Services

The way you can re-use code in Symfony

By https://andreiabohner.org

Symfony 4.2

Fetching and using services

There are 2 ways:

1. Autowirable services

can be used as type-hints in your methods/constructors

Creating a service

By default, all services you create are autowirable

A service is just a class that does work!
When you create your services (you can name it and put the code wherever you want) they are also automatically added in the container and available for autowiring!

Using a service

In your controller, or in your own classes, you can "ask" for a service from the container by type-hinting an argument with the service's class or interface name.

```
// src/Controller/ProductController.php

use Psr\Log\LoggerInterface;

/**
 * @Route("/products")
 */
public function list(LoggerInterface $logger)
{
    $logger->info('Look! I just used a service');

    // ...
}
```

Symfony will automatically pass the service object matching this type

Adding dependencies in your service

```
use Symfony\Component\Cache\Adapter\AdapterInterface;

class MyService
{
    private $cache;

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

    // ...
}
```

Just pass them via constructor! The constructor's arguments are autowired!

Some autowirable services available

Internally, each service has a unique name, or "id"

when there´s more than one type hint for one service id, you can use either to get the exact same object

Service Id	Service Type Hint
cache.app	CacheItemPoolInterface AdapterInterface
cache.app.simple	CacheInterface
doctrine.dbal.default_connection	Connection
service_container	ContainerInterface
debug.event_dispatcher	EventDispatcherInterface
debug.file_link_formatter	FileLinkFormatter
doctrine	ManagerRegistry
doctrine.orm.default_entity_manager	EntityManagerInterface
file_locator	FileLocator
filesystem	Filesystem
http_kernel	HttpKernelInterface
kernel	KernelInterface
monolog.logger	LoggerInterface
doctrine.orm.default_entity_manager	ObjectManager
annotations.cached_reader	Reader
parameter_bag	ParameterBagInterface ContainerBagInterface
request_stack	RequestStack
router.default	UrlGeneratorInterface UrlMatcherInterface RequestContextAwareInterface RouterInterface
router.request_context	RequestContext
serializer	DecoderInterface EncoderInterface SerializerInterface NormalizerInterface DenormalizerInterface
serializer.normalizer.object	ObjectNormalizer
session.handler	SessionHandlerInterface
service_container	ContainerInterface
session	SessionInterface
session.flash_bag	FlashBagInterface
session.storage.native	SessionStorageInterface
twig	Twig_Environment Environment

The command:
`$ php bin/console debug:autowiring`
list all services available for autowiring

2. Explicitly configuring services and arguments

```
# config/services.yaml
services:

    site_update_manager.superadmin:
        class: App\Updates\SiteUpdateManager
        autowire: false
        arguments:
            - '@App\Service\MessageGenerator'
            - '@mailer'
            - 'superadmin@example.com'

    site_update_manager.normal_users:
        class: App\Updates\SiteUpdateManager
        autowire: false
        arguments:
            - '@App\Service\MessageGenerator'
            - '@mailer'
            - 'contact@example.com'

    App\Updates\SiteUpdateManager: '@site_update_manager.superadmin'
```

service's id

disable autowire for this service

manually wire all arguments

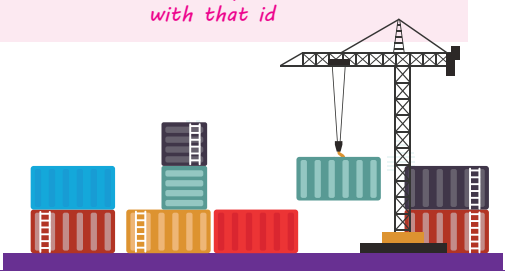
Create an alias, so if you type-hint SiteUpdateManager, the site_update_manager.superadmin will be used

Fetching a non-standard service

Access core services that cannot be autowired in your service via arguments:

```
services:
    App\Service\MyService:
        arguments:
            $logger: '@monolog.logger.email'
```

@ - used to pass the service with that id



By https://andreiabohner.org

A class that does work

Services

Symfony 4.2

all services are shared by default (i.e., each time you retrieve the service, you'll get the same instance)

Default configuration for services

All configurations below comes standard with every new Symfony project

config under services will affect all services in the application

```yaml
# config/services.yaml
services:
    _defaults:
        autowire: true
        autoconfigure: true
        public: false
```

default config values that should be applied to all services registered in this file

automatically injects dependencies in your services

automatically registers your services as commands, event subscribers, ...

allows optimizing the container by removing unused services; fetching services directly from the container via `$container->get()` won't work

makes classes in src/ available to be used as services, this creates a service per class whose id is the FQCN

```yaml
    App\:
        resource: '../src/*'
        exclude: '../src/{Entity,Migrations,Tests,Kernel.php}'
```

can be any valid glob pattern

is responsible for instantiating all the services

Service Container

where all services live!

Services are never instantiated until, and unless, someone asks for them.
Each service in the container is instantiated a maximum of once per request.

When you start a Symfony app, your container already contains many services.

E.g. of services:
log, email, database, security, cache, session

Other configuration for services

```yaml
# config/services.yaml
services:
    App\Foo:
        arguments:
            - !service
                class: App\AnonymousBar

    App\Service\OldService:
        deprecated: The "%service_id%" is deprecated since 2.8 and will be removed in 3.0.

    App\SomeNonSharedService:
        shared: false

    App\Service\EmailHelper:
        autowire: false

    bind:
        $emailLogger: '@monolog.logger.email'
        Some\Service: '@some.service'
```

anonymous service:
- prevent a service being used as a dependency of other services
- don't define an ID
- created where is used

deprecating a service

non shared service:
whenever you request the App\SomeNonSharedService, you will always get a new instance

disable autowire to this service (override the default config just for this service)

you can put the bind key under _defaults, services, or a specific service

you can bind by:
- argument name
- class or interface
- both above

if find any argument named $emailLogger, the monolog.logger.email will be passed to it

```php
public function __construct(LoggerInterface $emailLogger)
{
    $this->logger = $emailLogger;
}
```

Console

```
$ php bin/console debug:autowiring                                  show all services available for autowiring
$ php bin/console debug:container                                   full list of services available in the container
$ php bin/console debug:container 'App\Service\Mailer'              detailed info about a single service (you can use the service id too)
$ php bin/console debug:container 'App\Service\Mailer' --show-arguments   show the service arguments
$ php bin/console debug:container --show-hidden                     display "hidden services" (whose ID starts with a dot)
```