DI & Service Container

Dependency injection & Services

```php
class Mailer
{
    private $transport;

    public function __construct()
    {
        $this->transport = 'mailchimp';
    }

    // ...
}

class NewsletterManager
{
    private $mailer;

    public function __construct()
    {
        $this->mailer = new Mailer;
    }

    // ...
}
```

```php
class Mailer
{
    private $transport;

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

    // ...
}

class NewsletterManager
{
    private $mailer;

    public function setMailer(\Mailer $mailer)
    {
        $this->mailer = $mailer;
    }

    // ...
}
```

Types of injection

- **Constructor injection** - recommended
- **Setter injections** - prevent circular references
- **Property injection** - bad practice

Dependency Injection

Newer instantiate the class manually,

Unless it's a data object

Dependency Injection Container

- Object that knows how to instantiate all the services of your application
- Lazy loaded
- Frozen after build

```
$container->get("doctrine")
```

DIC knows how to create the service

```
class srcProdProjectContainer extends Container
{
  /*
   * Gets the public 'doctrine' shared service.
   *
   * @return \Oro\Bundle\EntityBundle\ORM\Registry
   */
  protected function getDoctrineService()
  {
      return $this->services['doctrine'] =
          new \Oro\Bundle\EntityBundle\ORM\Registry(
            $this,
            $this->parameters['doctrine.connections'],
            $this->parameters['doctrine.entity_managers'],
            'default',
            'default'
      );
  }
  // ...
}
```

Service Container, PSR-11

```php
namespace Psr\Container;

interface ContainerInterface
{
  public function get($id);
  public function has($id): bool;
}
```

```php
namespace Symfony\Component\DependencyInjection;

interface ContainerInterface extends PsrContainerInterface
{
  public function get($id, $invalidBehavior = self::EXCEPTION_ON_INVALID_REFERENCE);
  //...
}
```

Symfony\Component\DependencyInjection\ServiceLocator

Private services

```yaml
# app/config/services.yml
services:
    # explicitly configure the service
    AppBundle\Service\MessageGenerator:
        public: true
```

```php
# var/cache/prod/Container6kkskkf/getDoctrine_DatabaseCreateCommandService.php
<?php

use Symfony\Component\DependencyInjection\Argument\RewindableGenerator;

// Returns the private 'doctrine.database_create_command' shared service.

include_once $this->targetDirs[3].'/vendor/symfony/symfony/src/Symfony/Component/Console/Command/Command.php';
include_once $this
  ->targetDirs[3].'/vendor/symfony/symfony/src/Symfony/Bundle/FrameworkBundle/Command/ContainerAwareCommand.php';
include_once $this->targetDirs[3].'/vendor/doctrine/doctrine-bundle/Command/DoctrineCommand.php';
include_once $this->targetDirs[3].'/vendor/doctrine/doctrine-bundle/Command/CreateDatabaseDoctrineCommand.php';

$this->services['doctrine.database_create_command'] = $instance =
  new \Doctrine\Bundle\DoctrineBundle\Command\CreateDatabaseDoctrineCommand();

$instance->setName('doctrine:database:create');

return $instance;
```

Parameters

- parameters.yml
- Can be used in service configuration
- Unused will be cleaned up during build of container
- The dot notation and array values
- Can be set before the container is compiled

Compiling Container

- $container->compile();
- $container->registerExtension($extension);
- ExtensionInterface
- Extension example
- CompilerPassInterface
- Compiler Pass example
- Registering compiler passes

Compiler Passes order

- PassConfig::TYPE_BEFORE_OPTIMIZATION
- PassConfig::TYPE_OPTIMIZE
- PassConfig::TYPE_BEFORE_REMOVING
- PassConfig::TYPE_REMOVE
- PassConfig::TYPE_AFTER_REMOVING

```
$containerBuilder->addCompilerPass(
    new CustomPass(),
    PassConfig::TYPE_AFTER_REMOVING
);
```

Tagged Services

```php
// src/AppBundle/Mail/TransportChain.php
namespace AppBundle\Mail;

class TransportChain
{
    private $transports;

    public function __construct()
    {
        $this->transports = array();
    }

    public function addTransport(\Swift_Transport $transport)
    {
        $this->transports[] = $transport;
    }
}
```

```yaml
services:
    Swift_SmtpTransport:
        arguments: ['%mailer_host%']
        tags:
            - { name: 'app.mail_transport', alias: 'smtp' }

    Swift_SendmailTransport:
        tags:
            - { name: 'app.mail_transport', alias: 'sendmail' }
```

CompilerPass

Tagged Services

```php
// src/AppBundle/HandlerCollection.php
namespace AppBundle;

class HandlerCollection
{
    public function __construct(iterable $handlers)
    {
    }
}
```

```yaml
# app/config/services.yml
services:
    AppBundle\Handler\One:
        tags: ['app.handler']

    AppBundle\Handler\Two:
        tags: ['app.handler']

    AppBundle\HandlerCollection:
        # inject all services tagged with app.handler as first argument
        arguments: [!tagged app.handler]
```

Overriding services

- Decoration: decorator, definition
- Overriding

Using a Factory

```yaml
# app/config/services.yml
services:
    # ...
    AppBundle\Email\NewsletterManager:
        # call the static method that creates the object
        factory: ['AppBundle\Email\NewsletterManagerStaticFactory', createNewsletterManager]
        # define the class of the created object
        class: AppBundle\Email\NewsletterManager
```

```php
class NewsletterManagerStaticFactory
{
    public static function createNewsletterManager()
    {
        $newsletterManager = new NewsletterManager();

        // ...

        return $newsletterManager;
    }
}
```

Service Configurators

```
app.newsletter_manager:
    # new syntax
    configurator: 'AppBundle\Mail\EmailConfigurator:configure'
    # old syntax
    configurator: ['@AppBundle\Mail\EmailConfigurator', configure]
```

Non Shared services

```yaml
# app/config/services.yml
services:
    AppBundle\SomeNonSharedService:
        shared: false
        # ...
```

Avoid writing code
dependent on the Container!

Lazy services & Circular references

```yaml
# app/config/services.yml
services:
    AppBundle\Twig\AppExtension:
        lazy:  true
```

- Setters
- Proxy services (lazy)
- Inject the container
- ServiceLink
- Service Locator

Service Locator

```yaml
# app/config/services.yml
services:
    app.command_handler_locator:
        class: Symfony\Component\DependencyInjection\ServiceLocator
        tags: ['container.service_locator']
        arguments:
            -
                AppBundle\FooCommand: '@app.command_handler.foo'
                AppBundle\BarCommand: '@app.command_handler.bar'
```

Abstract Services

```yaml
# config/services.yaml
services:
    App\Repository\BaseDoctrineRepository:
        abstract:  true
        arguments: ['@doctrine.orm.entity_manager']
        calls:
            - [setLogger, ['@logger']]

    App\Repository\DoctrineUserRepository:
        # extend the App\Repository\BaseDoctrineRepository service
        parent: App\Repository\BaseDoctrineRepository

    App\Repository\DoctrinePostRepository:
        parent: App\Repository\BaseDoctrineRepository

    # ...
```

Autowiring & Autoconfiguration

```yaml
# app/config/services.yml
services:
    # default configuration for services in *this* file
    _defaults:
        autowire: true
        autoconfigure: true
        public: false
        bind:
            $adminEmail: 'manager@example.com'
            $requestLogger: '@monolog.logger.request'
            Psr\Log\LoggerInterface: '@monolog.logger.request'

    # makes classes in src/AppBundle available to be used as services
    AppBundle\:
        resource: '../../src/AppBundle/*'
        # you can exclude directories or files
        # but if a service is unused, it's removed anyway
        exclude: '../../src/AppBundle/{Entity,Repository}'

    AppBundle\Updates\SiteUpdateManager:
        arguments:
            $adminEmail: '%admin_email%'
```

Debugging

- debug:autowiring
- debug:container
- debug:config

>> Demonstration