Darth Killer

Zeste de savoir

Les filtres en PHP

12 août 2019

Table des matières

Traiter les informations reçues par le client via formulaire, ou plus généralement via COOKIE, GET ou POST. Un vrai calvaire miniature bien souvent tant la succession de conditions rend le code difficile à lire et donc à maintenir, surtout quand les tests de validité deviennent compliqués. On peut le voir sur l'exemple ci-dessous du traitement d'un formulaire d'inscription à un site.

```php
<?php
if($_SERVER['REQUEST_METHOD'] != 'POST') {
   // On ne peut pas nous avoir envoyé notre formulaire
}
else {
    if(empty($_POST['login'])) {
        // Identifiant d'authentification non-renseigné
    }

    if(empty($_POST['passwd'])) {
       // Mot de passe associé non-renseigné
    }
    else if(empty($_POST['passwd2'])) {
       // Répétition du mot de passe non-renseignée
    }
    else if($_POST['passwd'] != $_POST['passwd2']) {
       // Les mots de passe ne correspondent pas
    }

    if(empty($_POST['email'])) {
       // E-mail non-renseigné
    }
    else
        if(preg_match('¤^[a-z] (?: (?:\w|[.-])* [a-z0-9])? @ [a-z] (?: (?:\w|[
        $_POST['email'])) {
       // Syntaxe e-mail invalide
    }
    else if(empty($_POST['email2'])) {
       // Répétition de l'e-mail non-renseignée
    }
    else if($_POST['email'] != $_POST['email2']) {
       // Les adresses e-mail ne correspondent pas
```

```
31      }
32
33      if(empty($_POST['dateNaissance'])) {
34          // Mettre la valeur NULL histoire que ne pas avoir de
                Notice
35          $_POST['dateNaissance'] = null;
36      }
37      else {
38          $_POST['dateNaissance'] =
                DateTime::createFromFormat('d/m/Y',
                $_POST['dateNaissance']);
39          if($_POST['dateNaissance'] === false) {
40              // Date invalide
41          }
42      }
43  }
```

Les filtres PHP représentent un outil très efficace pour résoudre ce problème récurrent, en définissant les règles de validation avant de les appliquer. Tout s'articule autour de la fonction `filter_var()` et de ses variantes : `filter_var_array()`, qui filtre un tableau d'une traite, `filter_input()` qui filtre un élément d'une super-globale comme `$_GET` ou `$_POST`, et `filter_input_array()` qui filtre la super-globale en entier d'une traite.

1. Filtrer une variable avec filter_var()

Il existe 2 types de filtres de base : les filtres de validation, qui renvoient la valeur qu'on leur a donné ou false, et les filtres de nettoyage (”sanitize”) qui renvoient la valeur qu'on leur a donné privée de certains éléments. `FILTER_VALIDATE_URL` renverra false si une URL contient des caractères incompatibles, alors que `FILTER_SANITIZE_URL` retirera les caractères interdits et renverra l'URL ainsi nettoyée.

Chaque filtre dispose d'une liste d'options et de drapeaux (”flags”) qu'on peut ajouter pour altérer son comportement. Sans rentrer dans les détails techniques, un drapeau est une constante contenant une valeur entière, et on peut combiner les drapeaux entre eux avec l'opérateur |, ce qui nous permet d'activer des options qui n'attendent pas de valeur particulières. Par exemple, si on veut indiquer à `FILTER_VALIDATE_URL` qu'on veut dans notre URL un chemin et qu'on veut aussi des variables (partie ”query string”), on combinera les 2 flags correspondant : `FILTER_FLAG_PATH_REQUIRED | FILTER_FLAG_QUERY_REQUIRED`. Il existe un drapeau nommé différemment, qui est compatible avec tous les filtres (bien qu'il ne semble avoir effet que sur ceux de validation) : `FILTER_NULL_ON_FAILURE` forcera un filtre de validation à rendre NULL au lieu de false en cas d'échec.

```
1  <?php
2  $options = array(
3      'options' => array(
```

```
        'min_range' => 0 // Valeur minimum acceptable
    ),
    'flags' => FILTER_FLAG_ALLOW_OCTAL, // autorise les valeurs en
        octal
);
filter_var('0755', FILTER_VALIDATE_INT, $options); // Retourne 0775
filter_var('machin', FILTER_VALIDATE_INT, $options); // Retourne
    false : échec

// On veut changer la valeur par défaut à retourner en cas d'échec
$options['options']['default'] = 3;
filter_var('machin', FILTER_VALIDATE_INT, $options); // Retourne 3
    : nouvelle valeur pour l'échec

filter_var('machin@truc', FILTER_VALIDATE_EMAIL); // Retourne
    'machin@truc' : e-mail OK
filter_var('???', FILTER_VALIDATE_EMAIL); // Retourne false : échec
```

2. Traiter un formulaire complet avec filter_input() et filter_input_array()

La petite différence entre `filter_var()` et `filter_input()` est qu'avec cette dernière on indique sous forme de constance INPUT_* le nom de la super-globale à filtrer ainsi que l'index (chaîne en général) à regarder, de ce fait il se peut qu'on tombe sur un index qui n'existe pas, mais on n'aura pas de notice "undefined index" pour autant. Le filtre renverra tout simplement NULL s'il ne trouve pas l'élément.

```
<?php
// La ligne ci-dessous illustre le contenu de $_POST pour l'exemple
// $_POST = ['pseudo' => 'Marc', 'passwd' => 'kjhksdnksn'];
filter_input(INPUT_POST, 'pseudo', FILTER_SANITIZE_STRING); //
    Renvoie 'Marc'
filter_input(INPUT_POST, 'email', FILTER_VALIDATE_EMAIL); //
    Non-trouvé, renvoie NULL sans provoquer de notice.
```

Ce faisant, on peut utiliser ceci à notre avantage pour traiter tout notre formulaire d'une traite à l'aide de la variante `filter_input_array()`. Le tout est de créer un (gros) array définissant les filtres à utiliser. Ses index seront ceux à aller chercher, et les valeurs définiront le filtre et ses options et flags à appliquer. On récupérera en sortie un nouvel array, contenant toutes les réponses de filtres. Ceci combiné à une utilisation de `FILTER_NULL_ON_FAILURE` nous permet de savoir si le formulaire a été correctement rempli juste en regardant si l'array de réponse contient NULL : `in_array()`.

2. *Traiter un formulaire complet avec filter_input() et filter_input_array()*

```php
<?php
$filter_def = [
    'login' => FILTER_SANITIZE_STRING, // Pas d'option ni de flag, on peut préciser directement le filtre
    'passwd' => FILTER_UNSAFE_RAW, // Filtre "on ne change rien". Sert juste à avoir l'entrée "passwd" dans le résultat.
    'passwd2' => FILTER_UNSAFE_RAW,
    'email' => [
        'filter' => FILTER_VALIDATE_EMAIL,
        'flags' => FILTER_NULL_ON_FAILURE
    ],
    'email2' => FILTER_UNSAFE_RAW, // Pas besoin de valider l'email2, on va juste le comparer à email.
    'dateDeNaissance' => [ // On va voir une autre façon de valider la date de naissance plus tard
        'filter' => FILTER_VALIDATE_REGEXP, // Expression régulières
        'options' => [
            'regexp' => '#^\d{2}/\d{2}/\d{4}$#' // format de date : dd/MM/YYYY
            // Pas d'option default donc pas d'autre option.
        ],
        'flags' => FILTER_NULL_ON_FAILURE
    ]
];

$resultat = filter_input_array(INPUT_POST, $filter_def);
/*
 * contenu de $resultat :
 *
 * [
 *      'login' => 'Marc',
 *      'passwd' => 'lkqsjflqj',
 *      'passwd2' => 'lkqsjflqj',
 *      'email' => NULL
 *      'email2' => 'moi_toi.com'
 *      'dateDeNaissance' => NULL
 * ]
 */

if(in_array(null, $resultat, true)
        or $resultat['email'] != $resultat['email2']
        or $resultat['passwd'] != $resultat['passwd2']) {
    // Formulaire invalide
} else {
    // Formulaire OK
}
```

3. Filtres callback et classe de traitement

Pour aller plus loin, on peut se rendre compte de 2 soucis sur le traitement précédent : d'une part on n'obtient qu'un message unique en cas de formulaire invalide, dans la mesure où on ne regarde pas quel champs du formulaire a planté, et en plus on est obligé de faire à la main certaines vérifications, comme les 2 mots de passe qui doivent être identiques, ou les 2 e-mails. Ou le format de la date de naissance qui n'est pas vérifié du tout.

Pour pallier au second problème, on est amené à créer nos propres filtres. Il s'agira de filtres "callback", où on créera une fonction de notre cru qui servira de filtre. Dans le cas présent, on peut s'en servir pour faire le test de la date de naissance, et pour incorporer les tests relatifs à l'email2 et au passwd2 à l'intérieur de ceux de l'e-mail et du passwd. Rien ne sert d'avoir 2 fois une même information à la sortie.

```php
<?php
$filter_def = [
    'login' => FILTER_SANITIZE_STRING, // Pas d'option ni de flag, on peut préciser directement le filtre
    'passwd' => [
        'filter' => FILTER_CALLBACK,
        'options' => function($input) {
            $reponse_filtre = NULL; // Valeur de départ, pour reproduire FILTER_NULL_ON_FAILURE
            $passwd = $input; // ligne peu utile mais qui illustre l'équivalent de FILTER_UNSAFE_RAW
            $passwd2 = filter_input(INPUT_POST, 'passwd2', FILTER_UNSAFE_RAW); // On récupère le 2e pass
            if($passwd != null and $passwd == $passwd2) {
                $reponse_filtre = $passwd;
            }
            return $reponse_filtre;
        }
    ],
    'email' => [
        'filter' => FILTER_CALLBACK,
        'options' => function($input) {
            $reponse_filtre = NULL; // Valeur de départ, pour reproduire FILTER_NULL_ON_FAILURE
            $email = filter_var($input, FILTER_VALIDATE_EMAIL); // Validation de l'email principal
            if($email !== false) { // Donc l'email est valide
                $email2 = filter_input(INPUT_POST, 'email2', FILTER_UNSAFE_RAW); // On récupère l'e-mail 2
                if(!is_null($email2) and ($email == $email2)) {
                    $reponse_filtre = $email;
                }
            }
            return $reponse_filtre;
        }
```

```php
    ],
    'dateDeNaissance' => [
        'filter' => FILTER_CALLBACK,
        'options' => function($input) {
            $reponse_filtre = null;
            // Première étape : récupérer le jour, le mois et l'année
            // Plusieurs possibilités. Je vais rester sur la regexp.
            if(preg_match('#^\d{2}/\d{2}/\d{4}$#', $input, $matches)) {
                // Maintenant, on vérifie qu'il s'agit d'une date valide
                if(checkdate($matches[2], $matches[1], $matches[3])) {
                    // Et maintenant il faut vérifier qu'elle est dans le passé
                    $ddn = DateTime::createFromFormat('d/m/Y', $input);
                    if($ddn < (new DateTime())) {
                        $reponse_filtre = $ddn; // On peut renvoyer directement l'objet DateTime pour manipulations
                    }
                }
            }
            return $reponse_filtre;
        }
    ]
];

$resultat = filter_input_array(INPUT_POST, $filter_def);
/*
 * contenu de $resultat :
 *
 * [
 *      'login' => 'Marc',
 *      'passwd' => 'lkqsjflqj',
 *      'email' => NULL
 *      'dateDeNaissance' => NULL
 * ]
 */

if(in_array(null, $resultat, true)) {
    // Formulaire invalide
} else {
    // Formulaire OK
}
```

Ce faisant, nous avons toujours le problème du message unique. Qui plus est, la définition

du formulaire constitue toujours un gros pavé au milieu de notre code PHP. Les 2 problèmes peuvent se résoudre en mettant en place une classe qui extériorise tout celà, et qui propose ses propres méthodes de validation, afin de remplir un array de messages d'erreur en cas de soucis.

```php
<?php

/**
 * FilterClass_FormulaireInscription
 * Traitement du formulaire d'inscription d'un site web
 *
 * @author Darth Killer
 */
class FilterClass_FormulaireInscription {

    // Message d'erreur pré-rédigés
    private static $ERREURS_ELEMVIDE = [
        'login' => "Vous devez renseigner un pseudonyme.",
        'passwd' => "Vous devez renseigner un mot de passe.",
        'passwd2' => "Vous devez répéter votre mot de passe.",
        'email' => "Vous devez renseigner un e-mail valide.",
        'email2' => "Vous devez répéter votre e-mail."
    ];
    private static $ERREURS_ELEMINVALID = [
        'passwd2' => "Les mot de passe ne coïncident pas.",
        'email' => "L'e-mail est invalide.",
        'email2' => "Les adresses e-mail ne coïncident pas.",
        'dateDeNaissance' =>
            "La date de naissance n'est pas renseignée sous un format valide d
    ];

    private $errors = array();
    private $definitions = array();

    public function __get($var) {
        // Implémentée pour accéder à $errors en publique en
            lecture seule
        // Il faudrait faire plus complexe pour le faire bien, mais
            hors sujet.
        if ($var != 'errors') {
            throw new BadMethodCallException(__CLASS__ .
                "::$var : inaccessible ou inexistant.");
        }
        return $this->errors;
    }

    public function __construct() {
        $this->definitions = [
            'login' => [
```

```
41                  'filter' => FILTER_CALLBACK,
42                  'options' => [$this, 'filter_login'] // la callback
                        est $this->filter_login()
43              ],
44              'passwd' => [
45                  'filter' => FILTER_CALLBACK,
46                  'options' => [$this, 'filter_passwd']
47              ],
48              'email' => [
49                  'filter' => FILTER_CALLBACK,
50                  'options' => [$this, 'filter_email']
51              ],
52              'dateDeNaissance' => [
53                  'filter' => FILTER_CALLBACK,
54                  'options' => [$this, 'filter_ddn']
55              ]
56          ];
57      }
58
59      public function filter() {
60          $reponse_filtre = filter_input_array(INPUT_POST,
                $this->definitions);
61          foreach(array_keys($reponse_filtre, NULL, true) as $key) {
                // On parcours les index dont la valeur est NULL
62              if(empty($this->errors[$key]) and
                    !empty(self::$ERREURS_ELEMVIDE[$key])) {
63                  // Donc l'erreur n'a pas déjà été gérée, c'est un
                        élément absent du formulaire de départ
64                  // Et pourtant considéré obligatoire.
65                  $this->errors[$key] =
                        self::$ERREURS_ELEMVIDE[$key];
66              }
67          }
68      }
69
70      private function filter_passwd($input) {
71          $reponse_filtre = NULL; // Valeur de départ, pour
                reproduire FILTER_NULL_ON_FAILURE
72          $passwd = $input; // ligne peu utile mais qui illustre
                l'équivalent de FILTER_UNSAFE_RAW
73          $passwd2 = filter_input(INPUT_POST, 'passwd2',
                FILTER_UNSAFE_RAW); // On récupère le 2e pass
74          if($passwd2 === null) {
75              $this->errors['passwd2'] =
                    self::$ERREURS_ELEMVIDE['passwd2'];
76          }
77          else if ($passwd != $passwd2) {
78              $this->errors['passwd2'] =
                    self::$ERREURS_ELEMINVALID['passwd2'];
79          }
```

```
80          else {
81              $reponse_filtre = empty($passwd) ? null : $passwd; //
                    Chaînes vides refusées
82          }
83          return $reponse_filtre;
84      }
85
86      private function filter_login($input) {
87          $reponse_filtre = filter_var($input,
                FILTER_SANITIZE_STRING);
88          return empty($reponse_filtre)?null:$reponse_filtre;
89      }
90
91      private function filter_email($input) {
92          $reponse_filtre = NULL; // Valeur de départ, pour
                reproduire FILTER_NULL_ON_FAILURE
93          $email = filter_var($input, FILTER_VALIDATE_EMAIL); //
                Validation de l'email principal
94          if($email === false) {
95              $this->errors['email'] =
                    self::$ERREURS_ELEMINVALID['email'];
96          }
97          else {
98              $email2 = filter_input(INPUT_POST, 'email2',
                    FILTER_UNSAFE_RAW); // On récupère l'e-mail 2
99              if($email2 === NULL) {
100                 $this->errors['email2'] =
                        self::$ERREURS_ELEMVIDE['email2'];
101             }
102             else if($email != $email2) {
103                 $this->errors['email2'] =
                        self::$ERREURS_ELEMINVALID['email2'];
104             }
105             else {
106                 $reponse_filtre = $email;
107             }
108         }
109         return $reponse_filtre;
110     }
111
112     private function filter_ddn($input) {
113         $reponse_filtre = null;
114         // Première étape : récupérer le jour, le mois et l'année
115         // Plusieurs possibilités. Je vais rester sur la regexp.
116         // Amélioration possible de la regexp : gérer différents
                séparateurs possibles
117         if (preg_match('#^(\d{2})/(\d{2})/(\d{4})$#', $input,
                $matches)) {
118             // Maintenant, on vérifie qu'il s'agit d'une date
                    valide
```

```php
            if (checkdate($matches[2], $matches[1], $matches[3])) {
                // Et maintenant il faut vérifier qu'elle est dans le passé
                $ddn = DateTime::createFromFormat('d/m/Y', $input);
                if ($ddn < (new DateTime())) {
                    $reponse_filtre = $ddn; // On peut renvoyer directement l'objet DateTime pour manipulations
                }
            }
        }
        if($reponse_filtre == NULL) {
            $this->errors['dateDeNaissance'] = self::$ERREURS_ELEMINVALID['dateDeNaissance'];
        }
        return $reponse_filtre;
    }

    public function hasErrors() {
        return !empty($this->errors);
    }

}

/** Utilisation (normalement la classe est dans un fichier différent) **/
// require_once('FilterClass_FormulaireInscription.class.php');
$filtre = new FilterClass_FormulaireInscription();
$resultat = $filtre->filter();
if($filtre->hasErrors()) {
    // Il y a des erreurs
    $listeErreurs = $filtre->errors; // array associatif
}
else {
    // Le formulaire est OK
}
```

On pourrait aller plus loin en mettant en place une classe mère et des méthodes pour construire en interne un array de définition. Mais ce sera pour une autre fois.

Au final, nous avons extériorisé notre définition des filtres, et donc n'avons dans notre code principal plus que quelques lignes qui font tout le traitement à notre place. La lisibilité est grandement accrue, ce qui augmentera d'autant la facilité et rapidité de notre maintenance.

Je remercie btw03 et Trasphere pour leur relecture privée, et Coyote pour la mise en place de ce superbe logo.