# Module 07 - Services et *Inversion Of Control* (IoC)

- Les **services** dans le pattern MVC ;
- Le composant *service container* : pattern inversion de contrôle (IoC) et l'injection de dépendances automatique (*autowiring*).

## Retour sur le modèle MVC

En principe, **le contrôleur a pour unique responsabilité de traiter la requête entrante et de retourner la réponse**.

Pour cela, il doit invoquer le bon *use-case*, la bonne logique métier et la bonne *vue* pour présenter le résultat demandé. **Il ne doit pas contenir lui-même de logique métier**. C'est la partie *flot de contrôle*, *orchestration* de notre application.

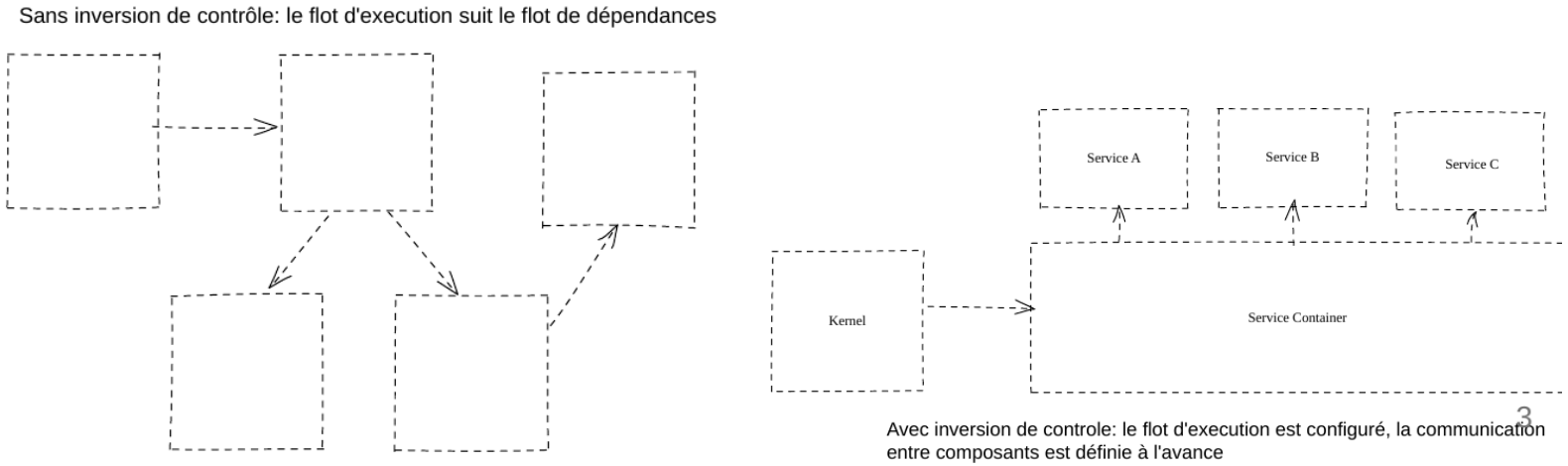
Le programme métier doit être délégué à des modules spécialisés : ces derniers font partie **du modèle**. Comme on l'a dit, **le modèle est indépendant du reste du système**.

Dans Symfony, et de manière générale, on appelle les modules métiers des **services**.

# L'inversion de contrôle (IoC)

L'Inversion of Control est un design pattern qui permet de **contrôler le flot d'exécution**.

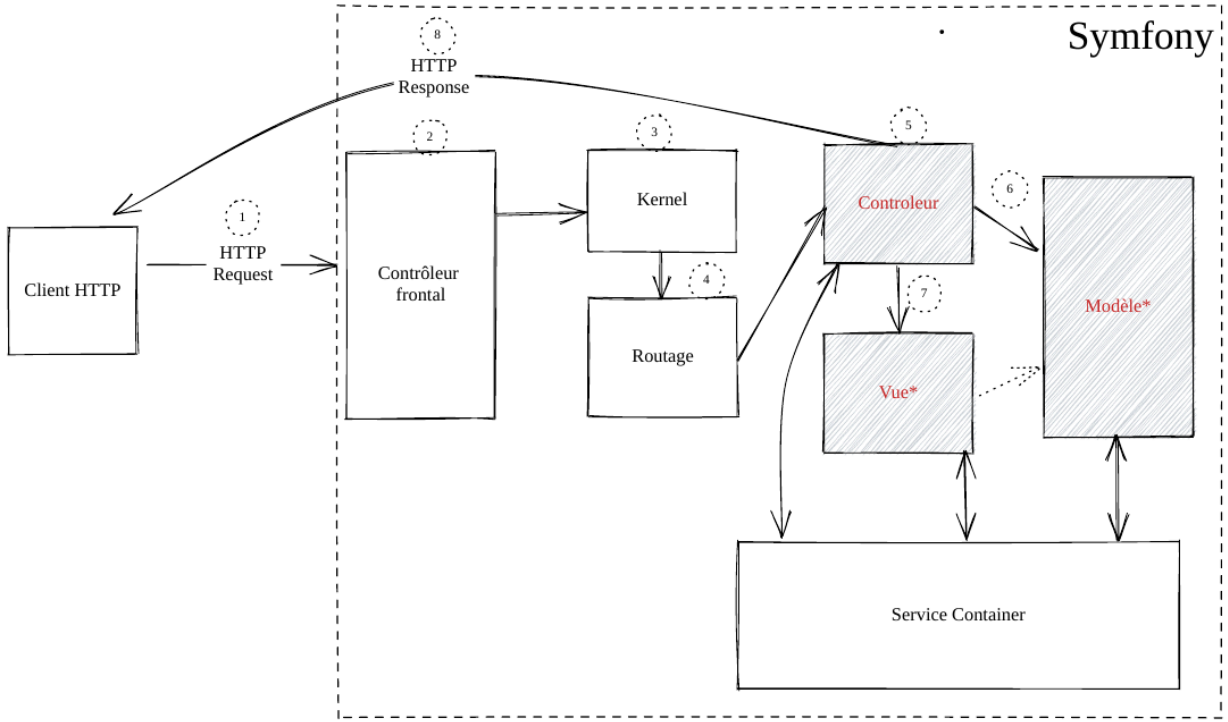
C'est ce que font la majorité des frameworks : ils vous *retirent* le flot de contrôle et les peines qui l'accompagnent (executer du code tout dans le bon ordre à chaque fois pour produire une requête http par exemple, encore appelé "code technique").




> Repensez à notre implémentation *vanilla* PHP du pattern MVC.

# Architecture de Symfony

Dans Symfony, c'est le **Service Container** qui joue ce rôle, le **composant central** du framework.



## L'inversion de contrôle (IoC)

**Avantages** :

- Réduit le code technique à écrire ;
- Diminue les dépendances entre vos composants et vous permet de les réutiliser plus facilement ;
- Améliore la lisibilité et la productivité ;
- Permet de mettre en place de l'**injection de dépendances automatique**.

## IoC et *Dependency Injection*

L'IoC permet de bénéficier de l'injection de dépendances.

L'injection de dépendances peut se faire:

- Via le constructeur;
- Via un setter.

## Le Service Container

Le Service Container est le composant central de Symfony. Il a le rôle suivant :

- Définir la manière d'instancier des classes ;
- Ne contient que des règles et possibilités d'injection ;
- Est une classe dont chaque méthode retourne une instance de classe (des services dans le cas de Symfony).

## Le Service Container

Rappelez-vous les contrôleurs :

```
#[Route("/some-url/{param1}")]
//L'objet de type Request est injecté par le Service container dans la méthode du contrôleur
public function mirror(Request $request, string $param1): Response
{...}
```

# Le Service Container

```
//Ici, l'instance $em de type EntityManagerInterface est fournie et injectée par le Service Container
#[Route('/books', name: 'list_books', methods: ['GET'])]
public function index(EntityManagerInterface $em): Response
{...}
```

# Auto-wiring

L'*auto-wiring*, ou chargement automatique, est assuré par le composant *Service Container* de Symfony.

Lorsque le chargement automatique est activé, **tous les services (classes) créés sont enregistrés en tant que service et instanciables n'importe où dans votre code**.

Les classes *services* sont déclarées dans le fichier `config/services.yaml`, sous la clef `services`.

```yaml
services:
    _defaults:
        autowire: true
 # Automatically injects dependencies in your services.
    App\:
    resource: '../src/'
```

Voir tous les *services* injectables dans les `Controller` :

```
php bin/console debug:autowiring
```

## Auto-wiring, magie noire ? Non !

L'*auto-wiring* est rendu possible par:

- L'autoloading du PSR-4 (association Nom Complètement Qualifié d'une Classe et path du fichier source de la classe)
- Le *type hinting* et l'API Reflection de PHP.

Le Service Container sait:

- **Où** chercher des services à charger (fichier de configuration `config/services.yaml`);
- **Le nom** complètement qualifié des services (PSR-4) ;
- **Comment** instancier les classes (API Reflection et *type hinting*)

À partir de ces ingrédients, **on peut générer du code pour instancier et injecter des objets**.

> Un bon exercice est d'implémenter un service container vous même pour faire de l'auto-wiring!

## Les services

- **Un Service = une instance d'une classe** (un objet). Dans Symfony, (presque) **tout est un service**.
- Un service n'est instancié que si on le demande explicitement ;
- Un service n'est **instancié qu'une fois** ;
- Les services sont **injectables dans d'autres services** ;

> Les contrôleurs sont également des services (un peu spéciaux) !

# Créer un service

Créer une classe dans le dossier src/Service (respectez bien PSR-4 !)

```php
// src/Service/MessageGenerator.php
namespace App\Service;

class MessageGenerator
{
    public function getHappyMessage(): string
    {
        $messages = [
            'You did it! You updated the system! Amazing!',
            'That was one of the coolest updates I\'ve seen all day!',
            'Great work! Keep going!',
        ];

        $index = array_rand($messages);

        return $messages[$index];
    }
}
```

Et *voilà* ! Le service peut être injecté dans n'importe quel contrôleur.

## Utilisation du service

Vérifier que le service est injectable automatiquement :

```
php bin/console debug:autowiring --all message_generator
```

# Utilisation du service

```php
// src/Controller/ProductController.php
use App\Service\MessageGenerator;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class ProductController extends AbstractController
{
    #[Route('/products/new')]
    //Une instance de MessageGenerator est injecté automatiquement dans le contrôleur !
    public function new(MessageGenerator $messageGenerator): Response
    {
        $message = $messageGenerator->getHappyMessage();
        // ...
    }
}
```

## Configuration

Le *Service Container* est configuré dans le fichier `config/services.yaml`

```yaml
parameters:
    services:
        # default configuration for services in *this* file
        _defaults:
            autowire: true # Automatically injects dependencies in your services.
            autoconfigure: true # Automatically registers your services as commands, event subscribers, etc.
```

> Si le service n'est pas trouvé, une exception bien définie est lancée et vous avertit.

## Configurer un service manuellement

```
parameters:
    services:
        ...
        App\Service\MonService:
            class: MonService
            arguments: ['valeur passée au constructeur']
```

Cette configuration crée un service `mon_service` (id du service) qui est une instance de `MonService` . Sous la clef `arguments` on retrouve les arguments passés au constructeur.

## Configurer un service instancié par un pattern *factory*

Si le constructeur du service est **privé**, ou instancié via un *design pattern factory*, vous pouvez le configurer avec la clef `factory` en indiquant la classe et la méthode à appeler pour instancier l'objet.

```yaml
parameters:
    services:
        App\Service\MonService:
            factory: ['App\Service\MonService','create'] # on précise ici la classe et la méthode factory
            arguments: ['indéfini'] # les arguments à passer en paramètre, ici un paramètre undefined.
```

Beaucoup d'autres possibilités pour enregistrer les services, consulter la documentation.

## Conclusion

Avec le *Service Container*, **vous n'avez jamais à vous soucier de comment instancier et mettre à disposition un service pour un autre service**. Le framework le fait pour vous.

Vous pouvez donc vous concentrer sur votre code métier (et le flot d'execution métier uniquement), ignorer une grande partie du flot d'execution et passer moins de temps sur "le code système".

Cela permet de **passer d'une programmation impérative à une programmation déclarative** sur les aspects système (routage, dépendances entre services).

> La programmation déclarative (comme SQL!) est toujours plus puissante et appréciable car on masque beaucoup de détails d'implémentation et cela fournit des abstractions utiles à l'utilisateur.

## Références

- Service Container, documentation officielle sur le composant Service Container;
- Types of Injection, documentation officielle sur les différents types d'injection ;
- Defining Services Dependencies Automatically (Auto-wiring), documentation officielle sur l'autowiring ou chargement automatique de classes
- Binding Arguments by Name or Type, utiliser le mot-clef `bind` dans `config/services.yaml` pour associer des valeurs/implémentations à des noms d'arguments ou à des types.

## Exercices

Réaliser les exercices fournis avec le module 7.

## TP Fil Rouge - Système de billetterie

Poursuivre le TP file rouge (Partie 2), correspondant au module couvert.