INFO33

Authentification en PHP : gestion de cookies et de sessions

Polycopié de Cours

O. Dubois

Table des matières

1 Introduction

Sur un site web interactif, il est parfois intéressant d'identifier un utilisateur d'une page à l'autre et de conserver des informations relatives à cet utilisateur.

Le terme "session" désigne la période de temps correspondant à la navigation continue d'un utilisateur sur un site. "Gérer les sessions" consiste donc à être en mesure d'identifier l'instant où un nouvel utilisateur accède à une page du site et de conserver des informations relatives à cet utilisateur jusqu'à ce qu'il quitte le site.

Beaucoup de sites interactifs proposent des fonctionnalités d'identification (membre, abonné …) car cela permet de conserver des informations sur l'utilisateur.

Plusieurs objectifs dans la gestion de sessions :

- identifier un utilisateur
- "suivre" cet utilisateur, et les données qui lui sont associées, d'une page à l'autre
- conserver des informations sur cet utilisateur d'une visite à l'autre

2 Authentification

2.1 Vue d'ensemble

Certains sites ont besoin d'authentifier les utilisateurs qui accèdent au site afin de vérifier que ces derniers sont bien inscrits.

Cette authentification comprend généralement deux étapes :

- saisie par l'utilisateur d'informations d'identification
  = typiquement un nom et un mot de passe
- vérification que l'identification saisie correspond bien à un utilisateur inscrit
  = typiquement, ce contrôle est réalisé à l'aide d'une base de données qui contient notamment la liste des utilisateurs.

2.2 Saisie de l'identification

L'identification peut être saisie de deux manières :

- par l'intermédiaire d'un formulaire prévu à cet effet
- par les fonctions d'authentification HTTP
  (en-tête HTTP `WWW-Authenticate: Basic realm="xxxxx"`)

3 Utilisation des cookies

3.1 Principe

Les cookies sont des fichiers texte courts stockés par le navigateur sur l'ordinateur de l'utilisateur, à la demande du serveur web.

Le serveur web envoie une valeur (le cookie) avec une page web et demande de la renvoyer lors des prochaines requêtes.

Grâce à cette valeur, le serveur peut retenir des informations concernant l'utilisateur.

Les utilisations les plus fréquentes des cookies sont :

- se souvenir du nom d'un utilisateur pour lui éviter de le ressaisir lors de sa prochaine authentification
- se souvenir des informations saisies dans un formulaire pour éviter de les redemander ou pour préremplir le formulaire la prochaine fois
- identifier chaque utilisateur de façon unique lors de ses visites à des fins statistiques

Les cookies sont également souvent utilisés pour stocker, coté client, un identifiant de session, les informations de celle-ci étant quant à elles enregistrées côté serveur.

PHP permet de récupérer très facilement, dans des variables, les données stockées dans le cookie.

La fonction **`setcookie`** permet de déposer un cookie sur le poste de l'utilisateur.

Syntaxe

```
bool setcookie(string nom [, string valeur [, int expiration [, string chemin
[, string domaine
[, bool sécurisé [, bool http_uniquement]]]]]])
```

- ***`nom`*** : Nom du cookie
- ***`valeur`*** : Valeur stockée dans le cookie
- ***`expiration`*** : Date d'expiration du cookie (timestamp Unix)
- ***`chemin`*** : Chemin du répertoire sur le serveur dans lequel le cookie est disponible. Mettre `/` pour rendre le cookie disponible sur le domaine entier ou `/rep/` pour rendre le cookie disponible dans le répertoire `/rep/` du domaine et tous ses sous-répertoires. Par défaut, égal au répertoire à partir duquel le cookie a été déposé
- ***`domaine`*** : Domaine auquel le cookie est renvoyé
- ***`sécurisé`*** : Mettre **`TRUE`** pour indiquer que le cookie ne doit être transmis que sur une connexion sécurisée (**`FALSE`** par défaut)
- ***`http_uniquement`*** : Mettre **`TRUE`** pour indiquer que le cookie ne doit être transmis que pour le protocole HTTP (**`FALSE`** par défaut)

Si la fonction n'est appelée qu'avec le paramètre ***`nom`***, le cookie portant ce nom est supprimé du poste de l'utilisateur. Si les paramètres ***`domaine`*** et ***`chemin`*** avaient été spécifiés lors du dépôt du cookie, il faut les spécifier à l'identique pour supprimer le cookie (mettre une date d'expiration dans le passé).

Si ***`valeur`*** est spécifié, un cookie portant le nom ***`nom`*** et contenant la valeur ***`valeur`*** est envoyé sur le poste de l'utilisateur ; s'il existe déjà un cookie portant ce nom, ce dernier est mis à jour avec la nouvelle valeur.

Le paramètre ***`expiration`*** permet de déterminer la date d'expiration du cookie (et donc la date de sa suppression du poste de l'utilisateur) ; si ce paramètre est non spécifié (ou égal à 0) le cookie expire à la fin de la session, c'est-à-dire lorsque l'utilisateur quitte le site.

Les cookies sont envoyés dans l'en-tête de la page. La fonction **`setcookie`** doit donc être appelée avant toute instruction (PHP ou HTML) qui a pour effet de commencer à construire la page HTML.

La fonction **`setcookie`** retourne **`TRUE`** si l'instruction a pu être exécutée (pas de données déjà transmises) et **`FALSE`** dans le cas contraire. Par contre, le code de retour de la fonction ne donne aucune information sur le fait que le cookie a réellement pu être déposé sur le poste de l'utilisateur : si ce dernier refuse les cookies, la fonction **`setcookie`** retourne quand même **`TRUE`** bien que le cookie n'ait pas été déposé.

Exemple

```
<?php
// Dépôt d'un cookie nommé "nom" contenant
// la valeur "Paul" et expirant à la fin de
// la session.
$ok = setcookie('nom','Paul');
// Idem mais expirant à date du jour (time() en secondes)
// plus 30 fois 24 fois 3600 secondes (soit 30 jours).
$ok = setcookie('nom','Paul',time()+(30*24*3600));
// Suppression du cookie nommé 'nom'.
$ok = setcookie('nom');
?>
```

Lorsque le cookie est renvoyé au serveur Web, par le navigateur, lors de la demande d'une page PHP, la valeur du cookie est accessible dans le tableau associatif **`$_COOKIE`** : la clé du tableau correspond au nom du cookie.

Un cookie est déposé sur le poste de l'internaute par la fonction **`setcookie`** puis renvoyé ultérieurement lors de la visite de n'importe quelle page du site (en fonction de la valeur des paramètres ***`chemin`*** et ***`domaine`***) ; le cookie n'est pas disponible immédiatement dans la page qui le dépose.

Pour savoir si un poste accepte les cookies, il faut déposer un cookie et recharger une page dans laquelle la présence ou non du cookie permettra de déterminer si le poste accepte les cookies.

Exemple de script **`tester_cookie.php`** qui permet de faire ce test :

```
<?php
// Tester si c'est le deuxième appel de la page.
if (! isset($_GET['retour'])) {
  // Non ...
  // Déposer le cookie.
  setcookie('test','test');
  // Et recharger la page avec une information dans
  // l'URL indiquant que c'est le deuxième passage.
  header('Location: tester_cookie.php?retour=1');
} else {
  // Oui ...
  // Tester si le cookie est "revenu".
  if (isset($_COOKIE['test'])) { // oui ...
    echo 'Cookie accepté';
  } else { // non ...
    echo 'Cookie refusé';
  }
}
?>
```

3.2 Application à la gestion des sessions

Les cookies peuvent être utilisés pour gérer les sessions, avec l'avantage d'être parfaitement indépendants à la fois de la navigation par balise **`<a href=...>`** et de la gestion des formulaires.

Ils présentent par contre de gros inconvénients : ils peuvent être refusés par les internautes, leur nombre par domaine est limité par les navigateurs et leur taille est elle aussi limitée.

Une gestion des sessions, dans laquelle un identifiant de session est transmis d'une page à l'autre (les autres informations étant stockées sur le serveur) peut être mise en place relativement facilement en utilisant les cookies, si l'utilisateur les accepte, et en utilisant l'URL dans le cas contraire. C'est exactement ce que propose la gestion des sessions de PHP.

4 Gestion de sessions en PHP

4.1 Principes

PHP propose un ensemble de fonctions qui facilitent la gestion des sessions. Les principes sont les suivants :

- un identifiant unique est automatiquement attribué à chaque session
- cet identifiant unique est transmis d'une page à l'autre, soit par cookie (si le poste accepte les cookies), soit par l'URL dans le cas contraire ; c'est PHP qui choisit automatiquement la bonne méthode et assure ce transfert
- les données à conserver d'une page à l'autre pendant la durée de la session sont indiquées à PHP qui se charge automatiquement de restituer leur valeur au début du script et de les sauvegarder à la fin du script

4.2 Mise en œuvre

Les principales fonctions du module de gestion des sessions sont les suivantes :

Nom	Rôle
`session_start`	Ouvre une nouvelle session ou réactive la session courante
`session_id`	Retourne (ou éventuellement modifie) l'identifiant de la session
`session_name`	Retourne (ou éventuellement modifie) le nom de la variable utilisée pour stocker l'identifiant de la session
`session_abort`	Annule les modifications apportées aux données de session et termine la session
`session_reset`	Réinitialise les données de session à leurs valeurs initiales
`session_destroy`	Supprime la session
`session_status`	Retourne le statut actuel d'une session

En complément, le tableau **`$_SESSION`** permet de manipuler très facilement les données de session.

4.2.1 session_start

Syntaxe

```
bool session_start([array tableauoptions])
```

- ***`tableauoptions`*** : tableau associatif qui permet de définir des options qui vont remplacer les directives de configuration relatives aux sessions.

La fonction **`session_start`** interroge l'environnement pour détecter si une session a déjà été ouverte pour l'utilisateur actuel. Si oui, les données enregistrées dans la session sont restituées. Autrement, une nouvelle session est ouverte avec attribution d'un identifiant.

`session_start` retourne **`TRUE`** si la session a pu être créée avec succès, et **`FALSE`** sinon.

Tout script concerné par la gestion des sessions doit appeler **`session_start`** pour pouvoir avoir accès aux données de session.

Si la session n'est pas encore ouverte, la fonction **`session_start`** va chercher à déposer un cookie contenant l'identifiant de session, sur le poste de l'utilisateur.

Comme la fonction **setcookie**, **session_start** doit donc être appelée avant toute instruction (PHP ou HTML) ayant pour effet de commencer à construire la page HTML.

Exemple

```
<?php
// Ouvrir/réactiver la session.
session_start();
?>
```

4.2.2 session_id

Syntaxe

string session_id([string *nouvelle_valeur***])**

- ***nouvelle_valeur*** : Nouvelle valeur attribuée à l'identifiant de session.

Appelée sans paramètre, la fonction **session_id** retourne la valeur de l'identifiant de session. Cette valeur sera toujours vide si **session_start** n'a pas été appelé dans le script.

Appelée avec un paramètre, la fonction **session_id** modifie la valeur attribuée à l'identifiant de session.

4.2.3 session_name

Syntaxe

string session_name([string *nouvelle_valeur***])**

- ***nouvelle_valeur*** : Nouvelle valeur attribuée au nom de la variable qui stocke l'identifiant de session.

Appelée sans paramètre, la fonction **session_name** retourne le nom de la variable dans laquelle l'identifiant de session est stocké. La fonction **session_name** retourne un résultat même si la fonction **session_start** n'a pas été appelée dans le script.

Appelée avec un paramètre, la fonction **session_name** modifie le nom de la variable.

4.2.4 Manipuler les données enregistrées dans la session

Après appel à la fonction **session_start** les données de session peuvent être manipulées directement dans le tableau associatif **$_SESSION**. Toutes les entrées stockées dans le tableau **$_SESSION** sont automatiquement enregistrées en tant que données de session.

Pour enregistrer une nouvelle donnée dans la session, il suffit de stocker cette donnée dans le tableau **$_SESSION** avec une clé donnée.

Pour lire ou modifier une donnée de session préalablement enregistrée, il suffit d'accéder au tableau **$_SESSION** en utilisant la clé correspondant à cette donnée.

Exemple

- Script **page1.php** qui ouvre une session et enregistre des données dans la session.

```
<?php
// Ouvrir/réactiver la session.
session_start();
// Enregistrer deux informations dans la session.
$_SESSION['prenom'] = 'Bart';
$_SESSION['informations'] =  // c'est un tableau ...
      array('prenom'=>'Bart','nom'=>'SIMPSON');
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Page 1</title>
    </head>
    <body>
    <div><a href="page2.php">page2.php</a></div>
    </body>
</html>
```

- Script **page2.php** qui affiche la valeur des variables de session.

```
<?php
// Appel à session_start.
session_start();
// Affichage.
echo '$_SESSION[\'prenom\'] = ',
     isset($_SESSION['prenom'])?$_SESSION['prenom']:'',
     '<br />';
echo '$_SESSION[\'informations\'][\'nom\'] = ',
     isset($_SESSION['informations']['nom'])?
                $_SESSION['informations']['nom']:'',
     '<br />';
?>
```

Pour supprimer une donnée de session, il est possible d'utiliser la fonction **unset** pour supprimer l'entrée dans le tableau **$_SESSION** : `unset($_SESSION ['prenom'])`.

Pour supprimer d'un seul coup toutes les données de session, il est possible d'affecter un tableau vide (**array()**) au tableau **$_SESSION**.

4.2.5 session_abort

Syntaxe

```
bool session_abort()
```

La fonction **session_abort** annule les modifications apportées aux données de session depuis l'ouverture de la session et termine la session. Cette fonction retourne **TRUE** en cas de succès ou **FALSE** si une erreur survient.

Cette fonction n'annule pas les modifications immédiatement dans le script en cours. Pour annuler les modifications immédiatement dans le script en cours, il faut appeler la fonction **session_reset**.

4.2.6 session_reset

Syntaxe

```
bool session_reset()
```

La fonction **session_reset** réinitialise les données de session à leurs valeurs initiales

Cette fonction retourne **TRUE** en cas de succès ou **FALSE** si une erreur survient.

À la différence de la fonction **session_abort**, cette fonction annule les modifications immédiatement dans le script en cours.

4.2.7 session_destroy

Syntaxe

```
bool session_destroy()
```

Après un appel à la fonction **session_destroy**, la session n'existe plus ; un appel ultérieur à la fonction **session_start** va ouvrir une nouvelle session.

session_destroy retourne **TRUE** en cas de succès et **FALSE** en cas d'échec.

La fonction **session_destroy** échoue et affiche une alerte si elle est appelée avant la fonction **session_start**.

La fonction **session_destroy** ne supprime pas les données de session dans le script en cours.

De même, cette fonction ne détruit pas le cookie de session éventuellement utilisé pour propager l'identifiant de session.

4.2.8 session_status

Syntaxe

```
int session_status()
```

La fonction **session_status** retourne une constante donnant le statut actuel de la session :

- **PHP_SESSION_DISABLED** : Les sessions sont désactivées
- **PHP_SESSION_NONE** : Les sessions sont activées mais aucune session n'est démarrée
- **PHP_SESSION_ACTIVE** : Les sessions sont activées et une session est démarrée