Editions ENI

PHP 7

Développez un site web dynamique et interactif

(2e édition)

Collection
Ressources Informatiques

Extrait

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 PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><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 PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><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>
```

– Script `page2.php`

```
<?php
echo "Page 2 - Bonjour $nom";
?>
```

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.

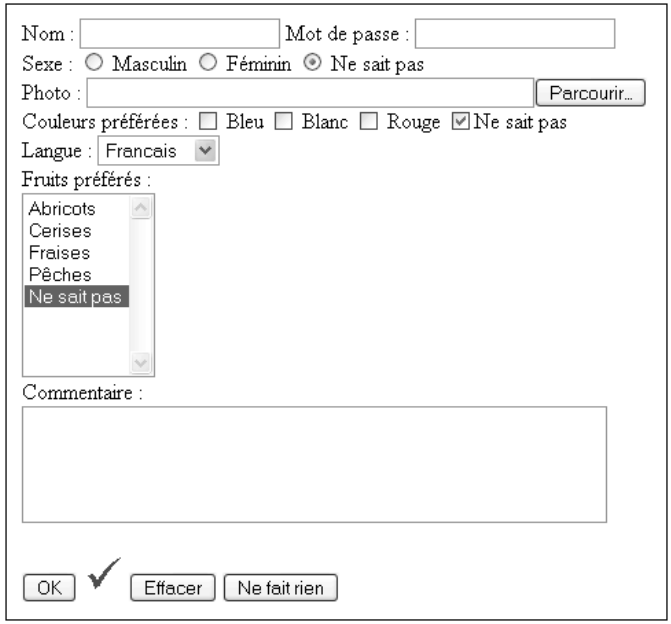
Entre les balises `<form>` et `</form>`, il est possible de placer des balises `<input>`, `<select>` ou `<textarea>` pour définir des zones de saisie.

Exemple (formulaire HTML complet)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>Saisie</title>
  </head>
  <body>
    <form action="" 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" value="" 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" 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).

Editions ENI

Laravel

Un framework efficace pour développer vos applications PHP

Collection
Expert IT

Extrait

Chapitre 6
Organiser grâce aux contrôleurs

1. Définition et usage

1.1 Qu'est-ce qu'un contrôleur ?

Dans le modèle MVC (modèle-vue-contrôleur), le contrôleur contient la logique concernant les actions effectuées par l'utilisateur. En pratique, dans une application Laravel, l'utilisation de contrôleurs permet de libérer les routes du code qu'elles contiennent dans leurs fonctions de rappel.

Un contrôleur est matérialisé par une classe et chacune de ses méthodes représente une action. Une action correspond généralement à une route.

1.2 Déplacer le code des fonctions de rappel des routes

Les routes créées dans les chapitres précédents sont surchargées de code pour traiter les actions dans leurs fonctions de rappel. L'objectif est de les libérer de ce code pour obtenir un fichier de routes propre qui fait simplement le lien entre les URI de l'application et les actions à effectuer pour chacune de ces URI.

Le fichier des routes `routes/web.php` est ainsi allégé, et devient un index des différentes URI avec leurs actions associées.

Exemple de fichier de routes sans fonction de rappel

```
Route::get('videos', 'VideoController@index');

Route::get('videos/creer', 'VideoController@create');

Route::post('videos/creer', 'VideoController@store');

Route::get('videos/{id}', 'VideoController@show');
```

Le fonctionnement des routes avec les contrôleurs est détaillé dans la suite de ce chapitre.

2. Créer et utiliser un contrôleur

2.1 Créer un contrôleur avec Artisan

Comme pour de nombreux autres éléments, Artisan, l'outil en ligne de commande fourni avec Laravel, permet de créer rapidement un fichier contenant la structure de base d'un contrôleur :

Créer un contrôleur

```
php artisan make:controller ProductController
```

Les contrôleurs sont créés dans le dossier `app/Http/Controllers`.

Remarque

Par convention, les contrôleurs sont en notation Camel Case et terminent tous avec le mot ***`Controller`****. Cela permet de les différencier des services et des modèles auxquels ils font référence.*

2.2 Structure d'un contrôleur

Un contrôleur est une classe qui étend la classe de base **`Controller`** et dont chaque méthode publique représente généralement une action qui correspond à une route.

Fichier `app/Http/Controllers/ProductController.php`

```
<?php

namespace App\Http\Controllers;

class ProductController extends Controller
{

    public function home()
    {
        return 'Bienvenue sur la page des produits';
    }

    public function show($id)
    {
        return 'Page du produit : ' . $id;
    }
}
```

Une méthode d'un contrôleur retourne une réponse, exactement comme dans les fonctions de rappels du chapitre « Créer des routes » à propos des routes. Dans l'exemple précédent, les méthodes **`home`** et **`show`** retournent une réponse sous la forme d'une chaîne de caractères, mais il aurait aussi bien pu s'agir d'une vue ou d'un objet JSON. Le code ci-dessous présente le même contrôleur, dont les méthodes retournent cette fois des vues.

Fichier `app/Http/Controllers/ProductController.php`

```
<?php

namespace App\Http\Controllers;

use App\Product;

class ProductController extends Controller
{
```

```
    public function home()
    {
        return view('products.home');
    }

    public function show($id)
    {
        return view('products.show', [
            'product' => Product::find($id)
        ]);
    }
}
```

2.3 Lier une route à un contrôleur

Dans le fichier `routes/web.php`, à la place d'utiliser des fonctions de rappel comme cela a été le cas dans les précédents chapitres, l'organisation avec des contrôleurs permet de simplement signaler au routeur quel contrôleur aller chercher et quelle action effectuer. La syntaxe à utiliser est **`NomDuControl-ler@action`** sous forme d'une chaîne de caractères.

Lier une route à un contrôleur

```
Route::get('products', 'ProductController@home');

Route::get('products/{id}', 'ProductController@show');
```

L'action correspond au nom de la méthode du contrôleur à exécuter.

2.4 Variables

2.4.1 Passer un paramètre à un contrôleur

Pour passer une variable à un contrôleur, il faut la déclarer dans le chemin de la route entre accolades et mettre la variable en paramètre de la méthode qui représente l'action à effectuer, comme dans l'exemple ci-dessous.

Fichier `routes/web.php`

```
Route::get('products/{id}', 'ProductController@show');
```

Fichier app/Http/Controllers/ProductController.php

```
class ProductController extends Controller
{

    public function show($id)
    {
        return 'Page du produit : ' . $id;
    }
}
```

2.4.2 Valeurs optionnelles

Les paramètres optionnels (qui permettent par exemple d'associer une même action aux URI `products` et `products/17`) doivent être déclarés avec un point d'interrogation dans le chemin de la route et avec une valeur par défaut à **`null`** dans l'action du contrôleur.

Fichier routes/web.php

```
Route::get('products/{id?}', 'ProductController@show');
```

Fichier app/Http/Controllers/ProductController.php

```
class ProductController extends Controller
{

    public function show($id = null)
    {

        if ($id === null) {
            return Product::all();
        } else {
            return Product::findOrFail($id);
        }

    }
}
```

3. Dépendances

3.1 Injection de dépendance

3.1.1 Fonctionnement

Il est possible grâce aux fonctions assistantes et au système de façades fourni par Laravel d'accéder à de nombreux éléments dans les contrôleurs comme la requête de l'utilisateur, le cache, les sessions, etc.

Il est également possible d'injecter les dépendances directement dans les méthodes des contrôleurs, simplement en donnant l'indication d'un type connu par Laravel à un paramètre de la méthode représentant l'action.

Par exemple, pour récupérer une instance de la classe **`Illuminate\Http\Request`**, on peut passer à une méthode un premier paramètre **`$request`** préfixé par son type.

Affichage de l'URL dans une méthode de contrôleur

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HomeController extends Controller
{

    public function home(Request $request)
    {
        return 'L'URL de la page est :' . $request->url();
    }
}
```

Remarque

*La classe **`Request`** et la fonction assistante pour y accéder seront étudiées dans la suite de ce chapitre et au chapitre « Traiter des formulaires », les sessions et le cache au chapitre « Les sessions et le cache ». En attendant, on retiendra simplement qu'on peut transmettre d'autres éléments du framework à une méthode du contrôleur.*