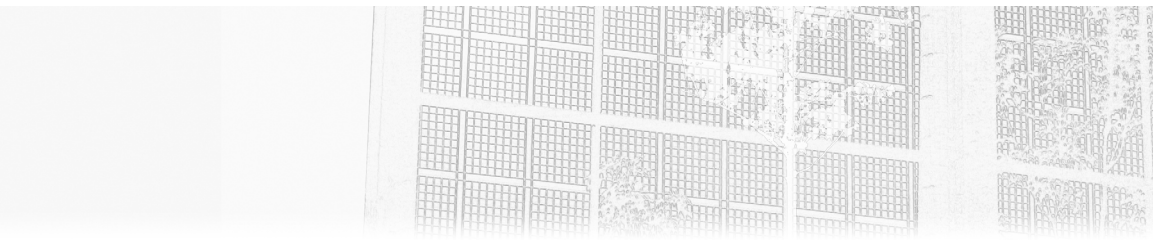


Chapitre 4
Services de Phalcon

1. Injection de dépendances (DI)

Le DI est un élément très important de la structure de ce framework. L'objectif principal de l'injecteur de dépendances : avoir un accès à différents services définis. Le principe consiste à définir des services (interaction avec une base de données, système de traduction, système de cache, etc.) qui peuvent être utilisés, peu importe où l'on se trouve dans l'application.

Si l'injecteur de dépendances n'existait pas, il faudrait rouvrir et refermer une connexion à une base de données chaque fois que l'on souhaite interagir avec elle, ce qui se traduirait par de nombreux copier-coller et une maintenance du code plus complexe.

Le système d'injection de dépendances propose deux modes :

– simple : appel complet du code ;
– partagé : fonctionne comme un Singleton.

1.1 Une dépendance simple

1.1.1 Explication

Une dépendance simple permet d'initialiser et de préparer un objet. Cette dépendance renverra toujours un nouvel objet. L'objectif principal, c'est d'avoir un objet avec une durée de vie limitée.

Présentation d'un cas concret

Dans le cadre d'un projet, vous devez créer différents PDF avec la même mise en forme (logo, en-tête, pied de page, police...). Afin d'éviter la duplication de code dans toute l'application, il est recommandé de créer une dépendance qui initialise un objet avec ces paramètres de style.

Si la dépendance est partagée

- L'utilisateur demande un export PDF de ses informations personnelles.
- Le développeur fait appel au service de création de PDF ; un objet préparamétré est envoyé.
- Le développeur ajoute les données personnelles dans le PDF et renvoie le PDF complet à l'utilisateur.
- L'utilisateur demande l'export PDF de ces dossiers.
- Le développeur fait de nouveau appel au service de création de PDF et obtient le même objet que précédemment.
- Le développeur ajoute les données relatives aux dossiers dans le PDF et renvoie le PDF complet à l'utilisateur.
- L'utilisateur obtient un PDF contenant ses données personnelles, ainsi que celles relatives à ses dossiers.

Si la dépendance n'est pas partagée

- L'utilisateur demande un export PDF de ses informations personnelles.
- Le développeur fait appel au service de création de PDF ; un objet préparamétré est envoyé.

– Le développeur ajoute les données personnelles dans le PDF et renvoie le PDF complet à l'utilisateur.
– L'utilisateur demande l'export PDF de ses dossiers.
– Le développeur fait de nouveau appel au service de création de PDF et obtient un objet tout neuf.
– Le développeur ajoute les données relatives aux dossiers dans le PDF et renvoie le PDF complet à l'utilisateur.
– L'utilisateur obtient un PDF contenant uniquement les informations relatives à son dossier.

1.1.2 Mise en œuvre d'un service avec une simple instanciation

Les dépendances sont définies dans le fichier app/config/services.php.

Syntaxe

```
$di->set('<nom du service>', new <classe>());
```

Exemple d'un service de PDF

```
$di->set('fpdf', new Fpdf());
```

Cet exemple permet de définir un service de création de PDF. Pour appeler le service depuis un contrôleur, par exemple, il suffit de procéder de la manière suivante.

Syntaxe de récupération d'un service

```
$this->di->get('<nom du service>');
```

Exemple avec le service de PDF

```
$oPDF = $this->di->get('fpdf');
```

Un nouvel objet *Fpdf* est renvoyé dans la variable `$oPDF`.

1.1.3 Mise en œuvre d'un service avec une fonction anonyme

Fonction simple

Syntaxe

```
$di->set('<nom du service>',function ()
    {
        // code
        return <objet de service>
    }
);
```

Exemple de création d'un service de PDF avec une fonction anonyme

```
$di->set('fpdf', function () {
        $oPdf = new Fpdf();

        $oPdf->AddPage();
        $oPdf->Image('logo_entete.png',8,5,25);

        return $oPdf;
    }
);
```

Cet exemple permet d'initialiser un objet Fpdf avec un logo en en-tête à chaque fois que le service est appelé.

Passage de variable

Pour certains types de services, il est utile de passer des paramètres de données.

Syntaxe

```
$di->set('<nom du service>',function () use (<liste de variables>)
    {
        // code
        return <objet de service>
    }
);
```

Exemple de connexion à une base de données avec l'objet de configuration

```
$config = new \Phalcon\Config
(
   [
       'adapter'     => 'Postgresql',
       'host'        => '192.168.99.100',
       'username'    => 'postgres',
       ...
   ]
);

$di->set('db', function () use ($config) {
   $class = 'Phalcon\Db\Adapter\Pdo\\' . $config->database->adapter;
   $params = [
       'host'     => $config->database->host,
       'username' => $config->database->username
       ...
   ];

   $connection = new $class($params);

   return $connection;
});
```

Passage de données avec le DI

Il est aussi possible d'utiliser les données à l'intérieur du DI pour créer un service.

Typiquement, pour une connexion à une base de données, il est plus pratique d'utiliser l'objet de configuration. Cet objet contient traditionnellement les informations de connexion des différents services de l'application web.

Syntaxe

```
$di->set('<nom du service>', function () {
       $oService = $this->get('<nom du service>');

       // Ou

       $oService = $this->getNomService();

       ...
   }
);
```

Exemple complet de connexion à une base de données

app/config/config.php

```
return new \Phalcon\Config([
    'database' => [
        'adapter' => 'Postgresql',
        'host'    => '192.168.99.100',
        ...
    ]
]);
```

app/config/services.php

```
$di->set('config', function () {
    return include APP_PATH . "/config/config.php";
});

$di->set('db', function () {
    $config = $this->getConfig();
    // ou $config = $this->get('config');

    $class = 'Phalcon\Db\Adapter\Pdo\\' . $config->database->adapter;
    $params = [
        'host'     => $config->database->host,
        'username' => $config->database->username,
        ...
    ];

    $connection = new $class($params);

    return $connection;
});
```

Dans cet exemple, un service **config** renvoie les informations présentes dans le fichier **app/config/config.php**. Ce service est ensuite utilisé pour créer la connexion à la base de données.

Chapitre 6
Gérer les formulaires et les liens

1. Vue d'ensemble

1.1 Introduction

Dans les sites web dynamiques, il est très souvent nécessaire d'interagir avec l'utilisateur.

En HTML, il existe principalement deux méthodes pour interagir avec un utilisateur :

– les liens (balise `<a>`) ;

– les formulaires (balise `<form>`).

Des scripts PHP peuvent être utilisés pour traiter le clic de l'utilisateur sur un lien ou la saisie de l'utilisateur dans un formulaire.

1.2 Les liens

Le lien est la technique de base qui permet à un utilisateur de naviguer entre les différentes pages d'un site.

Un lien HTML est défini entre les balises `<a>` et `</a>`.

Syntaxe simplifiée

```
<a
  [ href="url" ]
  [ id="identifiant_lien" ]
  [ target="cible" ]
```

```
>
...
</a>
```

Les attributs de la balise <a> sont les suivantes :

`href` URL (*Uniform Resource Locator*) relative ou absolue qui est appelée par le lien.

`id` Identifiant du lien. Si la page HTML contient plusieurs liens, l'identifiant permet de les différencier. En ce qui nous concerne, cet identifiant ne présente pas d'intérêt car il n'est pas récupéré dans le script de traitement du lien. Par contre, il peut être utilisé côté client, en JavaScript par exemple.

`target` Cible (par exemple une autre fenêtre) dans laquelle ouvrir l'URL cible. Par défaut, l'URL cible s'affiche dans la même fenêtre.

L'URL peut contenir des paramètres qui permettent de passer des informations d'une page à une autre.

Syntaxe

```
url_classique?nom=valeur[&...]
```

Le point d'interrogation (?) introduit la liste des paramètres de l'URL séparés par le caractère esperluette (&) ; chaque paramètre est constitué par un couple nom/valeur sous la forme `nom=valeur` :

```
www.monsite.com/info/accueil.php?prenom=Olivier
chercher.php?prenom=Olivier&nom=HEURTEL
```

Exemple

– Script `page1.php`

```
<?php
// Initialisation d'une variable.
$nom='Olivier';
?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="fr">
  <head><meta charset="utf-8" /><title>Page 1</title></head>
  <body>
    <div>
    <!-- lien vers la page 2 en passant la valeur de $nom
         dans l'URL -->
    <a href="page2.php?nom=<?= $nom ?>">Page 2</a>
    </div>
  </body>
</html>
```

– Source de la page dans le navigateur

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="fr">
  <head><meta charset="utf-8" /><title>Page 1</title></head>
  <body>
    <div>
    <!-- lien vers la page 2 en passant la valeur de $nom
         dans l'URL -->
    <a href="page2.php?nom=Olivier">Page 2</a>
    </div>
  </body>
</html>
```

Résultat

– Affichage de la page 1 :

– Résultat du clic sur le lien :

Pour l'instant, aucun nom n'est affiché dans la deuxième page. La variable `$nom` définie dans le script `page1.php` n'est pas disponible dans le script `page2.php` (voir le chapitre Introduction à PHP, section Les bases du langage PHP - Variables - Portée et durée de vie). De plus, notre script ne contient aucune instruction permettant de récupérer les données passées dans l'URL ; nous verrons comment procéder à la section Récupérer les données d'une URL ou d'un formulaire.

1.3 Les formulaires

1.3.1 Petit rappel sur les formulaires

Le formulaire est un outil de base indispensable pour les sites web dynamiques puisqu'il permet à l'utilisateur de saisir des informations et donc d'interagir avec le site.

Un formulaire HTML est défini entre les balises `<form>` et `</form>`.

Syntaxe simplifiée

```
<form
  [ action="url_de_traitement" ]
  [ method="GET"|"POST" ]
  [ id="identifiant_formulaire" ]
  [ target="cible" ]>

...
</form>
```

Les attributs de la balise `<form>` sont les suivants :

`action` URL (*Uniform Resource Locator*) relative ou absolue qui va traiter le formulaire (en ce qui nous concerne, un script PHP). Cet attribut est obligatoire pour se conformer à la recommandation XHTML stricte.

`method` Mode de transmission vers le serveur des informations saisies dans le formulaire.

`GET` (valeur par défaut) : les données du formulaire sont transmises dans l'URL.

`POST` : les données du formulaire sont transmises dans le corps de la requête.

`id` Identifiant du formulaire. Si la page HTML contient plusieurs formulaires, l'identifiant permet de les différencier. En ce qui nous concerne, cet identifiant ne présente pas d'intérêt car il n'est pas récupéré dans le script de traitement du formulaire. Par contre, il peut être utilisé côté client, en JavaScript par exemple.

`target` Cible (par exemple une autre fenêtre) dans laquelle ouvrir l'URL cible. Par défaut, l'URL cible s'affiche dans la même fenêtre.

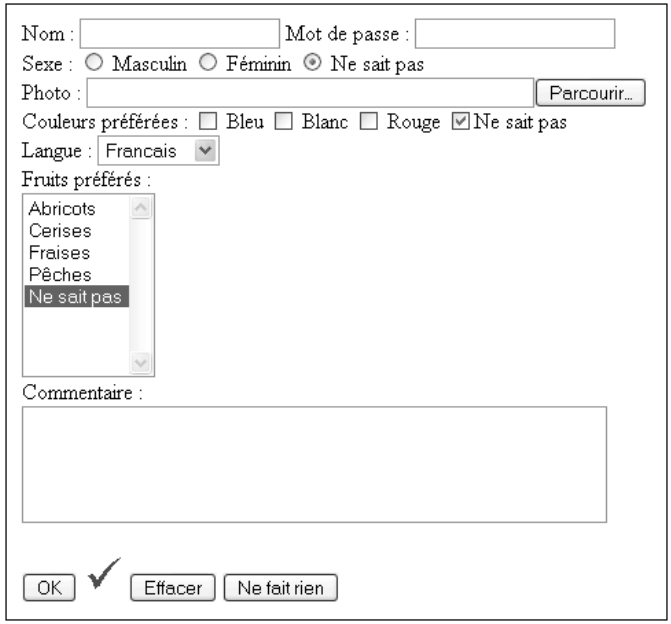
Entre les balises `<form>` et `</form>`, il est possible de placer des balises `<input>`, `<select>` ou `<textarea>` pour définir des zones de saisie.

Exemple (formulaire HTML complet)

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="fr">
  <head>
    <meta charset="utf-8" />
    <title>Saisie</title>
  </head>
  <body>
    <form action="saisie.php" method="post">
    <div>
      Nom :
      <input type="text" name="nom" value=""
           size="20" maxlength="20" />
      Mot de passe :
      <input type="password" name="mot_de_passe" value=""
           size="20" maxlength="20" />
      <br />Sexe :
      <input type="radio" name="sexe" value="M" />Masculin
      <input type="radio" name="sexe" value="F" />Féminin
      <input type="radio" name="sexe" value="?"
                          checked="checked" />Ne sait pas
      <br />Photo :
      <input type="file" name="photo" size="50" />
      <br />Couleurs préférées :
      <input type="checkbox" name="couleurs[bleu]" />Bleu
      <input type="checkbox" name="couleurs[blanc]" />Blanc
      <input type="checkbox" name="couleurs[rouge]" />Rouge
      <input type="checkbox" name="couleurs[pas]"
                               checked="checked" />Ne sait pas
      <br />Langue :
      <select name="langue">
        <option value="E">Espagnol</option>
        <option value="F" selected="selected" >Francais</option>
        <option value="I">Italien</option>
      </select>
      <br />Fruits préférés :<br />
      <select name="fruits[]" multiple="multiple" size="8">
        <option value="A">Abricots</option>
        <option value="C">Cerises</option>
        <option value="F">Fraises</option>
        <option value="P">Pêches</option>
        <option value="?" selected="selected">
                         Ne sait pas</option>
      </select>
      <br />Commentaire :<br />
      <textarea name="commentaire" rows="4" cols="50"></textarea>
      <br />
      <input type="hidden" name="invisible" value="123" /><br />
```

```
      <input type="submit" name="soumettre" value="OK" />
      <input type="image" name="valider" alt="valider"
src="valider.gif" />
      <input type="reset" name="effacer" value="Effacer" />
      <input type="button" name="action" value="Ne fait rien" />
    </div>
    </form>
  </body>
</html>
```

Résultat



PHP peut intervenir à deux endroits par rapport au formulaire :

– Pour la construction du formulaire, si ce dernier doit contenir des informations dynamiques.
– Pour le traitement du formulaire (c'est-à-dire des données saisies par l'utilisateur dans le formulaire).