlägt den wahrscheinlich häufigsten Use-Case für Compiler Passes.
- Alle Services, die mit einem bestimmten Tag versehen wurden, werden als Array eingefügt.
- Bessere Lesbarkeit.

```php
namespace Acme\MyApplication\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class MyController extends Controller
{
    public function myAction()
    {
        $myService = $this->get('my_service');
        $myService->doSomething();

        // ...
    }
}
```

```php
namespace Acme\MyApplication\Controller;

use Acme\MyApplication\Util\MyService;

class MyController
{
    private $myService;

    public function __construct(MyService $myService)
    {
        $this->myService = $myService;
    }

    public function myAction()
    {
        $this->myService->doSomething();

        // ...
    }
}
```

Controller as a Service

- Eigene Controller-Klasse muss nicht mehr von der abstrakten Controller-Klasse ableiten.
- Abhängigkeiten werden explizit.
- Zwingt zu sauberem Arbeiten.

CaaS: Best Practises

- Kleine fokussierte Controller bauen.
- Rendering in den View-Layer verlagern.
- Berechtigungsprüfung vorher durchführen.
- Param Converter/Argument Resolver verwenden.
- SensioFrameworkExtraBundle ist euer Freund.

Fazit

- DI-Konfiguration war immer sehr „ausführlich“.
- Symfony 3.3 und 3.4 führen „konfigurierbare Magie“ ein, um die Konfiguration zu vereinfachen.
- Symfony 4 setzt stärkeren Fokus auf kompaktere Container-Kompilate.

Danke

https://twitter.com/derrabus
https://github.com/derrabus
http://facebook.com/derrabus