

Librairie PHP pour sécuriser l'utilisation de Memcached

Pierre BLONDEAU

Laboratoire GREYC, Département d'informatique de l'Université de Caen Basse Normandie
Campus Côte de Nacre, Boulevard du Maréchal Juin
CS 14032
14032 CAEN cedex 5

Résumé

Memcached est un système ayant pour objectif d'augmenter la vitesse de réponse des sites Web associés à des bases de données. Il permet de stocker des données dans la mémoire vive afin d'accélérer les traitements. En effet, lors d'un traitement « lourd » (plusieurs secondes de calcul voire plusieurs dizaines de secondes), si l'on stocke le résultat dans Memcached, les futurs appels seront « instantanés ». L'utilisation de Memcached est de plus en plus courante dans les sites internet, mais il peut être utilisé dans toute autre application nécessitant un partage de données rapide.

Memcached est un démon qui permet de stocker tout type de données en mémoire vive afin d'accélérer les traitements. Il est principalement utilisé dans les infrastructures web. Il permet de conserver des données « statiques » de manière très rapides et partagées. En effet, lors d'un traitement « lourd » (plusieurs secondes de calcul voire plusieurs dizaines de secondes), on va stocker le résultat dans Memcached. Lors des futurs appels, les résultats seront récupérés en mémoire au lieu d'être recalculés. Il est également possible d'utiliser Memcached dans toute autre application nécessitant un partage de données « statique » rapide.

Dans le cas d'un serveur Memcache mutualisé (présence de plusieurs VHOST sur un serveur web par exemple), il y a un fort risque d'injection de code malveillant ou de vol de contenu.

La Secure Memcached Library propose la signature des données au moment de leur stockage dans Memcache. Lors de la récupération des données, la bibliothèque s'occupera de vérifier la signature et ne retournera le message seulement si elle est valide. Cette signature permet d'assurer la sécurité du site contre toute tentative malveillante d'injection de contenu, tout en ne nécessitant qu'un minimum de traitement.

Ces données demeurent cependant accessibles au « public ». Dans le cas où elles seraient plus sensibles (intranet par exemple), la bibliothèque peut également les chiffrer et les signer avant de les stocker dans Memcache. Ainsi seule la personne ayant chiffré les données pourra les lire, sous réserve que la signature soit valide.

Nous utilisons actuellement cette bibliothèque pour générer une page qui prend en temps normal plus de 3 secondes à s'afficher, ce qui peut paraître long. Le second chargement de cette page nécessite moins de 600 millisecondes.

Mots clés

Memcached, Sécurité, PHP

1 Introduction

Les réseaux et les machines sont de plus en plus rapides. De ce fait, nous supportons de moins en moins les lenteurs. Une page web s'affichant en plus d'une seconde nous fait penser qu'il y a un problème (connexion, navigateur, serveur, etc.).

Faites le test vous verrez ;).

Ce temps de réponse est également très important pour les moteurs de recherche qui tiennent compte de la vitesse d'affichage des pages lors de l'indexation. Les pages les plus rapides seront en effet mieux référencées.

Une des solutions pour améliorer ce temps de réponse consiste à stocker en mémoire vive (RAM) des éléments de votre site web relativement statiques et très souvent sollicités. Il s'agit en général d'éléments utilisés par toutes les pages et qui ont une durée de « validité » supérieure à 5 minutes. Plus la fréquentation de votre site est importante, plus la durée de validité pour laquelle il est intéressant de stocker en mémoire va diminuer. On peut, par exemple, l'utiliser pour des compteurs de visites ou d'articles.

2 Memcached

2.1 Présentation

Memcached [1] est un démon réseau qui permet de stocker des éléments directement en mémoire vive. Il possède de nombreuses bibliothèques permettant aux différents langages de programmation d'accéder à ses services. Le fonctionnement est très simple : il s'agit de stocker des couples clé / valeur.

L'utilisation classique consiste à insérer des données dans memcache lors du premier appel de la fonction. Lors des appels suivants, la valeur pourra être récupérée en utilisant la clé. Il ne sera plus nécessaire d'effectuer le traitement.

Une autre solution est d'avoir un programme en tâche de fond ou une tâche planifiée (cron) qui va enrichir ou mettre à jour Memcached avec les données qu'il aura calculé. Les données seront donc immédiatement accessibles dès le premier appel à votre page.

2.2 Contexte d'utilisation

Dans la plupart des cas, Memcached est mis en place pour accélérer un seul site. Il n'y a donc qu'une seule application qui y accède. Au Laboratoire GREYC et au Département d'informatique de l'Université de Caen, nous fournissons à nos enseignants / chercheurs et nos étudiants une page « personnelle ». Cette page leur sert à faire la promotion de leurs recherches ou de leurs cours (apprentissage de la programmation par exemple).

Étant donné le nombre de personnes, il nous est impossible de fournir un serveur complet (même virtuel) pour leurs propres sites. Les contraintes sont trop importantes, tant au niveau physique (taille des infrastructures d'hébergement) qu'au niveau du temps d'administration nécessaire. Nous fournissons donc des serveurs mutualisés (VHOST) [2] possédant eux-mêmes un démon Memcached.

2.3 Problèmes de sécurité

Par défaut, l'accès aux données de Memcached est public. C'est à dire que toute personne connaissant l'adresse IP et le Port du service Memcached est capable de lire et de modifier son contenu. Ceci pose deux problèmes :

1. La confidentialité des données n'est pas assurée. (Données en intranet par exemple).
2. Les sites deviennent sensibles aux tentatives d'injection de code malveillant.

Depuis la version 1.4.3, Memcached propose le support de l'authentification via SASL [3]. Cela permet d'obliger l'utilisateur à spécifier un identifiant et un mot de passe pour accéder aux fonctions de Memcached.

Cependant une fois l'utilisateur connecté, il lui est de nouveau possible d'accéder à toutes les données, y compris celles des autres utilisateurs. Cette fonctionnalité permet d'améliorer la sécurité sur le réseau pour un serveur distant, mais cela ne résout pas le problème d'accès au contenu dans le cadre d'un serveur mutualisé.

3 Secure Memcached Library

La Secure Memcached Library (SML) [4] est une bibliothèque PHP. Elle fournit deux méthodes pour sécuriser les données dans Memcached. Elle propose la signature et le chiffrement en utilisant les fonctions de chiffrement asymétrique d'OpenSSL [5] basé sur RSA [6]. La bibliothèque va générer et stocker de manière sûre une paire de clés publiques / privées. Elle chargera ces clés au premier appel et les utilisera tout au long de la session PHP.

3.1 La signature

SML utilise les fonctions de signature pour assurer l'intégrité des données. La bibliothèque va stocker le couple clé / valeur dans Memcached en y associant un autre couple clé / valeur contenant la signature. Lors des futurs appels, SML va récupérer les données en contrôlant la signature. La valeur ne sera donc retournée que si la signature est correcte. Attention, avec cette fonction les données demeurent lisibles par les autres utilisateurs du serveur.

3.2 Le chiffrement

Cette fonction consiste à chiffrer et signer les données avant de les stocker dans memcache. Les données seront donc illisibles pour tout autre utilisateur du serveur. Cependant, le chiffrement a un impact sur le temps de calcul. Il faudra donc plus de temps que pour une simple signature. Il convient donc de s'assurer que les données ainsi traitées ont un réel besoin de confidentialité.

3.3 Gestion des clés

Pour assurer la sécurité de vos données dans memcache, il faut garantir en priorité la sécurité de la clé de déchiffrement. Pour cela, il y a plusieurs solutions de stockage en fonction de votre environnement de mutualisation.

3.3.1 Stockage en mémoire partagée

Dans notre environnement, nous utilisons le serveur WEB Apache en version ITK [7]. Ce mode de fonctionnement permet d'exécuter chaque VHOST apache avec l'UID et le GID de l'utilisateur. Grâce à Apache ITK, nous pouvons « cloisonner » complètement nos utilisateurs. Comme chaque VHOST est exécuté par un utilisateur différent, nous pouvons utiliser des segments de mémoire partagée pour avoir un accès très rapide et sécurisé aux clés RSA.

3.3.2 Stockage dans le système de fichiers

Pour les autres cas, il faut stocker les clés dans des fichiers. Il faut s'assurer que ces fichiers ne sont pas accessibles par les autres VHOST / utilisateurs.

Dans la plupart des cas, Apache étant exécuté par un seul utilisateur (www-data), on doit donner à cet utilisateur un accès aux données de tous les sites web.

Afin d'éviter qu'un site puisse accéder aux données d'un autre site, le moyen le plus couramment utilisé est de générer une chaîne de caractères aléatoires pour le répertoire racine de chaque VHOST (par exemple : /var/www/ieteiGheiH0guluu/). Ainsi, si le contenu du répertoire /var/www et la configuration de Apache ne sont pas lisibles et que l'accès à des fonctions comme lsof sont interdits, alors un VHOST ne pourra pas accéder aux données d'un autre VHOST et donc pas non plus aux clés RSA.

4 Mise en oeuvre

La mise en oeuvre peut être décrite à travers un exemple. Nous souhaitons proposer une page contenant la liste des VHOST actifs hébergés sur un de nos serveurs web. Nous créons automatiquement tout les VHOST de nos utilisateurs, mais ils ne les utilisent pas forcément. Nous avons donc réalisé un script nous permettant de récupérer cette liste. Le problème est que ce script met environ 3 secondes à s'exécuter et qu'il est relativement gourmand en ressources (CPU + IO). Cette

page n'a pas besoin d'être mise à jour à chaque appel et nous avons considéré qu'une mise à jour toutes les 30 minutes était amplement suffisant.

Dans un premier temps, il faut initialiser la connexion avec le serveur Memcached. SML est compatible avec les modules « memcache » [8] et « memcached » [9] de PHP.

```
<?php
$memcache = new Memcache();
// ou $memcache = new Memcached();
$memcache->connect("localhost");
?>
```

Ensuite, il faut choisir un mode de stockage pour charger les clés de signature / chiffrement et enfin lancer la bibliothèque pour stocker et accéder aux données.

```
<?php
$storage = new SecureMemcachedLibraryStorageSHM();
$sml = new SecureMemcachedLibrary($storage, $memcache);
?>
```

Maintenant, l'objet \$sml fournit des méthodes pour enregistrer et lire nos données. La première chose à faire est de vérifier si les données, dont nous avons besoin, ne sont pas déjà dans le serveur Memcached.

```
<?php
$res = $sml->getSignArray("sites");
// $res = $sml->getDecryptArray("donnéesconfidentielles");
?>
```

Si les données sont dans Memcached, \$res les contiendra et nous pouvons directement les utiliser. Dans le cas contraire, il va falloir générer ces données avec une méthode php ou un exec par exemple. Il faudra ensuite les enregistrer dans Memcached pour les prochains appels.

```
<?php
if($res) {
    $sites=$res;
} else {
    exec("sudo /usr/local/sbin/list-site",$sites);
    $sml->setSignArray("sites",$sites, 1800 );
    // $sml->setEncryptArray("donnéesconfidentielles",array(""));
}
?>
```

Vous pouvez maintenant utiliser l'objet \$sites à votre convenance. Lors des prochains appels, les données seront automatiquement récupérées dans Memcached.

Démonstration : <https://blondeau.users.greyc.fr/sml/>

5 Conclusion

Nous utilisons cette bibliothèque depuis maintenant plus d'un an. Elle nous permet d'afficher nos liste de sites en moins de 150ms :

	Étudiants	Enseignants / Chercheurs
Premier Appel	4.200 s	2.450 s
Appel Suivant	0,140 s	0.130 s

Nous envisageons d'améliorer notre solution en provisionnant Memcached par une tâche planifiée (CRON). Il suffira de mettre une limite de validité supérieure à l'intervalle de mise à jour (par exemple, 35 minutes pour un cron exécuté toutes les 30 minutes). On évitera ainsi l'effet « premier appel » .

Bibliographie

- [1] Memcached. <http://memcached.org/>.
- [2] Apache virtualhost (vhost). <http://httpd.apache.org/docs/current/vhosts/examples.html>.
- [3] Memcached sasl howto. <http://code.google.com/p/memcached/wiki/SASLHowto>.
- [4] Blondeau P.. Secure memcached library, 2012. <https://forge.greyc.fr/projects/sml>.
- [5] Php fonctions openssl. <http://www.php.net/manual/fr/ref.openssl.php>.
- [6] Chiffrement rsa. http://fr.wikipedia.org/wiki/Chiffrement_RSA.
- [7] The apache 2 itk mpm. <http://mpm-itk.sesse.net/>.
- [8] Php memcache. <http://php.net/manual/fr/book.memcache.php>.
- [9] Php memcached. <http://php.net/manual/fr/book.memcached.php>.