ZEND | FLASH&FLEX | PHP 5.3 | CUTEFLOW | POO

Le plus grand magazine sur PHP au monde

phpsolutions

Nouvelles technologies et solutions pour les développeurs PHP

PHP N° 6/2010 (42) ISSN 1731-4593

ZEND FRAMEWORK
TOUTE LA PUISSANCE DE PHP !

FLASH ET PHP
COMMUNICATION FLEX 4 ET PHP 5.3

PROGRAMMATION ORIENTÉE OBJET
UN STYLE DE PROGRAMMATION PUISSANT ET COMPLEXE

VIRTUALISATION SOUS LINUX
TECHNOLOGIES ET SOLUTIONS

OUTILS

SYSTÈME DE FLUX DE TRAVAIL
DÉCOUVREZ LES POSSIBILITÉS DE CUTEFLOW

POUR LES DÉBUTANTS

MANIPULEZ LES SESSIONS AVEC PHP
DÉMARREZ ET RESTAUREZ UNE SESSION

HOSTING NEXT LEVEL

intel inside CORE i7

HETZNER DEDICATED ROOT SERVER EQ 4

- Intel®Core™ i7-920 Quad-core avec technologie Hyper-Threading
- 8 Go DDR3 RAM
- 2 x 750 Go SATA-II HDD (Software-RAID 1)
- Système d'exploitation Linux
- Windows Server à partir de 13 € par mois
- Trafic réseau illimité[1]
- Système « Rescue »
- 100 Go d'espace de sauvegarde
- Domain Registration Robot
- Pas de contrat minimum
- Frais d'installation : 126 €

42,- € par mois

HETZNER DEDICATED ROOT SERVER EQ 8

- Intel®Core™ i7-920 Quad-core avec technologie Hyper-Threading
- 24 Go DDR3 RAM
- 2 x 1500 Go SATA-II HDD (Software-RAID 1)
- Système d'exploitation Linux
- Windows Server à partir de 13 € par mois
- Trafic réseau illimité[1]
- Système « Rescue »
- 100 Go d'espace de sauvegarde
- Domain Registration Robot
- Pas de contrat minimum
- Frais d'installation : 126 €

75,- € par mois

HETZNER DEDICATED ROOT SERVER EQ 9

- Intel®Core™ i7-975 Quad-core avec technologie Hyper-Threading
- 12 Go DDR3 RAM
- 3 x 1500 Go SATA-II HDD (Software-RAID 5)
- Système d'exploitation Linux
- Windows Server à partir de 13 € par mois
- Trafic réseau illimité[1]
- Système « Rescue »
- 100 Go d'espace de sauvegarde
- Domain Registration Robot
- Pas de contrat minimum
- Frais d'installation : 126 €

84,- € par mois

1 Le trafic est gratuit. Nous limitons la vitesse à 10 Mbit/s si 5000 Go/mois sont dépassés. En option, une bande passante permanente garantie de 100 Mbit/s est proposée à 6 € par To supplémentaire.

2 En tant que nouveau client, vous pouvez bénéficier de 8 € de réduction sur n'importe quel produit présenté ici. Utilisez, s'il vous plait, le code promo **011907** en remplissant votre bon de commande. Cette offre expire le 01.08.2010.

Tous les tarifs s'entendent HT et sont soumis aux conditions de vente de Hetzner Online AG. Les prix sont susceptibles d'être changés. Tous droits réservés par les fabricants respectifs.

HETZNER ONLINE

« Hosting next level » signifie simplement que Hetzner Online propose aujourd'hui la plus puissante des solutions d'hébergement dédié actuellement disponibles ! Les plans serveurs dédiés Hetzner Online sont conçus pour une rapidité maximale et une stabilité réseau extrême dans nos propres datacenters basés en Allemagne. Avec nos tarifs compétitifs et un support hors du commun, nous dépassons déjà les exigences de nos clients partout dans le monde.

www.hetzner.info

info@hetzner.com

N'ayez plus honte de faire votre site web...

REFLEXION**FAITE** - © Crédits photos : Image Source.

Avec les packs WebSite,
créez un site web élégant et performant en quelques clics.

DÉCOUVREZ UNE SOLUTION SIMPLE, RAPIDE ET INTUITIVE POUR CRÉER UN SUPERBE SITE WEB : LE VÔTRE.

- 250 modèles de sites de grande qualité
- Bibliothèque de 1000 images haute résolution
- 14 jours d'essai gratuits sans engagement
- Un seul pack qui couvre tous vos besoins (nom de domaine, hébergement, email...)
- Amen, un interlocuteur unique

DOMAINE · EMAIL · HÉBERGERMENT · SERVEUR · PUBLICITÉ · PROTECTION DE MARQUE · E-COMMERCE · CRÉATION DE SITE

Amen
A DADA COMPANY

0 892 55 66 77
(0,34 €/mn)

www.amen.fr

Le périodique *phpsolutions* est publié par
Software Press Sp. z o.o. SK
Bokserska 1, 02-682 Varsovie, Pologne
Tél. 0975180358, Fax. +48 22 244 24 59
www.phpsolmag.org

Président de Software Press Sp. z o.o. SK :
Paweł Marciniak

Directrice de la publication :
Ewa Łozowicka

Dépôt légal :
à parution
ISSN : 1731-4593

Rédacteur en chef :
Łukasz Bartoszewicz

Maquette :
Agnieszka Marchocka

Couverture :
Sławomir Sobczyk

DTP :
Sławomir Sobczyk *Studio2W@gmail.com*

Composition :
Sławomir Sobczyk

Correction :
Valérie Viel, Jonathan Marois, Thierry Borel

Bêta-testeurs :
Fabrice Gyre, Brice Favre, Valérie Viel, Cyril David, Aymeric Lagier, Christophe Milhau, Alain Ribault, Stéphane Guedon, Eric Boulet, Mickael Puyfages, Christian Hernoux, Isabelle Lupi, Antoine Beluze, Timotée Neullas, Yann Faure, Adrien Mogenet, Jean-François Montgaillard, Turmeau Nicolas, Jonathan Marois, Wilfried Ceron, Wajih Letaief, François Van de Weerdt, Jeremy Rafflin, Eric Vincent.

Les personnes intéressées par la coopération sont priées de nous contacter :
editor@phpsolmag.org

Publicité :
publicite@software.com.pl

Pour créer les diagrammes on a utilisé le programme
smartdraw.com
SmartDraw

TABLE DES MATIÈRES

VARIA

OUTILS

PROJETS

DOSSIER

LINUX

PRATIQUE

POUR LES DÉBUTANTS

Wordpress 3

La nouvelle version de *Wordpress* vient de sortir. Celle-ci est une version majeure car elle propose de nombreuses nouveautés et de nouvelles fonctionnalités. Par ailleurs, vous allez pouvoir personnaliser le nom du compte administrateur comme vous le souhaitez, au lieu du classique *admin*. Vous pourrez aussi créer plusieurs blogs en même temps avec la même interface. Enfin, les thèmes et les plug-ins sont centralisés pour simplifier les installations et les configurations.
http://fr.wordpress.org/

Tests Unitaires

Les tests unitaires sont souvent un point mal traité (ou pas du tout) lors d'un développement. Une liste de 10 points a été réalisée par Anna Filina montrant qu'il est important de le faire. Ainsi, votre code sera assuré du bon fonctionnement de votre projet.
http://annafilina.com/blog/10-reasons-to-write-unit-tests/

Atutor

Atutor est une application écrite en PHP et MySQL. Il s'agit d'un LCMS/LMS (*Learning Content Management System*). Cette application possède de nombreuses fonctionnalités comme l'accessibilité, le réseau social, la sécurité, les cours, la messagerie, les profils, le stockage, les logs, les rédactions, les outils.
http://www.atutor.ca/

PHP 400 v2

PHP400 est un environnement d'applications de gestion en PHP pour AS/400-IBM. Cette application s'appuie sur Zend Core et se décompose d'un framework, d'un portail, d'un générateur d'applications, d'un requêteur SQL et des outils pour le développement.
http://www.phlsoft.com/

Joomla 1.5.18 et 1.6 beta 4

Le CMS Joomla vient de sortir une nouvelle version de PHP 1.5 avec quelques correctifs importants, qu'il faut prendre en compte, et qui est compatible avec PHP 5.3. Par ailleurs, la nouvelle version 1.6 est disponible avec une nouvelle version béta, ce qui devrait permettre de voir prochainement cette nouvelle version stable.
http://www.joomla.fr

Drupal 6.17

Le CMS Drupal vient de sortir la nouvelle version 6.17 qu'il est important de mettre à jour car de nombreux bugs ont été corrigés par rapport aux versions précédentes.
http://www.drupal.org

XMP PHP Toolkit extension 2.0

XMP PHP Toolkit est une nouvelle barre qui vous sera utile. Cette barre est inspirée de *Adobe XMP Toolkit* et réalisée par Mathias Vitalis, sous licence GPL v3. Grâce à elle, vous allez :

- Lire.
- Ecrire.
- Modifier.
- Parser.
- Parcourir les métadonnées d'un fichier multimédia.

C' est une extension de PHP 5 et vous pouvez utiliser les classes XMP d'*Adobe* et les différentes méthodes proposées : lectures, modifications, extraction des métadonnées des fichiers multimédia comme :

- les images (tiff, jpeg, eps, pdf, psd),
- l'audio (mp3, wav),
- la vidéo (avi, mpeg et mov).

Grâce à cette barre, les manipulations des ces fichiers devienent plus faciles et par conséquent rapides. Cette nouvelle version qui vous est proposée, est compatible avec PHP 5.3 et tourne avec de nouveaux OS. Ainsi, vous pourrez faire fonctionner cette barre sous Linux, Débian 32/64 bits, FreeBSD et Mac OS X.
http://xmpphptoolkit.sourceforge.net/

Les modules d'EasyPHP

EasyPHP est l'un des tout premiers AMP (Apache, MySQL, PHP) pour Windows. Depuis de nombreux mois, une version 2.0 a vue le jour avec les dernières versions disponibles pour chaque environnement.

Différentes versions de bases existent, permettant de proposer d'avoir :

- Choix 1 : PHP 5.3.2, Apache 2.2.15, MySQL 5.1.45, PhpMyAdmin 3.3.2.
- Choix 2 : PHP 5.3.2, Apache 2.2.13, MySQL 5.1.37, PhpMyAdmin 3.2.1, SQLite 2.8.17, Pecl 5.2.6.

Maintenant *EasyPHP* rejoint les nombreux outils qui vont vous aider à réaliser des sites internet. Ces outils ou environnements sont principalement les CMS et les boutiques en lignes. Pour utiliser ces outils, vous devez posséder dans votre ordinateur, un environnement AMP comme *EasyPHP*. Or, souvent, un certain nombre de paramètres et de configurations sont nécessaires pour les faire fonctionner. *EasyPHP* propose des modules supplémentaires que vous devez installer mais déjà paramétrés pour que vous n'ayez presque rien à faire.

Les outils que vous pouvez utiliser avec un minimum de clics sont : Spip, Wordpress, Prestashop. Bien entendu, vous pouvez pré-installer aussi Joomla ou Drupal très facilement. Enfin, *EasyPHP* est aussi un environnement, c'est à dire que vous pouvez l'installer sur différents supports comme un disque amovible, une clé USB ce qui vous permettra de l'utiliser sur différents ordinateurs.
http://www.easyphp.org

Ne soyez plus à la merci de votre hébergeur

30% de réduction
grâce à phpsolutions*

B
POWERS

l'hébergement autrement

B-Powers c'est :

- une entreprise de taille humaine
- un service professionnel
- des solutions flexibles
- un support personnalisé

*Promotion applicable la première année pour toute souscription annuelle.
Valable jusqu'au 31 août 2010. Code promotionnel : phpsolutions

www.bpowers.com

Intertview
de Philippe Montarges, co-président d'Alter Way

Philippe Montarges,
co-président d'Alter Way.

PHP Solutions : Bonjour, tout d'abord nous vous remercions de nous accorder du temps pour cette interview. Pourriez-vous raconter à nos lecteurs sur quels projets vous êtes en train de travailler ?
Philippe Montarges : Nous sommes un opérateur de service Open Source et à ce titre nous intervenons sur des projets de conseil et de réalisation, mais également sur de l'hébergement et de la formation sur 4 socles technologiques principaux : PHP, Python, Java et Ruby.

Concernant plus particulièrement la plateforme PHP, nous travaillons actuellement à la fois sur des projets d'intégration de solutions aussi bien dans la gestion de contenu avec notamment Drupal, que dans le commerce électronique avec Magento, ou encore dans le CRM avec SugarCRM, mais également sur des projets construits intégralement sur des développements spécifiques en PHP. Dans ce cas, nos appuyons systématiquement nos développements sur un framework : Symfony ou Zend Framework, en fonction de la typologie de projet.

PS: Comment Alter Way voie l'avenir des technologies HTML 5 et CSS 3 et quelles actions faites-vous pour vous positionner sur ce marché ?
PM : HTML5 et CSS 3 sont le moyen :

- De transformer le web en véritable plateforme multifonction. Il s'agit de décupler les possibilités d'une application Web pour tendre vers les possibilités d'une application de bureau. On peut imaginer que demain tout se passera sur le Web, et on commence à en voir l'expression : Google Docs permet de s'affranchir d'une suite bureautique, les nouveaux OS proposent l'accès à des services Internet plutôt que des logiciels classiques sur le disque dur de la machine, les nouveaux matériels tels que l'*iPad* sont également résolument tournés vers les services en ligne.
- De concevoir des applications qui peuvent travailler indifféremment en mode connecté ou déconnecté. HTML5 donne la possibilité d'utiliser des bases de données locales structurées. Les API bases de données HTML5 ont un mode de fonctionnement asynchrone évitant le blocage du navigateur en cas de requêtes importantes.
- De créer un Web ouvert : HTML5, correctement utilisé, permet de s'affranchir de bon nombre de plug-ins propriétaires (par exemple Flash) et de s'appuyer de plus en plus sur des standards. Ceci va bien sûr dans le sens d'une interopérabilité accrue. Un Web ouvert est un gage d'évolutivité et de pérennité du médium, au même titre que les solutions Open Source au sens large, le sont.

HTML5 et CSS 3 vont donc accroître la productivité des utilisateurs, en leur permettant d'obtenir des applications riches où et quand ils le souhaitent.

Reste une certaine incertitude, malgré l'implication désormais réelle des 4 grands éditeurs de navigateurs *Microsoft ; Apple ; Mozilla* et *Opera*, sur la date de sortie effective de cette nouvelle version.

PS : Est-ce que vous pensez, comme beaucoup de contributeurs, que la mort de PHP 6 ne changera rien au niveau de la confiance qu'accordent les entreprises en cette technologie ?

PM : Difficile de parler de la mort d'un projet qui n'est même pas né! La fin de PHP6 témoigne de la capacité de la communauté PHP à admettre ses erreurs et repartir sur de nouvelles bases plus saines. Dans ces conditions, les entreprises devraient être rassurées quant à la maturité du langage et de sa communauté. Il est également du rôle d'Alter Way d'évangéliser autour de ces événements et d'en expliquer les enjeux.

La démarche choisie pour développer PHP6 s'est avérée inadaptée, car techniquement trop complexe. Elle a fini par décourager les contributeurs qui ont reporté leur énergie sur la branche 5 ce qui a entraîné la sortie de la version PHP5.3 qui n'a de mineure que le nom. Au bout du compte, la version 5.4 reprendra probablement l'essentiel des fonctionnalités de la 6 et sera un bon tremplin pour une nouvelle version majeure de PHP.

PS : Quels sont les projets à venir d'Alter Way ?

PM : En ce qui concerne le développement de nouvelles activités, nous menons actuellement une réflexion sur l'intégration des technologies de virtualisation d'Ubuntu et l'opportunité de devenir un centre de 'private open cloud'. Nous étudions les initiatives portées dans ce domaine par les pôles de compétitivité, notamment le projet Compatible One (un projet de cloud ouvert mené par *System@tic* et SCS).

Par ailleurs en ce qui concerne le développement de la société, nous allons procéder à de nouvelles opérations de croissance externes d'ici à la fin de l'année. Nous annoncerons d'ailleurs dans quelques jours une acquisition dans le secteur du Web qui viendra renforcer notre activité Solutions (intégration, développement, infogérance et TMA).

Enfin, afin de soutenir nos objectifs de croissance pour l'année 2010 (15 millions d'euros de chiffre d'affaires consolidé, dont 11 millions en croissance organique), nous avons mis en place une politique de recrutement très volontariste avec près de 50 postes à pourvoir d'ici la fin de l'année, soit une croissance de 40% de notre effectif.

PS : Pourquoi, à votre avis, PHP a-t'il pris autant d'importance dans le monde web ?

PM : L'origine du succès originel de PHP tient probablement à sa simplicité de prise en main et la facilité avec laquelle on peut produire une page web fonctionnelle. Cela a souvent permis à PHP de combler son retard entre les applications DSI structurantes et les défis métiers du quotidien. Aujourd'hui, ces applications de confort deviennent des applications critiques. L'industrialisation est une nouvelle frontière pour PHP : la capacité du langage à être industrialisé, à produire des projets de qualité professionnelle dans des délais et des budgets maîtrisés.

Un autre avantage toujours à l'ordre du jour : les coûts d'hébergement d'applications PHP sont très réduits. Il est beaucoup plus économique de concevoir et utiliser des plateformes d'hébergement en PHP qu'en Java ou Ruby par exemple.

PS : Est-ce que vous pensez qu'actuellement obtenir une certification PHP 5 ou Zend Framework est un réel plus, ou juste une décoration ?

PM : Ces certifications exigent toutes deux un très bon niveau dans les capacités testées. Elles incitent les développeurs qui les passent à faire le point sur leurs connaissances. Elles permettent véritablement de juger de la qualité du profil certifié.

La certification représente donc un atout à plusieurs niveaux : la confiance qu'un client peut apporter à un prestataire qui possède des profils certifiés, mais aussi être rassuré sur les capacités d'un candidat certifié lors d'un processus d'embauche. De plus, ces certifications participent encore une fois au caractère industriel des composants concernés.

PS : Comment vous voyez le web de demain et du développement de PHP dans les années à venir ?

PM : Massivement orienté *cloud*. Massivement multiplateformes. Un web orienté productivité et social notamment grâce à HTML 5 et CSS 3.

Après 5 années au cours desquelles PHP est passé du stade d'outsider au stade d'outil professionnel majeur, le langage entre à présent dans une phase d'industrialisation. Les premiers outils sont déjà bien établis (tests unitaires, tests fonctionnels, intégration continue, analyse de code) et n'ont pas à rougir face à ceux disponibles pour d'autres plateformes. D'autre part, une initiative a été lancée par les meneurs des principaux projets PHP (Symfony, Zend Framework, Drupal, Joomla, PEAR, Doctrine, etc.) afin de définir des règles communes visant à améliorer l'interopérabilité entre les librairies PHP. À cela s'ajoute l'arrivée d'outils clairement orientés entreprise comme Zend Server, une pile LAMP commercialisée par Zend qui propose un support commercial. On voit là la capacité des acteurs tant commerciaux que communautaires du monde PHP à travailler ensemble pour faire évoluer cette technologie de façon très pragmatique par rapport aux attentes des utilisateurs.

PS : Merci beaucoup pour le temps que vous avez bien voulu nous accorder.

Le système de flux de travail CuteFlow

Plusieurs systèmes de collaboration électroniques sont maintenant disponibles, mais ces systèmes sont souvent trop complexes pour de petites équipes.

Cet article explique :

- Cet article explique comment installer, configurer et utiliser le système de flux de travail CuteFlow.

Ce qu'il faut savoir :

- Cet article s'adresse à des personnes qui veulent améliorer leur vie professionnelle ayant une connaissance générale sur PHP/MySQL.

Aujourd'hui, des pressions internes à l'entreprise et externes nous imposent d'évoluer constamment. C'est de toute évidence le moyen le plus efficace pour stimuler le dynamisme d'une entreprise. Pour cette raison, un système de flux de travail, permettant la circulation et la validation des tâches à accomplir entre différents acteurs d'un processus, constitue un élément fondamental dans une entreprise.

Un outil open source appelé CuteFlow [1] prend en charge les flux de documents inter-plate-forme, il permet la circulation et la validation de documents entre les utilisateurs tout en gérant les permissions de chacun [2].

C'est un logiciel de travail collaboratif proposant un flux de travail. Il est fondé sur du PHP/MySQL [2].

Dans cet article, nous vous présentons tout d'abord les fonctionnalités du logiciel, son installation et sa configuration ; par la suite, nous expliquerons la circulation et la validation des documents.

Fonctionnalités

Les fonctionnalités du logiciel sont :

- Création de listes d'utilisateurs ;
- Création de modèles de documents ;
- Définition de la circulation du document ;
- Téléchargement du document à faire circuler ;
- Multi-langues ;
- Stockage des données dans une base de données MySQL ;
- Programmation en PHP [2].

Installation

La circulation des documents par le système de workflow CuteFlow nécessite un serveur web (Apache 1.3.x ou plus récent), avec PHP et MySQL comme base de données et un serveur de courrier électronique que vous pouvez régler via SMTP.

L'archive d'installation des fichiers CuteFlow est disponible sur le site de *SourceForge* : *http://downloads.sourceforge.net/cuteflow*

Il vous suffit alors de la décompacter directement dans le répertoire public de votre serveur web. Ensuite, n'oubliez pas de remplacer l'adresse IP par celle de votre serveur web.

Le programme lance immédiatement l'installation qui comprend cinq étapes :

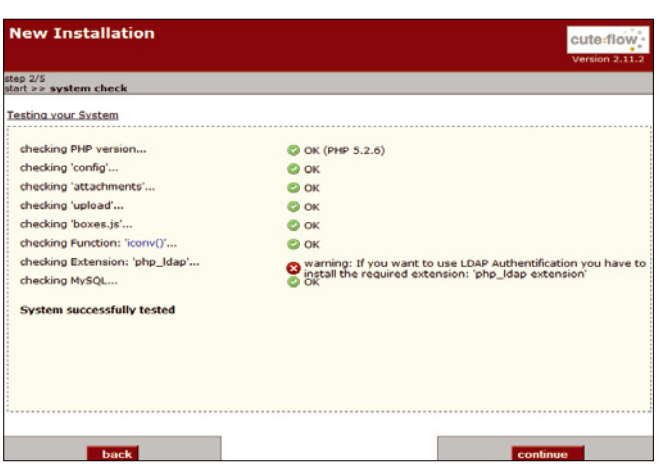


Figure 1. *L'interface de vérification*

- Cliquez sur le bouton ***Installer*** ;
- L'étape suivante consiste à vérifier que votre système répond aux exigences et que les autorisations pour les répertoires et les fichiers sont correctement configurés (figure 1) ;
- Durant cette étape, vous fournissez certaines informations concernant le type de la base de données qui est MySQL, le nom d'hôte de votre serveur MySQL, le nom de la base de données, le nom de l'utilisateur et le mot de passe. Si vous avez décompressé l'archive dans le répertoire racine de documents en tant que root, l'utilisateur d'administration sera le propriétaire des répertoires et des fichiers. En appuyant sur le bouton ***Continue***, CuteFlow confirmera que la base de données est installée avec succès ;
- Ensuite, en poursuivant l'installation, le programme vous invite à fournir des détails concernant votre serveur SMTP. Vous n'avez pas besoin d'installer un serveur de courrier : utilisez votre fournisseur de messagerie électronique à la place. Si plus tard, vous rencontrez des problèmes pour envoyer des e-mails avec CuteFlow, laissez votre nom d'utilisateur, votre mot de passe et la case d'authentification vides afin d'être en mesure d'envoyer des messages aux utilisateurs du système indépendamment des circulations.
- Le programme vient compléter l'installation en vous donnant la possibilité de mettre en place une base de données test. Si vous souhaitez tester CuteFlow avec des utilisateurs prédéfinis et des modèles de documents, cochez la case ***Install test data*** et entrez une adresse e-mail (Figure 2) ;

Configuration

Après avoir terminé l'installation, vous êtes porté au front-office. Une fois que vous ouvrez une session avec ***admin*** comme nom d'utilisateur et ***admin*** comme mot de passe (Figure 3), le programme ouvre automatiquement la page ***A faire*** (Figure 4) ; cette page vous donne un aperçu des circulations actuellement dans l'attente de votre nœud.

Sur le côté gauche, vous pouvez consulter le programme des quatre fonctions principales : ***Circulations***, ***Gestion***, ***Administration*** et ***Déconnexion***.

En tant qu'administrateur, vous pouvez maintenant configurer CuteFlow, ajouter et supprimer un utilisateur, modifier la base de données, le serveur de messagerie et les paramètres de front-office.

En cliquant sur ***Utilisateur***, vous voyez une liste des employés, qui comprend des données typiques telles que les noms et les prénoms, les noms des utilisateurs, leurs e-mails, un indicateur informant si l'employé est en ligne ou hors ligne, des options sous forme d'icônes donnant le privilège soit de supprimer un utilisateur, soit de montrer des détails de son compte.

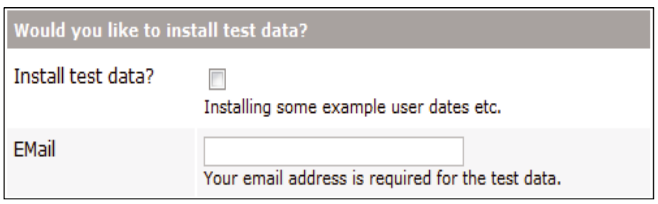


Figure 2. *L'interface de mettre en place une base de données test*

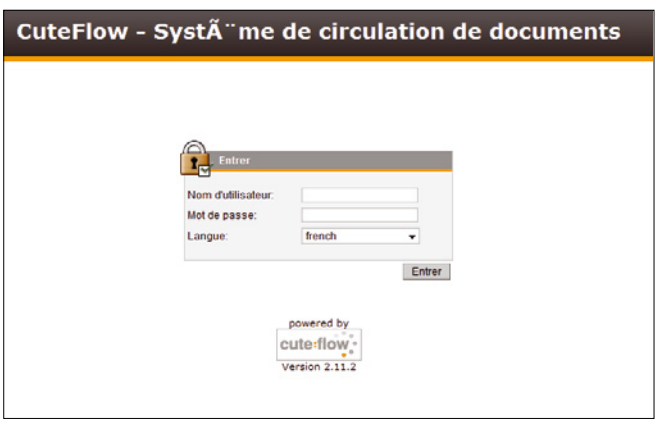


Figure 3. *L'interface d'authentification*

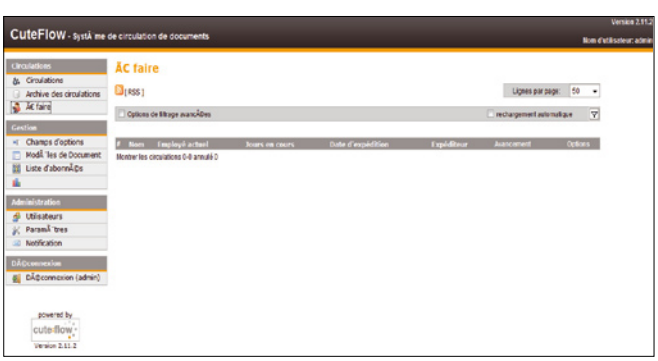


Figure 4. *La page A faire*

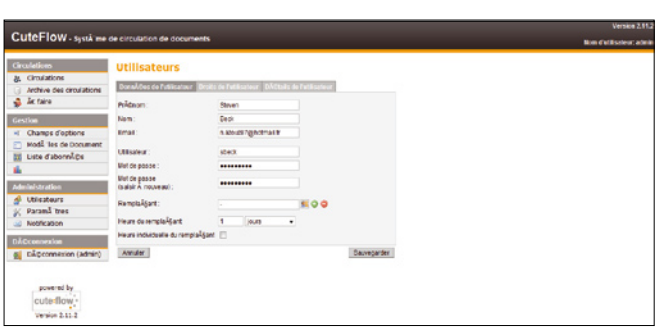


Figure 5. *La page d'ajout d'un utilisateur*

Figure 6. *La page de notification*

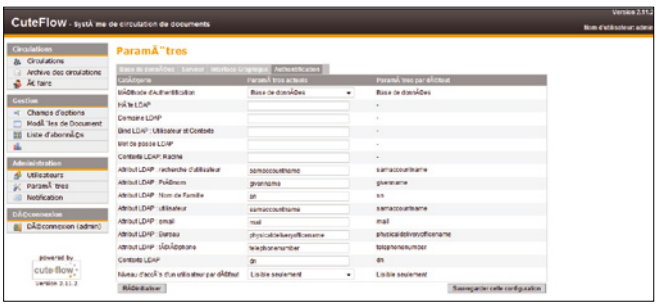


Figure 7. *L'interface de détermination des paramètres LDAP*

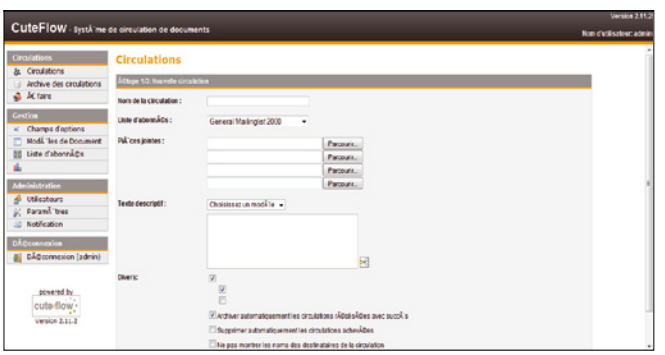


Figure 8. *L'interface de création d'une circulation*

Figure 9. *L'interface de la liste d'envoi*

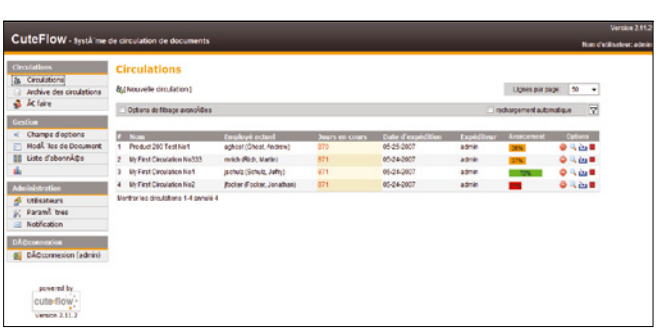


Figure 10. *La page présentant toutes les circulations*

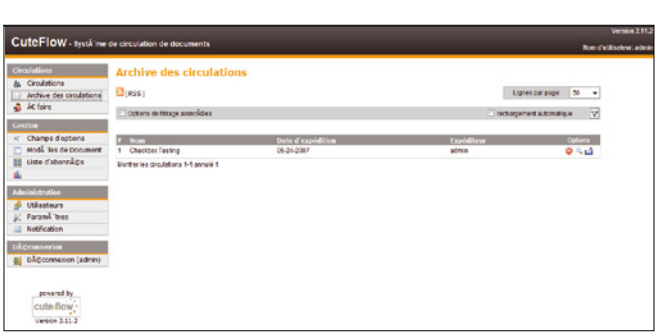


Figure 11. *La page des circulations archivées*

Le lien ***Nouveau Utilisateur*** vous permet d'ajouter un nouvel utilisateur, en vous affichant une boîte de dialogue dans laquelle vous êtes amené à fournir les informations concernant cet utilisateur (Figure 5). Assurez-vous que vous avez ajouté une adresse e-mail valide ainsi que le nom d'utilisateur, car CuteFlow a besoin de cette adresse pour informer l'utilisateur qu'un document est en attente.

L'administrateur peut utiliser l'adresse e-mail pour envoyer des messages aux utilisateurs enregistrés (Figure 6).

Une chose très importante à savoir est la façon dont vous définissez les privilèges d'un nouvel utilisateur. Lorsque vous créez un utilisateur, en choisissant le bouton radio dans l'onglet ***Droit de l'utilisateur,*** le système fait la distinction entre ***Administration, Expéditeur, Lisible seulement et Destinataire.***

Les deux dernières autorisations permettent à l'utilisateur de consulter le document en circulation. Les administrateurs et les expéditeurs sont autorisés à les renvoyer et à lancer ou supprimer des circulations.

Au niveau de la fonction ***Administration***, dans l'élément ***Paramètres,*** vous trouvez la configuration de la base de données, le serveur de messagerie, l'interface graphique et l'authentification.

L'onglet ***Base de données*** répertorie les paramètres configurés lors de l'installation du programme et ne vous permet pas de faire des changements.

L'onglet ***Serveur*** est l'endroit où vous saisissez l'adresse d'hôte sur lequel le serveur CuteFlow réside.

Pour que le programme fonctionne, vous devez spécifier un serveur SMTP que le programme utilise pour expédier des messages aux utilisateurs.

L'onglet ***Interface graphique*** vous permet, dans une certaine mesure, de modifier les éléments de présentation graphique. Ainsi, vous pouvez changer la langue, le nombre de lignes par page, l'ordre de tri, l'ordre des colonnes dans les circulations de documents et bien d'autres choses.

Après avoir changé les paramètres de base, assurez-vous de cliquer sur le bouton ***Sauvegarder cette configuration*** en bas de la page.

Le bouton ***Réinitialiser*** est un outil pratique qui réinstalle les valeurs par défaut si les choses tournent mal.

L'onglet ***Authentification*** vous permet de définir les paramètres de LDAP, qui seront éventuellement utilisés comme méthode d'authentification (Figure 7).

Circulation et Validation

Pour lancer votre première circulation de documents avec CuteFlow, vous devez définir les champs d'options, le modèle de document et la liste de diffusion.

Les ***Champs*** vous permettent de définir la façon dont les utilisateurs sont autorisés ou non à ajouter des informations à la circulation. CuteFlow distingue différents types d'information.

Vous pouvez même permettre aux utilisateurs de télécharger des fichiers en pièces jointes à la circulation.

Vous pouvez transmettre des vidéos ou des enregistrements audio sous forme de pièces jointes à la diffusion des documents.

Le modèle de documents vous permet de créer un modèle associé à un service.

A l'étape suivante, lorsque vous ajoutez des listes d'abonnés, vous décidez des associations de modèles aux documents. Vous pouvez attribuer un modèle différent à chaque liste.

Pour lancer une nouvelle circulation, allez à l'onglet ***Circulations***, cliquez sur l'élément ***Circulations***, puis sur le lien ***Nouveau circulation***, saisissez le nom de la circulation et ajoutez une pièce jointe (Figure 8). En cliquant sur le bouton ***Suivant***, les boîtes de dialogue qui suivent vous permettent de modifier la liste d'envoi (Figure 9) et les paramètres du modèle.

En cliquant sur le bouton ***Terminer***, votre circulation est activée (Figure 10). Enfin, pour la validation des documents, en cliquant sur l'icône de la circulation ***Modifier*** dans la colonne ***Options*** et si le document dans votre nœud est accepté, vous pouvez passer la circulation au nœud suivant.

Quand la circulation du document est terminée, CuteFlow l'a mise en archive (Figure 11) et fait automatiquement la mise à jour de la liste des circulations des documents.

Conclusion

Vous venez de découvrir le système de workflow CuteFlow qui est un système de collaboration électronique simple pour les petites équipes.

Il ne nous reste plus qu'à vous souhaiter bon courage dans l'utilisation de ce système qui vous fera gagner du temps.

AZZA NAFTI

Maîtrisard en informatique. Elle travaille dans une société de développement web Synergy Space (http://www.synergy-space.com/). Les divers projets qui lui ont été confiés l'ont amenée à enrichir ses connaissances en matière de conception et de développement dans divers langages et de se passionner pour le développement web. Contact : n_azza@hotmail.fr.

Communication
Flex 4 et PHP 5.3

Toute tentative de lancer un débat stérile mise à part, si HTML5 est actuellement sous les feux de la rampe, Flash reste bien présent sur le web. Il existe beaucoup d'options pour communiquer avec la couche métier d'une application, le protocole AMF en fait partie.

Cet article explique :

- Mettre en place une communication entre Flash et PHP via le protocole AMF, en utilisant le composant AMF du Zend Framework (version 2.0). Utilisation de GIT pour le versionning.

Ce qu'il faut savoir :

- Connaître les bases de Flash/Flex, son système évènementiel et sa syntaxe orientée objet. Côté PHP, c'est PHP 5.3 qui sera utilisé afin d'introduire la notion de namespaces.

Dans le cadre d'un projet, il est préférable d'utiliser un *Système de Contrôle de Version* (SCV, en anglais *Control Versioning System* (CVS)) facilitant le travail en équipe, le suivi de version et apportant beaucoup d'autres avantages qu'il serait fastidieux de détailler ici, d'autant qu'il ne s'agit pas du but premier de cet article. Aujourd'hui, il existe différent système de contrôle de version :

- CVS : un peu âgé aujourd'hui, il est toujours utilisé par la communauté de FreeBSD, par exemple, et ce pour des raisons historiques.
- SVN : beaucoup utilisé aujourd'hui, SVN est un outil classique de suivi de version.

Ces deux exemples ont en commun le coté *centralisé*, c'est à dire que l'utilisateur **crée** sa copie de travail (*Working Copy*, en anglais) en local sur son poste, mais envoie ses modification sur un serveur distant. Ce principe de centralisation du suivi de version n'est pas sans contraintes, c'est pourquoi de plus en plus aujourd'hui on lui préfère des outils comme GIT ou Bazaar, qui sont décentralisés. A contrario d'un SVN, l'utilisateur aura son propre serveur de suivi de version sur sa machine, et une liste de **dépôts** *distants* (en anglais, *remote*) depuis lesquels il va récupérer les modifications effectuées (c'est l'action de *pull* ou de *fetch*), et vers lesquels il va envoyer ses modifications (action de *push*). Les avantages sont multiples (possibilité de ***commiter*** ses modifications sans les envoyer à ses collègues directement, système de gestion de branches extrêmement puissant, copie intégrale du dépôt en cas de crash d'un des dépôts distants). Pour plus d'informations, le livre *Pro Git http://progit.org/book/* (rédigé par Scott Chacon) est une excellente source d'information sur l'utilisation de GIT.

Le protocole AMF

AMF signifie *Action Message Format*, il s'agît d'un protocole d'échange de données basé sur SOAP (*Simple Object Access Protocol*) introduit il y a maintenant quelques années, lors de la sortie du Flash Player 6. Le principal avantage contrairement aux solutions *textuelles* (XML, JSON, etc.) est que ce n'est plus du texte qui va transiter entre le serveur et le client, mais des instances d'objets, rendant la réponse directement utilisable. Mieux encore, il est possible de mettre en place un *class Mapping*, c'est-à-dire d'associer une classe PHP à une classe Actionscript et ainsi s'assurer que les deux technologies *parlent la même langue*.

Attention, les alternatives XML ou JSON restent tout à fait viable (même si l'utilisation de données au format JSON n'est pas supportée nativement par Flash) mais peuvent être un vrai gouffre dans les ressources consommées et le temps de traitement (conversion des nœuds en instance de classes du modèle). Ici, le temps de traitement n'est plus puisque, comme déclaré plus haut, la réponse du serveur est directement utilisable, dû au fait que les données circulent dans un format binaire.

Mise en place de l'environnement de travail

Pour commencer, la structure générale du projet sera aussi simple que le projet en lui même. Ce projet, baptisé AIA *(AMF Is Awesome*), sera la couche métier d'une application permettant de bloguer via une application Flex.

```
$ mkdir -p lib/{vendor,AIA/{model,service}}
web
$ git init
$ git add .
$ git commit -m "Created default project
structure"
```

Etant donné que GIT est utilisé, l'ensemble des commandes requises commencent par *git*. La commande *init* permet d'initialiser le dépôt GIT, *add* va ajouter récursivement l'ensemble des fichiers du dossier (d'où le git add .) et enfin *commit* va permettre d'enregistrer les changements sur le dépôt. Le versioning de fichier via GIT passe donc par 2, et éventuellement 3 stades, si l'on prend en compte le *push*, c'est à dire l'envoi des modifications sur les autres **dépôts** distants. Une petite précision avant de continuer avec GIT et PHP : lors d'un ajout, suivi d'un commit, GIT ne prends pas en compte les dossiers vides.

Seulement 3 fichiers ont été créés : le fichier *Post.php*, qui sera le modèle représentant un élément posté sur le blog, le fichier *Blog.php*, qui sera le service et regroupera les actions disponibles demandées depuis Flex, et enfin le fichier *gateway.php* qui sera le fichier appelé par Flex et qui traitera les requêtes.

Pour que l'application fonctionne et soit un exemple pertinent, la tâche suivante va être de créer une base de données. Parce que c'est le plus simple, il s'agira d'une base de données MySQL tout ce qu'il y a de plus basique.

```
$ mysqladmin -uchuck -pnorris create aia
$ mysql -uchuck -pnorris aia
mysql> create table post(
    -> id integer not null primary key auto_
increment,
    -> title varchar(100) not null,
    -> content blob not null,
    -> created_at timestamp not null default
CURRENT_TIMESTAMP);
```

La table contiendra donc des posts avec un titre (*title),* un contenu (*content)* et une date de création (*created_at*), rien de très complexe, donc. Vient ensuite, la création de la classe `Post`, qui sera le *value-object* utilisée pour représenter une ligne de la table et *Blog* qui sera le service. Ces fichiers ayant été modifiés, il est nécessaire de faire en sorte que GIT les prenne en compte :

```
$ touch lib/AIA/model/Post.php lib/AIA/
service/Blog.php
$ git add .
$ git commit -m "Created model and service
classes"
```

L'aspect *versionning* des classes PHP ne sera pas abordé à partir de maintenant, il s'agissait simplement de présenter la manière de travailler avec GIT et l'importance de faire des *commits* réguliers.

Ajout des méthodes de la classe de service : Blog.php

La classe de service `Blog` **comprendra** 2 méthodes : une méthode `save` utilisée pour sauvegarder l'objet en base (ce qui comprend la création ou la modification) et une méthode `find` permettant de récupérer la liste des éléments postés (cf. **Listing 1**).

Mise en place de l'application Flex

La communauté Flex a été très active ces dernières semaines puisque Adobe a sorti Flex 4 *http://opensource.adobe.com/wiki/display/flexsdk/Flex+4* et son environnement de développement nommé Flash Builder *http://www.adobe.com/products/flashbuilder/*. Ce logiciel — faisant partie de la suite CS5, également disponible depuis quelques semaines — reste la solution de prédilection pour le développement d'application Flex. Flash Builder sera utilisé dans le cadre de ce tutoriel, ainsi que GIT pour le suivi de version.

Le problème avec Flash Builder est qu'étant basé sur Eclipse, il génère dans le dossier du projet plusieurs fichiers/dossiers qui **ne doivent pas** être versionnés. Pour cette raison, ces fichiers doivent être ignorés, en les déclarants dans un fichier spécifique nommé *.gitignore*, situé à la racine du projet. Paradoxalement, ce fichier **doit** être versionné.

```
bin-debug
.actionScriptProperties
.flexProperties
.project
.settings
.DS_Store
```

L'application Flex comprendra une classe `Facade`, regroupant les éléments graphiques de l'interface (qui seront détaillés juste après), une classe `Blog` qui s'interfacera avec la classe `Blog` de notre couche métier et enfin une classe `Post` qui est l'alias de la classe `Post` déclarée côté serveur (**cf. Listing 2**).

L'une des améliorations les plus importantes apportées par Flex 4 est la nouvelle architecture des composants. En effet, il est désormais possible de séparer le comportement d'un composant graphique de son apparence (en anglais, *skin*).

L'application est très simple, elle va contenir deux composants principaux. Le premier sera la ***façade*** de l'application, elle comprendra un *List* et un bouton qui permettra d'afficher le formulaire d'ajout d'un post. Le deuxième composant sera ledit formulaire d'ajout/édition d'un post, il comprendra un *TextInput*, permettant de saisir le titre du post, un *TextArea* permettant d'en saisir le contenu, un bouton permettant d'envoyer le formulaire et **éventuellement** un bouton pour annuler et fermer le formulaire.

Pourquoi *éventuellement* ? La séparation comportement/aspect graphique des composants permet, via un metatag *SkinPart* de déclarer quels sont les éléments graphiques qu'un skin doit posséder (un bouton, un champ de texte, une image, etc.). Ce metatag prend un attribut *required* qui permet, comme son nom l'indique, de spécifier si ce composant est obligatoire, ou s'il est facultatif. Dans le cas de cet article, le bouton *cancel* est facultatif, pour l'exemple.

Le **Listing 3** présente les classes `Facade` et `PostForm`, cependant les classes de `skin` ne sont pas contenues dans cet article, faute de place. L'ensemble du projet Flex est disponible sur *Github* à l'adresse suivante : *http://github.com/Palleas/AIA-desktop*.

Intégration du Zend Framework et mise en place de la passerelle AMF

Pour cet article, nous utiliserons la version 2 du Zend Framework, aujourd'hui toujours en développement. La raison à cela est que cette version *casse* la compatibilité avec la branche 1 en utilisant notamment les espaces de nom. De plus, le Zend Framework est désormais accessible via un dépôt GIT. Pour l'intégrer dans ce tutoriel, il va s'agir de passer par une fonctionnalité de GIT : les *submodules*.

Très similaire aux dépôts déclarés externes (propriété SVN *externals*) de *Subversion*, les submodules vont permettre de lier un dépôt GIT à un autre :

```
$ git submodule add git://git.zendframework.
com/zf.git lib/vendor/zf
$ git submodule update
```

Là encore, plus d'informations sur les **submodules** sont disponibles dans l'excellent livre de Scott Chacon.

Pour les personnes familières du Zend Framework, le contenu du fichier *gateway.php* n'est pas très compliqué à appréhender. En effet, les 10 premières lignes ne servent qu'à modifier le chemin d'inclusion de php (en anglais, *include path*) pour lui faire prendre en compte le Zend Framework et à inclure la classe de modèle `Post` ainsi que la classe de service `Blog`. Les lignes suivantes sont cependant plus intéressantes puisqu'elles font intervenir le mot clé `use` de PHP 5.3. L'intérêt de ce mot clé est de pouvoir simplifier le nom des classes que nous allons instancier. De cette manière, plutôt que de faire un `new \Zend\AMF\Server();`, il sera plus simple de faire `new Server()`.

Afin, de ne pas trop perdre de temps avec les différentes dépendances entre les composants du Zend Framework, il est de bon ton de laisser faire le composant *Autoloader*.

Les lignes suivantes permettent de mettre en place le serveur AMF qui traitera les demandes envoyées depuis l'application Flex. La méthode `setClass` permet de dire au serveur de prendre en compte la classe de service `Blog`, la méthode `setClassMap` permet d'associer (en anglais, *map*) la classe Actionscript `com.palleas.AIA.model.Post` à la classe PHP `AIA\Post`.

Le **Listing 4** présente le contenu dudit fichier. De la même manière que pour le projet Flex, les sources PHP de cet exemple sont disponibles sur Github, à l'adresse suivante : *http://github.com/Palleas/AIA-backend*.

Communication entre Flex et PHP

Une fois les composants Flex et la passerelle mis en place, il ne reste pas beaucoup de choses à développer pour établir une première communication entre Flex et PHP. Le **Listing 5** présente la manière d'appeler une méthode (via la méthode `getOperation`) d'une classe de service précise (via la propriété source) via une instance de la classe `RemoteObject`.

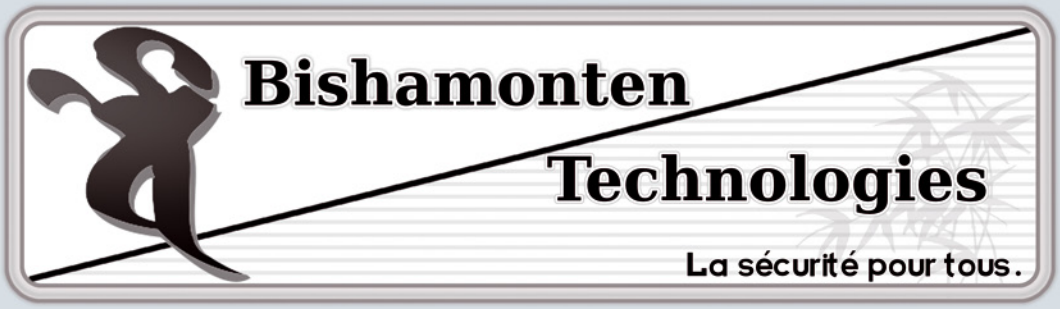
Conclusion

Cet article a montré la simplicité de mettre en place une passerelle pour faire communiquer une application Flex avec une couche métier développée en PHP. Ce protocole ouvre de nombreuses possibilités, il serait en effet également possible d'uploader un fichier directement via le protocole AMF, mettre en place un système de session, etc.

Pour plus d'information sur le protocole AMF, la page wikipedia dédiée (*http://fr.wikipedia.org/wiki/Action_Message_Format*) propose quelques liens, notamment vers la spécification du protocole. La page officielle du ZendFramework v2, est également une très bonne source d'informations (*http://git.zendframework.com/?a=summary&p=zf*). Enfin, la page Flex sur le site d'Adobe (*http://www.adobe.com/products/flex/*) propose des liens vers des exemples, des tutoriels (dont des tutoriels vidéos) et la page du projet Flash Builder, EDI dédié au développement d'applications Flash/Flex.

ROMAIN POUCLET

L'auteur est développeur multimédia travaillant sur Lyon, majoritairement avec les technologies Flex et PHP.



CRÉATION - MODIFICATION -SÉCURISATION D'APPLICATIONS WEB

+ PHP, MySQL, PgSQL
+ Sous-traitance
+ E-mailling
+ Visio-conférence
+ Modification de solutions Open Source
+ Suivi et Gestion de solutions Open Source

Bishamonten Technologies

23 rue du 19 Mars 1962
58 660 Coulanges-les-Nevers

Tél. : 06 30 57 82 85 - E-mail : contact@bsmt.fr
Site internet : www.bsmt.fr

Acquérir de la vitesse avec Zend Framework

Le Zend Framework existe maintenant depuis plusieurs années. La dernière version (1.10) apporte déjà tout ce que l'on peut souhaiter avoir comme éléments pour accélérer la production de site web dynamique. Les classes et outils proposés permettent de couvrir nos besoins les plus exotiques, depuis la génération du projet jusqu'à la gestion de barre code en passant par les flux RSS.

Cet article explique :

- Comment choisir et utiliser les librairies PHP du framework Zend.
- Les règles de programmation à respecter.
- Les pièges à éviter pour avoir un développement propre et sécurisé.

Ce qu'il faut savoir :

- Rien, à part connaître PHP et je suppose que vous avez lu les autres articles sur Zend parus dans PHP Solutions.
- Connaître le pattern MVC ou tout au moins les termes et principes sous-jacent.

Je me propose, par cette série d'articles de passer en revue les classes les plus importantes pour faire un site évolué. Par conséquent, nous parlerons création d'un projet ZF, organisation des librairies et *Autoload* de classe, Interfaçage avec le Javascript et bien entendu AJAX. De quoi démarrer un projet en attendant le ou les prochains articles où j'aimerai parler de *debug* sous Eclipse, de modèle de données, de la création de vues efficaces et donc de *helpers* de vue, de cache, de validation des données issues de l'extérieur, des formulaires, des tests et de bien d'autres choses. J'essayerai aussi de vous donner des éléments de recherche dans la documentation Zend car depuis l'évolution de celle-ci, la recherche en ligne ne fonctionnant plus, il n'est pas toujours aisé de s'y retrouver.

Je ne reviendrai pas sur les sujets déjà abordés dans PHP Solutions autour du Framework Zend (que je nommerai ZF dans le reste de l'article) sauf si la version 1.10 apporte des éléments nouveaux ou si la lisibilité de l'article l'impose. Dans la mesure où je les connais, je citerai les articles et n° de PHP Solutions qui traitent du sujet.

Mais avant d'aller plus loin, j'aimerais vous faire part d'une remarque personnelle. Si ZF rencontre un si vif succès c'est que ce n'est pas vraiment un framework comme Cake PHP ou Symfony. C'est plus une librairie de classes ayant une certaine homogénéité et des câblages prés définis. Ce qui apporte de la souplesse, notamment lors des migrations. J'ai ainsi vécu en ZF 1.7 le passage, par étapes successives, d'un framework MVC qui n'était plus soutenu vers ZF. Les développements d'un Framework (accès à la Base de données par exemple) cohabitant très bien avec les développements de l'autre.

Autre avantage de ZF : c'est son coût d'entrée. C'est-à-dire l'investissement (en terme de temps) nécessaire pour l'utiliser ne serait-ce que correctement. Fan de Symfony de la première heure (celle où ZF n'était encore qu'un embryon prometteur), je dois admettre qu'il m'a fallu plus de temps pour entrer dedans que le ZF. Pourquoi me direz-vous, tous simplement parce que l'on s'intéresse aux choses au fur et à mesure, graduellement, jours après jours. En conséquence, je ne l'utilise pas encore à 100%. Mais, au final cela ne me gène pas, et surtout, le plus important, je reste dans une dynamique de diminution du temps de développement.

Débutons ensemble

Je parts donc du principe que vous avez lu ou pouvez lire les articles sur comment démarrer avec ZF 1.8 et comment gérer l'authentification. Articles parus dans le numéro spécial PHP STARTER KIT FRAMEWORKS de septembre 2009. Malgré cela, quand

```
1. pear discover-channel pear.zfcampus.org
2. pear install zfcampus/zf
```

Figure 1. *Extrait doc ZEND*

cela sera nécessaire pour la compréhension de l'article, il se peut que je répète des éléments déjà exprimés dans ces articles.

Je considère aussi comme acquis, le fait que vous ayez sur votre machine un serveur WEB (Apache ou autre mais plutôt Apache) avec un PHP raccordé. Vous devez certainement avoir aussi une base de données. Quelle que soit celle que vous avez choisie, les points abordés, ci-après, resteront vrais. Surtout que je ne parle pas de base de donnée dans cet article !

Comment débuter ?

Par expérience, ce n'est pas l'installation initiale du Framework qui pose problème. De nombreux tutoriels (dont ceux de la *doc* Zend) permettent de le faire très facilement. Par contre on me demande à chaque fois : comment est-ce que j'organise mon application, où est-ce que je mets mes déclarations, mes initialisations mes développements ? Comment je l'utilise avec Eclipse ? Autant de questions sur l'architecture globale. Je vais me concentrer sur ces aspects, rarement abordés.

Avec ZF 1.10.X est arrivée la librairie *Zend_Tool*. Elle apporte une couche homogénéisant les *Zend_Tool_Framework* et *Zend_Tool_project* déjà existant en 1.8. Même si je ne suis pas fan des outils d'automatisation, ils permettent de rapidement créer une arborescence de projet ce qui est extrêmement pratique. Surtout pour des tâches répétitives comme la création de modules, de contrôleurs et d'actions (termes issus de la découpe MVC du Framework, sujet déjà abordé dans d'autres articles de PHP Solutions).

Cette librairie vous permettra, via *Zend_Tool_Framework*, de créer des outils maison et donc, pourquoi pas, de ne déployer que les librairies que vous souhaitez (par exemple vous ne souhaitez pas mettre les librairies liées service WEB ou restreindre l'usage de *Helper* Javascript à Dojo, ... ou encore ajouter vos librairies spécifiques). Utile quand vous aurez à définir votre forge logiciel PHP. J'y reviendrai dans un autre article.

La première étape consiste à télécharger ces outils. Personnellement, j'utilise déjà d'autres librairies PEAR et donc je préfère utiliser le *loader* PEAR pour récupérer ces outils. Ainsi je suis certain que les chemins d'accès sont déclarés dans mon système. Et que l'outil en ligne de commande zf est toujours accessible. Mais il est possible de le mettre ailleurs, l'essentiel étant qu'en tapant zf dans une fenêtre console l'accès au fichier *zf.bat* ou *zf.sh* soit résolu par votre système d'exploitation.

L'image de la Figure 1, issue de la doc ZEND, décrit comment charger, non seulement, *Zend_Tool* mais tout le framework Zend via PEAR. Même s'il y a une journée de décalage dans la publication des versions de ZF

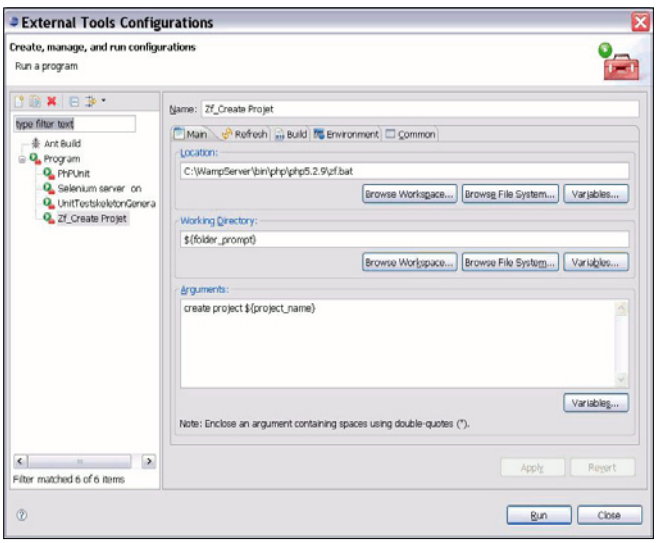


Figure 2. *Création d'une commande ZF*

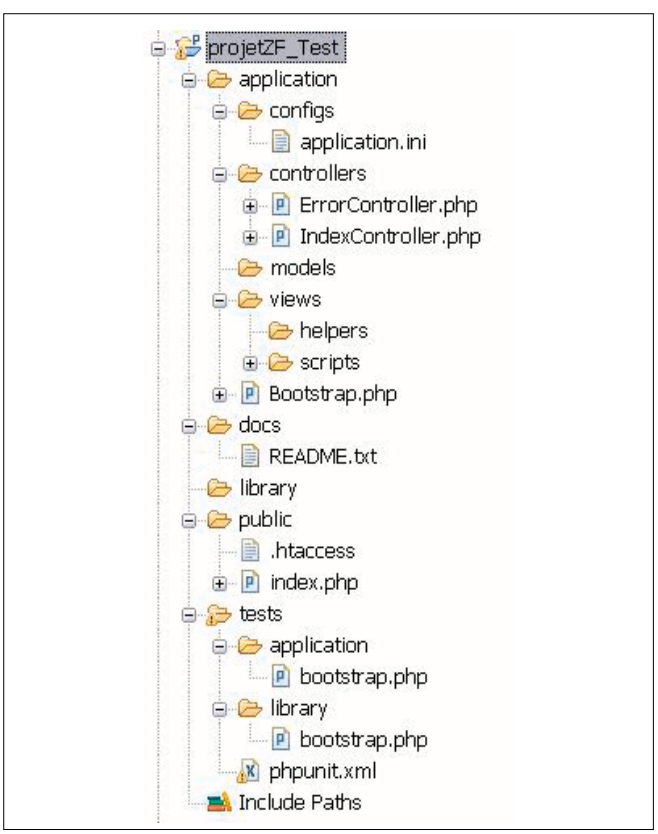


Figure 3. *Arborescence d'un projet créé par ZF*

sur *zfcampus.org*, vous serez certain que votre librairie Zend Framework sera accessible dans le *Include_path PHP*. Pratique non ?

Cette méthode ne fait pas l'unanimité. L'un de mes collègues ne l'aime pas du tout car il n'est pas toujours possible de l'utiliser sur les serveurs de production. Surtout s'ils sont mutualisés ! Il préfère copier la librairie *Zend* dans le répertoire *library* de l'arborescence par défaut. C'est une excellente remarque et au final, je fais souvent les deux. Sans que cela ne pose de problème. Par contre, attention si vous utilisez plusieurs versions de PHP sur votre machine ... On peut vite

Listing 1. *Fichier application.ini fourni par défaut*

```
[production]
phpSettings.display_startup_errors = 0
phpSettings.display_errors = 0
includePaths.library = APPLICATION_PATH "/../library"
bootstrap.path = APPLICATION_PATH "/Bootstrap.php"
bootstrap.class = "Bootstrap"
appnamespace = "Application"
resources.frontController.controllerDirectory =
APPLICATION_PATH "/controllers"
resources.frontController.params.displayExceptions = 0

[staging : production]

[testing : production]
phpSettings.display_startup_errors = 1
phpSettings.display_errors = 1

[development : production]
phpSettings.display_startup_errors = 1
phpSettings.display_errors = 1
resources.frontController.params.displayExceptions = 1
```

se perdre et ne plus savoir ce que l'on utilise comme librairie *Zend*. Entre les chemins d'include de votre OS, de PHP et l'*autoload* de Zend (j'y reviendrai) il faut être clair et net.

Premier point, avec une version 1.7.2 comme avec une version 1.9.0 de PEAR c'est channel-discover qui existe ... et oui la doc Zend peut contenir des erreurs. Alors en plus de la doc, lisez toujours les commentaires de la communauté, ils sont tout aussi important.

Nous sommes sur le point de nous lancer dans l'inconnu. Mais, si comme moi vous utilisez *Eclipse PDT all in one*, vous pourriez ne pas vouloir sortir de Eclipse pour lancer vos commandes ZF. Il est très facile de réaliser un outil externe qui permettra de tout créer sans sortir d'Eclipse. La Figure 2 donne une vue de la commande de création d'une architecture projet. Si vous utilisez une fenêtre en ligne de commande, il faudra taper zf create project *NomDeMonProjet*. C'est ce que nous allons essayer de recréer avec l'outil d'application externe de Eclipse. La variable `${folder_prompt}` indique que la commande demandera de définir le répertoire où se mettra le projet. Répertoire qui est en général le chemin du *workspace* Eclipse (si vous le souhaitez vous pouvez utiliser `workspace_loc` à la place). Pour les autres fonctions, vous aurez certainement besoin de `string_prompt` pour saisir les noms des modules, des contrôleurs ou des actions. Vous aurez besoin de `ressource_name` pour identifier les modules ou contrôleurs où créer des contrôleurs ou des actions. N'hésitez pas à cliquer sur le boutons variable pour voir la liste des variables disponibles et leur description. Faire des outils n'est pas aussi compliqué qu'il y parait et c'est surtout sans risque.

Le résultat de la fonction de la Figure 2 ou de la ligne de commande est que vous avez maintenant une arborescence projet complète identique à celle donnée dans la Figure 3. Plus besoin de se faire des nœuds au cerveau, tout est là. La configuration de votre serveur WEB devra définir que la racine de votre site pointe sur le répertoire */public*. Vous y trouverez l'*index.php* qui sera le point d'entrée unique de votre application et le .htaccess (les utilisateurs de Windows seront heureux de le voir ...) qui limitera les accès et forcera ce point d'entrée unique. De mon expérience, le fichier *index.php* n'est pas à modifier.

Une seule exception concerne le lancement de l'objet application. Ce lancement se fait en donnant deux paramètres : l'environnement d'exécution (voir le prochain paragraphe) et la configuration de l'application. Par défaut, cette dernière est réalisée à l'aide du fichier application.ini mais si vous préférez le xml il suffit de passer un fichier application.xml voire un objet de type `Zend_config` ou un array. C'est à vous de choisir. L'objet application s'adaptera et lancera le *bootstrap* que l'on trouvera dans le fichier *bootstrap.php* sous le répertoire /application. Pour plus d'information sur l'objet application, reportez-vous à la Figure 4 qui donne le diagramme de classe autour de cet objet.

Ce fichier est le premier que vous aurez à modifier car c'est là que vous aurez à enregistrer vos actions d'initialisation de vos ressources ... Nous y reviendrons. Avant d'aller plus loin dans l'usage du Framework j'aimerai faire quelques points d'architecture et d'usage.

Le choix de l'environnement de travail

Le *.htaccess* peut être utilisé pour définir l'environnement dans lequel travailler. Via le fichier de configuration *application.ini*, que vous trouverez sous le répertoire */application/configs/*, il est possible de donner des configurations différentes selon que l'on soit en mode développement, test ou production. Le Listing 1 donne le contenu par défaut de ce fichier. Les différents environnements héritent des définitions faites dans production. Pour vos besoins, elles peuvent être écrasées par une nouvelle déclaration. Le tout est de définir cet environnement. Ma première réaction a été de modifier l'index .php pour définir la variable d'environnement : ce n'est pas la bonne manière. Il est préférable de jouer sur *.htaccess*. Car en général, ces configurations correspondent à des environnements physiques différents.

Le simple ajout d'une ligne dans le *.htaccess* avec la fonction `Apache SetEnv` permet de définir des variables d'environnement. Pour ce qui nous intéresse, nous avons besoin de définir la variable `APPLICATION_ENV` (définie dans *index.php*) et lui donner une des valeurs définies dans le fichier de configuration. Par défaut sont définis [production] (qui est la valeur par défaut si la variable n'est pas définie), `[staging]` (correspond à la phase de *preprod* de votre application), `[testing]` et `[development]`. Pour la phase de développement la ligne à ajouter dans *.htaccess* est donc :

```
SetEnv APPLICATION_ENV development
```

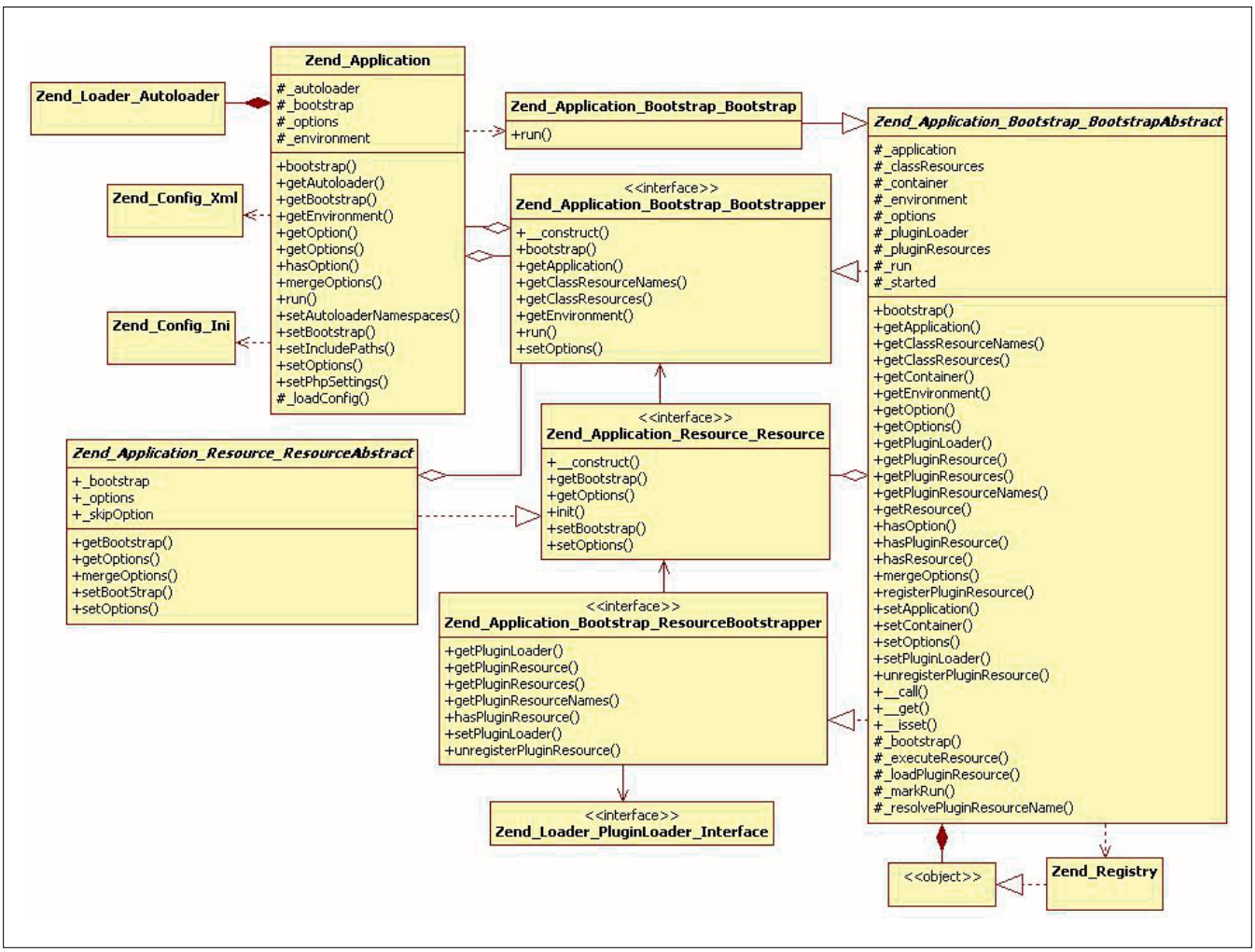


Figure 4. *Diagramme de classe de l'objet application*

Cela permet, tout au moins, d'avoir plus d'informations sur les erreurs en phase de développement et d'utiliser une base de données locale et bien d'autres choses encore. Là encore j'y reviendrai. Remarquez que development hérite de toutes les valeurs de production via `[development : production]`.

La structure de l'application

Il peut être intéressant d'avoir une application séparée en plusieurs parties autonomes mais utilisant les mêmes objets métier voire la même base. Par exemple avoir une application de *Front Office* et une de *Back Office* (un site WEB et son interface d'administration). Zend Framework nous propose la possibilité de créer des modules pour répondre à ce problème. Ceci n'aura d'impact que sur la partie Application de notre dossier projet le reste étant partagé par les différents modules. Pratique non ?

La création de module peut se faire manuellement mais zf_tool facilite grandement les choses. Une commande permet de créer un module : `ZF create module NomDeMonModule`. Attention à se trouver sous le répertoire du projet pour lancer la commande. Là encore nous pouvons créer un raccourci sous Eclipse pour faciliter la

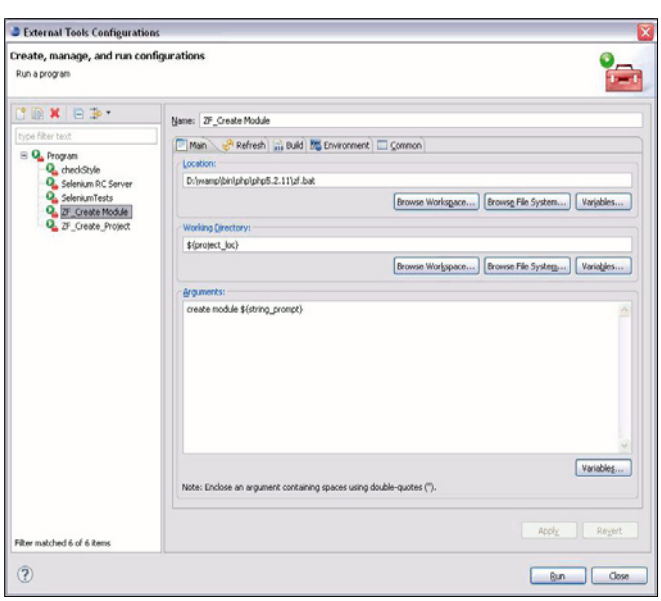


Figure 5. *Outil externe pour créer un module sous Eclipse*

création. La Figure 5 donne l'outil à créer et la Figure 6 décrit l'arborescence obtenue.

Vous remarquerez que le module s'est ajouté dans le répertoire application en créant ses sous-répertoires MVC propres. Ainsi avec www.monnomdedomaine.com/

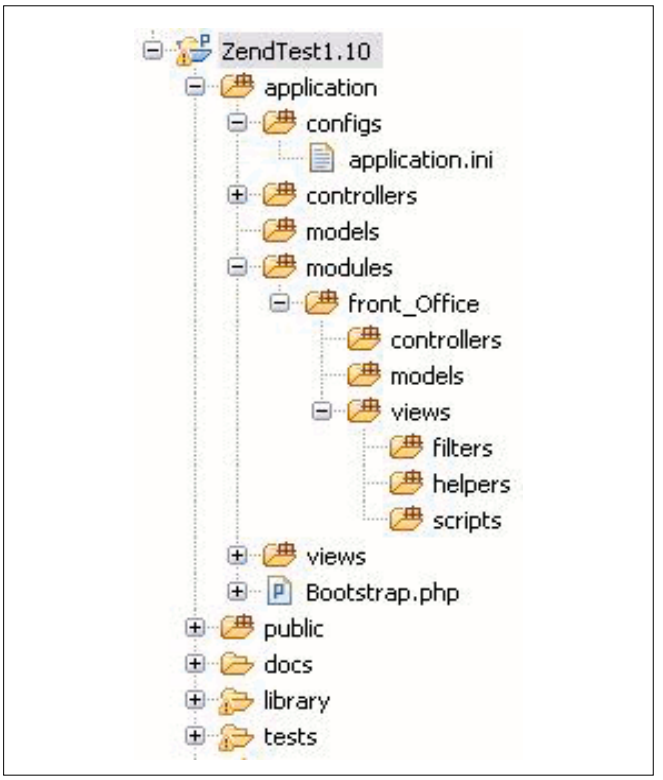


Figure 6. *Arborescence projet résultant de la création du module front_office*

Listing 2. *Commentaire à éviter*

```
<?php

class Bootstrap extends Zend_Application_Bootstrap_
Bootstrap

{
  /**
  * fonction d'init de l'autoload, elle sera chargée
au lancement de la
  * ressource autoload
  * @return Zend_Loader_Autoloader
  **/

  function _initAutoloader() {
    $autoloader = Zend_Loader_Autoloader::
getInstance();
    $autoloader->registerNamespace('Proj_');
    return $autoloader;
  }
}
```

J'attire votre attention sur la zone de commentaire placée avant la fonction : c'est plus qu'une bonne pratique car sous eclipse ces données sont utilisées dans l'aide en ligne. De plus votre génération de doc sera grandement facilitée si vous utilisez les tag *phpdoc* comme *@return*.

j'accéderai au contrôleur index par défaut (celui dans */application/controllers/*) et avec *www.monnomdedomaine.com/front_office/* j'accéderai à un contrôleur index placé sous */application/modules/front_office/controllers/*. Le *bootstrap.php* sera commun à tous les modules et vous pourrez créer des bootsraps complémentaires pour chaque module …J'y reviendrai dans un autre article car l'usage des modules n'est pas aussi compliqué qu'il n'y paraît au premier abord et est extrêmement pratique pour gérer des environnements de vue différents. Par exemple si vous devez réaliser une application avec des accès web et mobiles par exemple.

Où ranger mes développements

Comme avec tout Framework, ma première question a été : bon c'est joli tout cela mais je range mes développements où ? Il y a plusieurs endroits où nous rangerons les fruits de notre labeur.

1. Les contrôleurs et les actions qui répondront aux URL *www.monAdresse/controleur/action* seront rangés dans le répertoire *\application\controllers* ou *\application\modules\NomDeMonModule\controllers*. A titre d'exemple vous pourrez regarder les deux fichiers créés par *zf_tool : IndexController.php* et *ErrorController.php*. Ils gèrent, respectivement, l'accès par défaut au site et les cas d'erreur.
2. Les vues (c'est-à-dire les fichiers qui composeront vos pages HTML) seront rangées dans *\application\views\scripts* et dans des répertoires ayant le même nom que le contrôleur *index* pour *IndexController.php*. Le *Layout (*c'est-à-dire la partie commune à toutes vos pages) est rangé dans ce répertoire directement sous la racine. Comme toujours, pour les modules il faut aller dans *\application\modules\NomDeMonModule\ views\scripts*.
3. Les classes attachées à votre modèle de données, plus précisément aux tables de votre base de données sont à ranger dans \application\models. Là, pas de différence d'un module à l'autre. C'est un premier intérêt.
4. Vos objets métiers (en bref tout ceux que vous aller créer pour vos besoins propres) sont à ranger dans un répertoire dans *\library*. Le nom de ce répertoire sera utilisé comme préfixe de vos noms de classe. Je vais vous expliquer dans quelques lignes pourquoi.
5. Les fichiers CSS seront à ranger dans *\public\css*, les fichiers images dans *\public\images* et les fichiers javascript dans *\public\js*. Sur ce dernier sujet, il peut être pratique d'isoler dans *\public\js\ext* les librairies javascript externes comme Jquery. Si vous voulez utiliser Dojo avec les *helper* Dojo, vous devrez mettre Dojo (dijit, dojo et dojoX) tout au même niveau. Si ce sujet vous intéresse, je pourrai y revenir dans un prochain article.

Ces modes de rangements imposés par le framework nous permettrons d'automatiser un maximum de chose par la suite.

La fin des require et des include

Qui n'a pas galéré avec des includes ou des require manquants ? Tous, nous sommes passés une fois par là. Avec `__Autoload()` puis la SPL de PHP 5.2 est apparue la possibilité d'*autoloader* les classes au fur et à mesure qu'elle sont utilisées. Le Zend Framework a poussé cela bien plus loin. Il a créé un singleton *Autoload* pour charger automatiquement les fichiers Zend, quel que soit leur position. Ou presque, puisque *Autoload* va rechercher, par défaut, dans l'`include_path` PHP et dans le répertoire *library.* Ce dernier est très intéressant car c'est là que nous avons mis nos objets métier, souvenez-vous !

Supposons que vous ayez mis vos objets métier dans un répertoire dénommé *Proj*. Alors en préfixant vos classes avec `Proj_`, il est très simple de définir de l'*autoload*. Et cela peut se faire dans le *bootstrap* en définissant une fonction `_initAutoloader()` comme définie dans le code du Listing 2.

Notez la présence du sous tiret dans le nom du *Namespace.* Il n'est pas (ou plus) utilisé par défaut comme délimiteur de nom (ce l'était jusqu'en 1.7). Il existe une autre façon de faire cela. Vous pouvez utiliser la configuration de votre application en définissant cela dans le fichier *application.ini*. Pour cela nous allons ajouter, dans la rubrique production, cette déclaration. Pourquoi production vous demandez-vous ? Tout d'abord car tout doit être déclaré dans production pour que votre application fonctionne et si vous avez des choses spécifiques pour un environnement donné alors vous redéfinirez la valeur dans cet environnement. Le Listing 3 donne le code de la rubrique production. Ensuite, dans ce cas, l'init est valable dans tous les environnements.

A partir de maintenant, vous n'aurez plus à vous souciez d'INCLUDE ou de REQUIRE sauf pour les fichiers de constantes globales … Personnellement je les mets dans les fichiers de *config*. Pour les chaînes de caractères, mieux vaut utiliser le système d'internationalisation pour les stocker ! Avant de passer à autre chose, sachez que les fonctions dont le nom débutera par `_init` seront automatiquement lancées au démarrage de l'application et permettront d'initialiser les ressources de votre application. Voilà qui est fort intéressant. Par exemple, pour initialiser vos log, vous créerez une fonction `_initLog()`. Fonction que vous pourrez conjuguer avec une déclaration dans votre *application.ini* du type *log.file ="/logs/logErrors.log*". Données que vous récupérerez facilement dans votre fonction `_initLog()` par `$file = $this->getOption(«file»)`. La notation dans application.ini est la suivante : ressource.option = valeur. Dans notre cas, la ressource est *Zend_Log.*

Si vous désirez mettre des commentaires dans votre *application.ini*, il faut mettre un caractère diese en tête de ligne.

Listing 3. *Configuration de l'autoload*

```
[production]
phpSettings.display_startup_errors = 0
phpSettings.display_errors = 0
includePaths.library = APPLICATION_PATH "/../library"
bootstrap.path = APPLICATION_PATH "/Bootstrap.php"
bootstrap.class = "Bootstrap"
appnamespace = "Application"
resources.frontController.controllerDirectory =
APPLICATION_PATH "/controllers"
resources.frontController.params.displayExceptions = 0
autoloadernamespaces[] = "Proj_"
includepaths[] =  APPLICATION_PATH "/proj"
```

Listing 4. *Déclaration Ajax*

```
    public function init()

    {
        $ajaxContext = $this->_helper->getHelper('Aja
xContext');
        $ajaxContext->addActionContext('nomAction',
'html')
                    ->initContext();
    }
```

Listing 5. *htmllayout.php*

```
<?php

        echo $this->ajaxResponse;

?>
```

Vive le WEB 2.0 avec ZF

Le Zend Framework propose de nombreuses facilités pour faire du Web 2.0. Quel que soit le moyen par lequel vous appelez l'application vous aurez à répondre en JSON, l'XML ou en HTML.•• Personnellement, je n'utilise que le JSON ou le HTML mais voyons ce que le framework nous propose pour cela.

Utiliser le JSON

Pour faire du JSON, Zend à réaliser la classe `Zend_Json`. Grâce à elle, pas besoin de créer un layout spécifique, la classe Zend_Json s'en charge pour vous.

Pour renvoyer des données en Json, il suffit d'utiliser :

```
$this->view->data = zend_json::encode($data->
toArray());
```

Facile non ? Rien de plus n'est nécessaire ! Cela retournera un tableau en JavaScript. Attention à ne pas utiliser la fonction `eval()` en Javascript mais plutôt traiter les données du tableau pour créer vos objets applicatifs. La fonction `eval()` est une faille de sécurité majeure. C'est pour cette raison que dans jQuery il existe la fonction `jQuery.getJSON()` qui utilise un parseur de

Sur Internet

- Eclipse PDT : *http://www.eclipse.org/pdt/downloads/,*
- Librairie PEAR : *http://pear.php.net/,*
- Téléchargement du framework : http://framework.zend.com/download/latest,
- Documentation Zend Framework : *http://framework.zend.com/manual/fr/manual.html,*
- Forum Français autour de Zend Framework : *http://www.z-f.fr/,*
- Liste des tags phpDoc : *http://manual.phpdoc.org/HTMLSmartyConverter/HandS/phpDocumentor/tutorial_tags.pkg.html.*

données pour intégrer les données de la réponse. C'est un peut plus lent mais plus sûr.

Pour aller un peut plus loin il existe une petite fonction pratique `Zend_Json::prettyPrint()` qui permet d'afficher le JSON généré de façon lisible. Cela évite de regarder la réponse sous Firebug. Il est possible de passer un objet en paramètre de la fonction `encode` mais le résultat restera un tableau javascript et seuls les paramètres publics seront accessibles à la fonction `Zend_Json::encode()`. Si vous définissez dans votre objet une fonction `toJson()` (attention à la majuscule) alors `Zend_Json` l'utilisera pour générer le javascript.Si vous voulez que la réponse soit un objet Javascript plutôt qu'un tableau il faudra le spécifier avec l'option `Zend_Json::TYPE_OBJECT`.

Utiliser un autre format en Ajax

Il est possible de définir des modes de retour particuliers. Comme je le disais en introduction j'aime utiliser le HTML pour l'inclure directement dans mes DIV avec la fonction jQuery `$(#DIV).load()`. Là c'est le coté *Client* mais que devons-nous faire côté serveur ? Cela se fait en plusieurs étapes. Dans un premier temps, il faut déclarer dans l'init du contrôleur la spécificité de retour attendue.Pour cela il faut ajouter dans la classe du contrôleur, avant la déclaration des actions, le code du Listing 4.

A partir de là l'URL qui appellera, via Ajax, sera de la *forme /controlerName/ActionName/format/html* et la réponse utilisera un layout dédié dénommé *htmllayout.php* ! Il sera placé sous */application/views/*. Il utilisera comme de normal la vue *nomAction* rangée, quant à elle, sous */application/views/controlerName/*. Le contenu du fichier *htmllayout.php* est aussi simple que le contenu du Listing 5.

Simple aussi non ? Bon, oui pas tout à fait aussi simple que le précédent mais extrêmement pratique, vous verrez. J'en profite pour rebondir sur la notion de *Helper.* Dans l'exemple du Listing 4 mais aussi dans le paragraphe sur JSON, j'ai utilisé des *Helpers.* Zend Framework propose deux ensembles de *Helpers.* Des *Helpers d'Action* (que vous trouverez décrits dans la classe Zend_Controller) et des *Helpers* de vue (que vous trouverez décrits dans la classe `Zend_View`). Je les aborderai dans mon prochain article, c'est promis. Il sont extrêmement pratiques surtout les *Helpers* de vue.

Conclusion

Voici les premiers éléments que je souhaitais partager avec vous. Ils ne sont pas suffisants pour utiliser pleinement le framework Zend. Il vous faudra attendre mon (ou mes) prochain(s) article(s) pour aller plus vers l'avant, mais n'hésitez pas à demander les sujets qui vous intéressent au journal, j'en tiendrai compte dans le choix de mes prochains thèmes.

STÉPHANE GUÉDON

L'auteur travaille au sein d'une société de développement d'applications informatiques depuis quinze années. Aujourd'hui, consultant pour une grande société de service d'un groupe international, il anime le projet PHP de la Direction Technique Nationale de cette société et participe aux projets WEB 2.0 de cette même direction. Adepte du PHP depuis ... l'ère des dinosaures, il œuvre à la communication et à la promotion du PHP au sein de cette entreprise.

En complément de ces activités, l'auteur est également Coach Agile et assiste les entreprises dans le déploiement des méthodes de développement Agile.

LIBRA LINUX

www.libra-linux.com

... des formations Linux sur mesure

La route est longue mais

La voie est libre

La virtualisation
sous Linux, technologies et solutions

Pourquoi virtualiser notre architecture et quels intérêts avons-nous à le faire ?

Cet article explique :
- Les grandes principes de la virtualisation .
- L'importance de comprendre les mécanismes et les intérêts de la virtualisation.

Ce qu'il faut savoir :
- Une bonne connaissance en systèmes et des connaissances en réseaux.

Aujourd'hui, la virtualisation présente un grand nombre d'avantages :

- une meilleure utilisation de ressources matérielles,
- une plus grande disponibilité,
- une diminution du TCO[1] pour les entreprises,
- un gain non négligeable pour l'environnement notamment au niveau de la dissipation de chaleur et de la consommation d'énergie.

Au travers de cet article, nous allons vous présenter succinctement les différentes technologies de virtualisation disponibles sous Linux. Pour ce faire, nous procéderons à une brève présentation de la virtualisation et ses concepts, ainsi que les enjeux autour de ce phénomène. Ensuite, nous aborderons les différentes technologies afin d'en exposer les différents aspects, et fournirons un exemple de solution associée à chacune des technologies.

La virtualisation – présentation et enjeux

Dans cette première partie, nous allons d'abord définir la virtualisation et discuter sur ses apports.

Présentation

On définit en informatique la virtualisation comme un ensemble de techniques matérielles et/ou logicielles mises en œuvre pour faire fonctionner plusieurs systèmes d'exploitation sur la même machine.

On comprend bien que le but de ce concept est de faire fonctionner plusieurs machines logiques différentes que l'on nommera par la suite ***machines virtuelles*** sur une même machine physique identifiée par ***système hôte***. Ce concept s'appuie sur différentes technologies que nous aborderons au sein de cet article ; seules les solutions disponibles sous Linux seront traitées.

Dans ce périmètre, trois grandes techniques émergent :

- La virtualisation par isolation, communément appelée cloisonnement.
- La virtualisation totale.
- La para- virtualisation & hyperviseur.

Ces techniques offrent des mises en œuvre et performances totalement différentes. Chacune d'elles doit être déployée et utilisée dans son contexte.

Au travers des exemples de cet article, nous spécifierons le contexte qui nous semble le plus approprié à l'utilisation de chacune d'elles. Au vu des infrastructures informatiques de plus en plus importantes et complexes, on peut s'apercevoir de l'intérêt de déployer des solutions de virtualisation. Mais quels sont les enjeux principaux et les gains apportés par la virtualisation ?

Grâce à la virtualisation, nous avons la possibilité de faire fonctionner différents systèmes définis par la suite comme OS[2] sur le même serveur physique ;

il apparaît donc trois enjeux majeurs (liste non exhaustive) :

- L'enjeu économique.
- L'enjeu technique.
- L'enjeu écologique.

L'enjeu économique

La mise en œuvre de ce type de solution permet effectivement de diminuer le TCO[1] informatique : en mutualisant les services, on peut largement diminuer le nombre de machines physiques. La virtualisation dispose aujourd'hui de mécanismes pour déployer des machines virtuelles très rapidement avec les solutions logicielles proposées ; ces mécanismes sont basés sur des *Templates*. Ceci permet une installation très rapide d'un nouveau serveur. Nous avons un gain de productivité non négligeable. En outre, le fait de diminuer le nombre de machines physiques permet de jouer sur le paramètre de la consommation électrique ainsi que l'encombrement lié à l'informatique.

L'enjeu technique

C'est grâce à la virtualisation que les administrateurs systèmes ont trouvé d'importantes avancées techniques :

- gain de temps dans le déploiement d'un serveur,
- augmentation de la disponibilité de l'application ou de l'architecture informatique,
- des environnements de test beaucoup plus accessibles,
- a possibilité de tester plus facilement de nouvelles distributions ou systèmes.

La virtualisation facilite bien des domaines car, couplée avec d'autres outils comme Linux HA[3], nous avons à notre portée toutes les techniques pour mettre en place des infrastructures plus disponibles et sécurisées (mise en œuvre de clusters et autres technologies). Toutefois il faut faire attention à ne pas atteindre des niveaux de complexité extrême; une gestion et une connaissance rigoureuse de ce type d'infrastructure sont capitales.

L'enjeu écologique

Il nous paraît pertinent d'aborder l'aspect écologique et le gain en matière de consommation électrique réalisé grâce à la virtualisation :

- Moins de besoins en climatisation.
- Des mécanismes de gestion intelligente de l'alimentation commencent à apparaître au sein des solutions de virtualisation, ce qui diminue la consommation électrique globale.

Enfin, moins de matériel détenu sera d'autant moins de matériel à recycler.

Les techniques de virtualisation

Après une brève présentation de la virtualisation et de ses enjeux, nous allons maintenant aborder des aspects plus techniques. Avant tout, voici quelques notions liées à la virtualisation.

Chaque solution de virtualisation s'appuie sur une ou toutes les notions suivantes :

- Une couche abstraction du matériel et/ou logiciel.
- Un système d'exploitation hôte.
- Une solution logicielle.
- Réseau virtuel au sein de la machine hôte (réseau de niveau 2 par la mise en œuvre d'un pont).
- Possibilité de déployer des Templates et réaliser des sauvegardes par Snapshots des machines virtuelles.

Principe de virtualisation

Aujourd'hui, un des intérêts majeurs de déployer ce type de technologies est de maximiser l'utilisation des ressources des machines physiques. Dans la plupart des architectures, les applications sont isolées lorsqu'on les installe sur de nouvelles machines, soit à cause d'une incompatibilité des OS (Linux ou Windows), soit à cause de librairies manquantes mais non compatibles avec des applicatifs déjà existants sur la plateforme. De plus, dans la plupart des cas, les ressources matérielles du serveur ne sont jamais utilisées à 100 %.

La virtualisation répond à tous ces problèmes en fonction des besoins, il suffit d'utiliser les bons outils afin obtenir les meilleures performances possibles. Nous allons traiter dans la suite de l'article les trois catégories que nous avons mises en production dans le cadre de nos fonctions soit :

- Une solution de virtualisation par isolation de contexte : isolateur ou cloisonnement.
- Une solution de virtualisation totale (dans notre cas, nous entendons par virtualisation totale les solutions qui simulent aussi le processeur au niveau de la machine virtuelle).
- Une solution de para-virtualisation & hyperviseur.

Nous allons examiner, au sein des paragraphes suivants, les différentes technologies listées auparavant en expliquant leur fonctionnement et en présentant une solution associée.

Virtualisation par isolation de contexte ou cloisonnement

Cette technologie consiste à enfermer les processus du serveur virtuel dans une cage appelée contexte dont ils ne pourront théoriquement pas sortir.

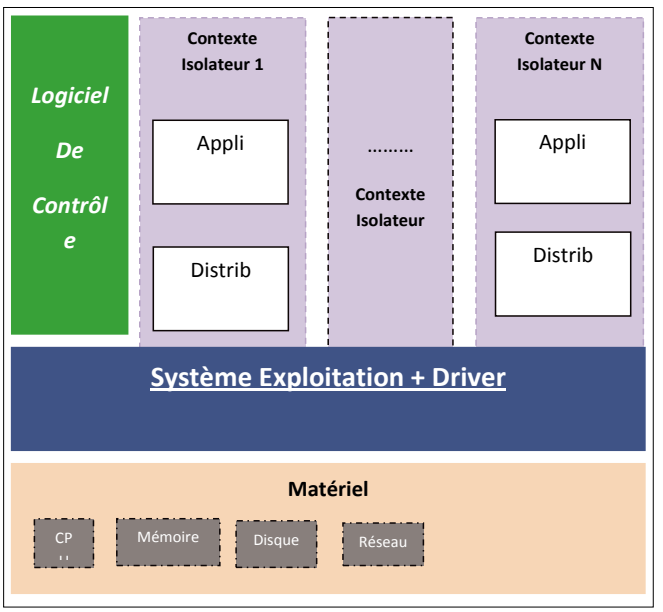


Figure 1. *Virtualisation par cloisonnement*

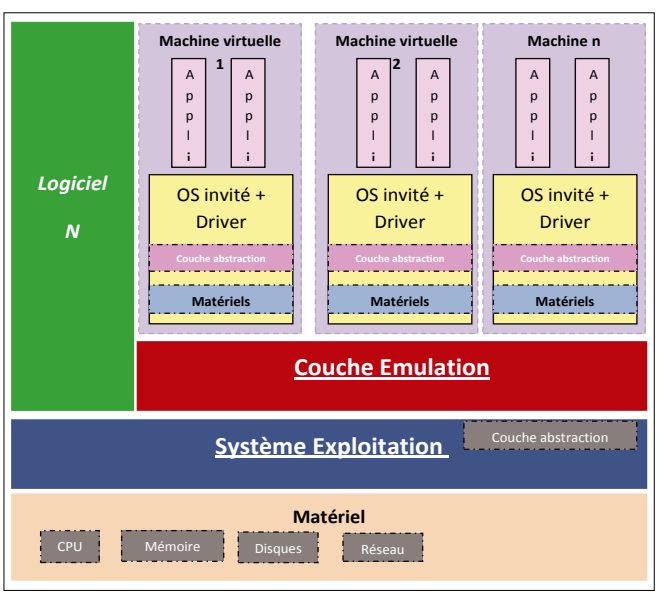


Figure 2. *Virtualisation complète*

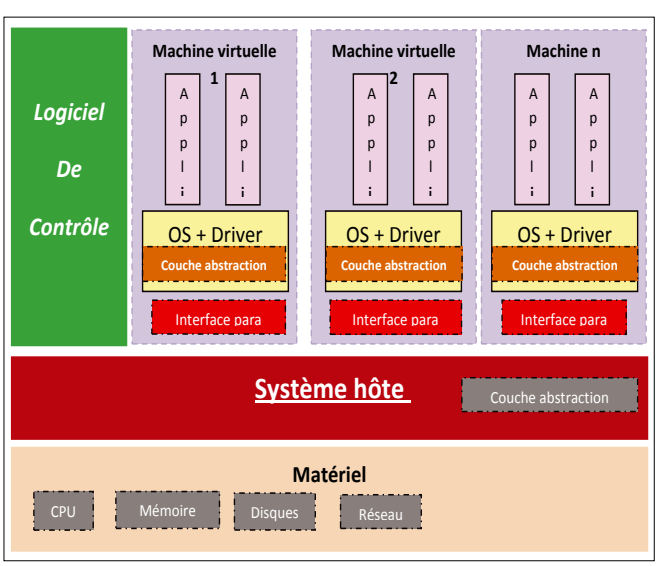


Figure 3. *Para-virtualisation*

Principe de fonctionnement

Ce type de virtualisation présente l'avantage d'obtenir des performances quasi-natives, avec peu de pertes de performances de la machine physique (environ 2 à 4 %). Effectivement, les serveurs virtuels sont identifiés par un numéro unique (n° de contexte) ; grâce à cet identifiant unique, la machine hôte est capable d'identifier chaque process qui fonctionne (voir Figure 1).

On comprend que l'inconvénient majeur de cette solution est la limitation des systèmes pouvant être exécutés : seuls des systèmes Linux pourront être déployés par cette technologie. Cette solution est la plus employée lorsque l'on déploie en majeure partie des systèmes Linux (peu importe la distribution) ou encore lorsqu'on veut faire de l'hébergement Web. Avec cette technologie, il est facile de déployer des plateformes de type LAMP[7].

Solution : OpenVz

Les deux solutions que nous avons mises en production sont les suivantes :

- Linux Vserver (projet open source).
- OpenVz.

A ce jour, seule la solution OpenVz est restée en production. En effet, au moment de son déploiement, seule OpenVz nous permettait une gestion du pare-feu au niveau du serveur virtuel (*NetFilter*) et la possibilité de faire de l'IPV6. OpenVz est la version libre du projet Parallels Virtuozzo. Cette solution est disponible en package Debian (*.Deb*) pour les distributions Debian et Ubuntu ; pour le reste des distributions, si le paquet n'est pas disponible, il est possible de récupérer le patch du noyau pour l'appliquer.

Virtualisation Totale (ou Complète)

Grâce à cette technologie, nous allons pouvoir virtualiser n'importe quel OS non modifié en l'encapsulant dans une machine virtuelle (voir Figure 2).

Principe de fonctionnement

Cette technologie simule au sein de la machine virtuelle une nouvelle couche matérielle (dont le processeur) et aussi une nouvelle couche d'abstraction. Dans ce cas, **la machine virtuelle n'a aucune conscience d'être virtualisée** et donc pense disposer du matériel.

Du point de vue du système d'exploitation, la machine virtuelle se trouve au même niveau de privilège que n'importe quel autre programme en fonctionnement. On comprend bien l'impact sur les performances : nous allons avoir une perte directe et une importante dégradation des performances en fonction du nombre d'applications

démarrées sur la machine. Une perte des performances comprise entre 5 et 40 % est possible.

Solution : Virtual Box

Dans ce type de technologie, nous avons comme solution disponible Virtual Box. Ce logiciel permet de faire des tests de nouveaux OS[2]. C'est un produit que nous utilisons pour tester les nouvelles fonctionnalités d'un nouveau système ou une nouvelle distribution de Linux. Cette approche évite de garder une machine physique pour les tests, ou bien de formater à chaque sortie d'un nouveau système ; elle est très utile pour la veille technologique.

Nous rappelons que ce type de technologie est très consommateur en ressources matérielle. La version stable de Virtual Box est à ce jour la version 3.2.0; c'est un produit qui appartient aujourd'hui à *Oracle (*depuis le rachat de *SUN*). Il est disponible pour toutes les distributions Linux et se présente sous forme de package soit en .deb ou encore en *.rpm.* Une procédure d'installation est disponible sur le site : *http://www.virtualbox.org/wiki/VirtualBox.*

Para-virtualisation & virtualisation avec hyperviseur

Ce dernier type de virtualisation peut être classé en deux catégories que nous allons traiter dans ce paragraphe :

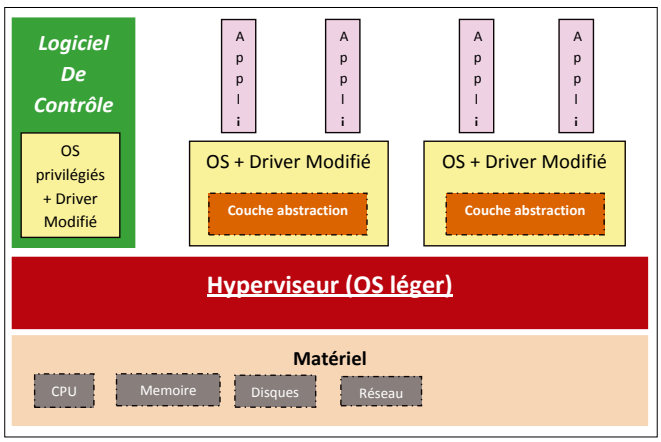


Figure 4. *Virtualisation avec hyperviseur*

Figure 5. *Interface Web de management Proxmox*

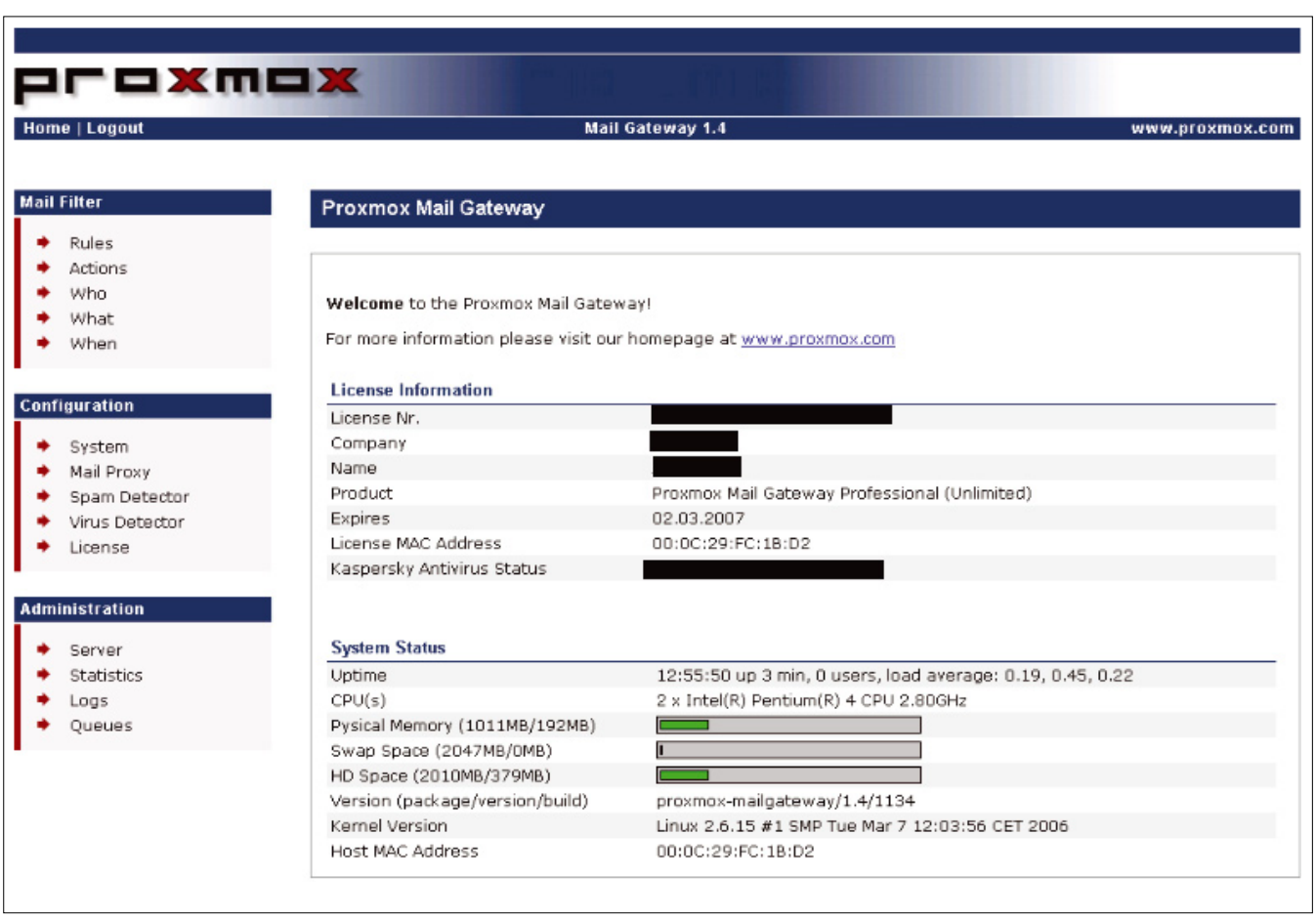


Figure 6. *Interface management Proxmox : Information serveur virtuel*

Tableau 1. *Comparatif technique des solutions de virtualisation*

	Cloisonnement	Para-virtualisation	Hyperviseur	Totale
OpenVz	✓			
Linux Vserver	✓			
Promox	✓			✓
Xen		✓	✓	✓
Virtual Box				✓
KVM + Qemu				✓
KVM + Virtio		✓		

- La para-virtualisation.
- La virtualisation avec hyperviseur.

La technologie est identique, la différence se fait au niveau des accès aux ressources matériel, ce que nous verrons par la suite.

***Para-virtualisation* et principe de fonctionnement**

Ce type de virtualisation est une évolution de la précédente. La virtualisation totale n'avait besoin d'aucune modification des systèmes hôtes; or, dans le cas de la para-virtualisation, nous avons affaire à des systèmes hôtes modifiés, ainsi que celui du système invité. Ces modifications sont nécessaires pour améliorer les performances des machines virtuelles (voir Figure 3).

On s'aperçoit qu'avec la para-virtualisation, contrairement à la solution précédente, la machine virtuelle est modifiée. Celle-ci dispose maintenant d'une interface de para-virtualisation qui lui sert à obtenir des accès privilégiés, soit au niveau du matériel, soit au niveau du système hôte. Cette modification permet d'améliorer nettement les performances des machines virtuelles. Le seul inconvénient pour bénéficier au mieux des gains de performances, est qu'il faut utiliser les pilotes modifiés, sinon on se retrouve dans le cas de la virtualisation totale.

Conséquence directe de cette technologie, **les systèmes invités ont conscience d'être virtualisés** (du fait qu'ils intègrent des pilotes modifiés).

Lexique :

[1] **TCO** : Total cost ownership ou coût total de possession.
[2] **OS** : operating system ou système d'exploitation.
[3] **Linux HA** : pour high-availibility ou haute disponibilité.
[4] **Domain U** : pour domain unpriviliged ou domaine non privilégié.
[5] **KVM** : Kernel-based Virtual Machine.
[6] **Intel-VT et AMD-D** : jeux d'instructions processeur permettant de faire tourner plusieurs systèmes d'exploitation.
[7] **LAMP** : plateforme basée sur la suite de logiciels : **L**inux **A**pache **M**ySQL **PHP.**

Hyperviseur

Pour le cas des hyperviseurs, le principe de fonctionnement est quasiment le même que dans le cas de la para-virtualisation : l'hyperviseur est une évolution de cette dernière, avec toujours un OS modifié mais très légèrement (voir Figure 4).

La différence majeure est que **l'hyperviseur est là uniquement pour gérer les ressources matérielles**. Ensuite, il fait appel à un OS[2] privilégié qui est utilisé pour administrer l'hyperviseur. Cet **OS[2] privilégié est au même niveau que les systèmes invités** et partage ainsi les ressources alors que, dans le cas de la para-virtualisation, c'est le système hôte qui a un accès direct au matériel. Avec cette solution, c'est l'hyperviseur qui gère directement les ressources et les accès concurrentiels à ces mêmes ressources, grâce à quoi nous obtenons une granularité de réglage beaucoup plus fine.

Cette technologie est employée le plus souvent pour consolider des infrastructures et mettre en œuvre des solutions de disponibilité. Les solutions proposées disposent à ce jour des interfaces de management de haut niveau et très intuitives.

Ce type de technologie reste malgré tout la plus utilisée, car une des plus performantes actuellement. De plus, elle offre davantage de possibilités que les dernières technologies. En effet, cette technologie permet de virtualiser n'importe quel système fonctionnant sur une architecture X86.

Solution : Xen

Dans cette technologie, la solution que nous n'avons pas encore eu l'occasion de tester (car nous avons acquis récemment son concurrent direct : ne fait pas l'objet de cet article) est bien sûr la solution ***Xen.*** Xen est actuellement dans sa version 3, et permet de virtualiser n'importe quel système invité non modifié. Il faut tout de même posséder un processeur avec les jeux d'instructions Intel-VT ou AMD-V (pour version HVM-Xen).

Dans son fonctionnement, Xen nomme son système privilégié Domain 0 pour la gestion de l'infrastructure Xen, et les systèmes invités *Domain U*[4]. Il faut savoir,

Sur Internet, sources utilisées pour la rédaction :

- *http://fr.wikipedia.org/wiki/Virtualisation,*
- *http://wiki.openvz.org/Main_Page,*
- *http://xen.org/,*
- *http://www.linux-kvm.org/page/Main_Page,*
- *http://www.proxmox.com/,*
- *http://fr.wikipedia.org/wiki/Intel_VT,*
- *http://fr.wikipedia.org/wiki/AMD-V#Pacifica.2FAMD-V,*
- *http://www.linux-ha.org/wiki/Main_Page.*

sauf erreur de notre part, que la version 2 de Xen ne permettait pas de virtualiser des systèmes non modifiés, et donc pas de virtualisation de Windows possible. L'installation de Xen reste assez simple, le noyau modifié pour l'hyperviseur est disponible le plus souvent par package dans les distributions Linux. Il est à noter que *Citrix* propose aujourd'hui un hyperviseur basé sur Xen.

Autres solutions : Proxmox

Nous ne pouvons pas terminer cet article sans parler de la solution Proxmox. C'est une solution hybride, c'est-à-dire que, contrairement aux solutions examinées qui ne peuvent gérer qu'une seule technologie (voire deux technologies mais très proches l'une de l'autre : para-virtualisation et virtualisation complète), Proxmox permet un accès à deux technologies de virtualisation totalement différentes :

- La virtualisation par cloisonnement avec OpenVz, virtualisation légère.
- La virtualisation complète avec le couple KVM[5] + QEMU.

Proxmox propose aujourd'hui cette alternative ; c'est une solution basée sur une distribution Debian Lenny 5.0.3 qui utilise des modules du noyau disponible comme KVM[5] pour la virtualisation complète. Par contre, pour faire fonctionner cette dernière, il faut absolument disposer d'un processeur avec les jeux d'instructions Intel-VT[6] ou AMD-V[6].

De plus, les développeurs ont prévu une interface de management simple et pertinente, et font apparaître la notion de cluster pour deux serveurs Proxmox.

Pour les utilisateurs habitués aux solutions de virtualisation, et surtout à OpenVz, nous conseillons vivement d'essayer cette solution.

Récapitulatif

Voici un tableau récapitulatif des solutions abordées dans l'article et de leurs technologies (voir Tableau 1). Pour conclure cet article, il nous paraît utile de rappeler les multiples intérêts à virtualiser les infrastructures, aussi bien sur un plan technique que financier :

- Meilleure disponibilité du Système d'information.
- Suppression des redondances.
- Des plateformes de test plus faciles et rapides à déployer.
- Une diminution du TCO[2].

Il convient toutefois de remarquer que nous disposons actuellement de nombreuses solutions : en faire le tri, ou simplement de la veille technologique devient de plus en plus difficile. Nous avons néanmoins essayé, tout au long de l'article, de présenter rapidement, sans rentrer dans des explications techniques trop détaillées, les solutions de virtualisation actuellement sur le marché.

Lorsque l'on étudie une solution de virtualisation, les paramètres importants sont :

- L'évaluation du contexte.
- Le type de virtualisation nécessaire.

Nous avons pu voir que, selon la technologie utilisée, on obtient des performances totalement différentes, ce qui nous permet d'établir le classement suivant, de la plus performante à la moins performante :

- Virtualisation par Isolation ou cloisonnement (virtualisation légère).
- Hyperviseur.
- Para-virtualisation.
- Virtualisation complète.

L'installation de chacune d'elles diffère, ainsi que leur complexité. Il est possible de trouver sur internet des tutoriels pour accompagner l'utilisateur. Nous espérons, au travers de cet article, avoir pu apporter des éléments de réflexion et certaines réponses pour faciliter la prise de décision. Nous tenons à remercier la rédaction de nous avoir donné l'occasion de rédiger cet article.

JEAN-FRANÇOIS ALBERTINI

Il occupe un poste d'administrateur système et réseau au sein d'un centre de ressources informatiques d'une université. Il a en charge, entre autres, les solutions de virtualisation mises en production ainsi que les baies de stockages. Aujourd'hui une cinquantaine de machines sont virtualisées et réparties sur plusieurs clusters en production hébergés les baies de stockages. L'auteur s'occupe aussi avec ses collaborateurs de l'exploitation de l'ensemble des services proposés par le centre de ressources informatiques. Pour le contacter : albertini.jf@free.fr.

Introduction à la POO avec PHP

Découvrez comment développer vos applications PHP en utilisant la programmation orientée objet, qui vous permettra de concevoir des productions plus fiables, plus puissantes et plus faciles à maintenir !

Cet article explique :

- Les bases de la programmation orientée objet dans PHP.
- Quelques recettes vous permettant de modéliser votre application.
- Les motifs de conceptions les plus courants en PHP.

Ce qu'il faut savoir :

- Il est préférable d'être familier avec les bases de PHP5.
- Comprendre le principe des fonctions en programmation.

PHP devient mature : depuis sa version 5, il offre une implémentation du paradigme objet (voir terminologie) relativement complète, qui permet de développer des applications web puissantes, plus faciles à améliorer et à maintenir.

La programmation orientée objet (POO, ou OOP, *oriented object programming*) a été introduite pour permettre de fixer des méthodes facilitant la conception et le développement de programmes. Elle permet notamment d'améliorer l'organisation, de faciliter le travail en groupe, la réutilisation ou la maintenance du code.

L'objet

Le concept de base de la POO est l'objet. C'est une *brique logicielle* représentant une idée, une action ou une entité *du monde réel*. Un objet peut représenter une infinité de notions : un champ dans un formulaire, une communication entre deux serveurs, et même un animal ou un pantalon !

La programmation par objets consiste à donner un ensemble de valeurs ou d'opérations pour chaque type d'objet, en rendant les points communs et leurs relations explicites, et en masquant leur fonctionnement interne. Dans un programme, chaque donnée traitée et d'un certain type, PHP propose un ensemble de types natifs (un nombre entier, une chaîne de caractères, un booléen), ou des types structurés, comme les tableaux. La programmation par objet présente une nouvelle famille de types : des types *abstraits* définis par le programmeur.

La classe

Les classes correspondent à l'expression d'un type abstrait. Une classe réunit des données composant le type et les opérations permettant de traiter ces données. Par exemple, un verre serait définit par sa forme, le volume qu'il peut contenir et son contenant, les opérations liées au verre seraient remplir ou vider.

Un objet est une instance d'une classe, c'est à dire la réalisation de cette classe. Par analogie, on peut considérer que la classe est un moule et l'objet un gâteau ayant la forme de ce moule. Une classe est composée de propriétés (ou d'attributs), ce sont des variables qui correspondent aux données composant et caractérisant le type, et de méthodes qui sont des fonctions définies à l'intérieur de la classe et permettant de manipuler l'objet. Méthodes et propriétés peuvent être désignées sous le terme commun de *membre de la classe*.

Pour ne pas se perdre et se familiariser avec ces deux premières notions, voici un premier exemple : l'écriture en PHP du type *verre* que nous avons imaginé plus tôt.

Terminologie

- Paradigme : un paradigme est un ensemble de règles permettant de représenter des informations cohérentes, en informatique, on peut considérer que c'est un style de programmation, une manière dont le programmeur voit son programme s'organiser et s'exécuter.

La définition d'une classe s'écrit ainsi :

```
class Verre { /* contenu de la classe*/ }
```

Les propriétés et méthodes de cette classe s'écrivent entre les accolades. Ainsi l'implémentation de notre verre sera :

```
class Verre {
  public $forme;
  public $volume;
  public $contenu;

  function remplir() { /* opération remplir
*/ }
  function vider() { /* vider */ }
}
```

Maintenant, que notre classe est définie, nous allons pouvoir créer un objet verre, que l'on pourra manipuler. Pour l'instant le mot clé public reste obscur, son sens sera décrit lors du paragraphe suivant.

```
$unVerreEau = new Verre();
```

On accède aux membres de cet objet à l'aide de l'opérateur -> :

```
$unVerreEau->contenu = 'eau';
```

Pour accéder aux membres à l'intérieur d'une méthode de la classe, on utilisera le mot clé `$this`.

Ainsi, on peut écrire nos opérations `remplir()` et `vider()` :

```
function remplir($quoi) { $this->contenu =
$quoi; }
function vider() { $this->contenu = 'rien'; }
```

Encapsulation

L'un des principes fondamentaux de la POO est l'encapsulation, cela signifie que pour utiliser un objet, il n'est pas nécessaire de connaître le fonctionnement interne de sa classe : on peut conduire une voiture sans être doué en mécanique !

Ainsi, si schématiquement, plusieurs développeurs travaillent ensemble, que l'un développe la classe et l'autre l'utilise, ce dernier n'aura besoin de connaître que l'interface de la classe, c'est à dire la liste des méthodes et attributs qu'il peut utiliser. Dans certains cas, la classe comprend des membres qui ne doivent pas être manipulés directement, l'encapsulation permet alors de protéger ces membres et de les cacher au développeur qui utilise la classe. Dans la plupart des cas, on peut souhaiter contrôler les données que l'utilisateur

Listing 1. *La classe Verre*

```
class Verre {
  public $forme;
  public $volume;
  private $contenu;
  public function remplir($quoi) {
    $this->contenu = $quoi;
  }
  public function vider() {
    $this->contenu = 'rien';
  }
}
```

Listing 2. *La manipulation des références aux objets*

```
class Test {
  public $nombre;
}
$objet = new Test();
$objet->nombre = 10;
$autre_objet = $objet;
$autre_objet->nombre++;
echo 'objet : '.$objet->nombre.', autre objet :
'.$autre_objet->nombre;
```

souhaite assigner aux attributs, ainsi on peut protéger leur accès.

Les membres accessibles sont dit publics et ceux que l'on souhaite protéger sont privés. Il existe un troisième niveau de protection que l'on verra un peu plus tard. Une écriture possible de notre classe `Verre` est proposée dans Listing 1.

Les membres privés sont accessibles et utilisables uniquement au sein de la classe. Avec la nouvelle implémentation de la classe, l'opération `$unVerreEau->contenu = 'eau';` générera une erreur fatale qui stoppera l'exécution du script.

Attention, si vous utilisez un autre objet de la même classe au sein de la classe, ses membres privés restent accessibles. On peut imaginer ainsi une méthode `transvaser()` dans notre classe `Verre` qui fonctionnerait ainsi :

```
public function transvaser(Verre $autre) {
  $this->remplir($autre->contenu);
  $autre->vider();
}
```

Ceci ne fonctionne bien sûr qu'avec un objet issu de la classe `Verre`. On peut également noter que dans cet exemple, nous avons précisé le type du paramètre `$autre` de la fonction, ainsi, lors de l'appel de cette méthode, si le paramètre n'est pas un objet `Verre`, cela déclenche une erreur fatale. Cette fonctionnalité ne fonctionne aujourd'hui que pour les types abstraits.

Le cycle de vie de l'objet

Chaque objet possède son propre cycle de vie, il est construit, utilisé et détruit. L'objet est construit lors de l'exécution de l'instruction `$objet = new Classe();`

Listing 3. *L'implémentation de la classe Vaisselle*

```
class Vaisselle
{
  private $forme;
  private $contenu;
  public function _ _construct($forme, $contenu =
'rien')
  {
    $this->forme = $forme;
    $this->remplir($contenu);
  }
  public function remplir($quoi)
  {
    $this->contenu = $quoi;
  }
  public function vider()
  {
    $this->contenu = 'rien';
  }
}
```

ici, la variable `$objet` référence une nouvelle instance de la classe `Classe`. Cela signifie que la variable `$objet` représente l'adresse en mémoire de l'objet nouvellement créé, et les conséquences sont nombreuses, car plusieurs variables peuvent référencer un même objet.

Pour être plus clair, analysons un extrait de code. Dans celui-ci, nous assignons à une variable une valeur entière, nous assignons cette variable à une autre que nous modifions.

```
$nombre = 10;
$autre _ nombre = $nombre;
$autre _ nombre++;
echo 'nombre : '.$nombre.', autre nombre :
'.$autre _ nombre;
```

Le résultat sera `nombre : 10, autre nombre : 11`. Regardons maintenant un autre exemple, présenté dans le Listing 2, où nous manipulons cette fois un objet.

Ici le résultat sera `objet : 11, autre objet : 11`, les deux variables `$objet` et `$autre_objet` référencent en effet le même objet en mémoire. Il est possible de créer un nouvel objet en remplaçant l'instruction `$autre_objet = $objet;` par `$autre_objet = clone $objet;`, on retrouve alors le comportement observé en manipulant les types primitifs. Attention cependant, `$autre_objet` n'est pas un alias de la variable `$objet`, comme il aurait été le cas avec cette assignation : `$autre_objet = &$objet`.

Observez cet exemple, pour être un peu plus clair :

```
$a = 1;
$b = &$a;
$b = 2;
echo $a;
```

Le résultat affiché est `2` : `$b` est `$a` sont la même variable, mais portant un nom différent. Tandis que :

```
$a = new Test();
$a->nombre = 1;
$b = $a;
$b = new Test();
$b->nombre = 2;
echo $a->nombre.', '.$b->nombre;
```

Le résultat est `1, 2`, `$b` et `$a` ne référencent plus le même objet. Enfin, dans ce dernier cas :

```
$a = new Test();
$a->nombre = 1;
$b = &$a;
$b = new Test();
$b->nombre = 2;
echo $a->nombre.', '.$b->nombre;
```

Le résultat est ici `2, 2`, `$a` et `$b` sont la même variable, le premier objet `Test` est remplacé par le second. L'opérateur `==` permet de comparer deux objets, il retourne vrai si l'ensemble des attributs des deux objets sont identiques. Pour vérifier que deux variables référencent le même objet (la même instance), il faudra utiliser l'opérateur `===`.

Il est possible de définir une méthode exécutée lors de la création d'un objet. Cette méthode spéciale s'appelle le constructeur. Cette méthode doit porter le nom `__construct()` (avec deux _). Elle sert généralement à initialiser les valeurs des principaux attributs de l'objet.

Par exemple, le constructeur de notre classe `Verre` pourrait être :

```
public function _ _construct($forme, $volume,
$contenu = 'rien') {
  $this->forme = $forme;
  $this->volume = $volume;
  $this->remplir($contenu);
}
```

L'instanciation de la classe s'écrit alors : `$unVerreEau = new Verre('classique', 0.33, 'eau');`.

Un objet est détruit lorsqu'il n'existe plus de variable le référençant. Tout comme le constructeur, il existe une méthode qui peut être appelée lors de la destruction d'un objet, elle s'appelle logiquement un destructeur, elle porte le nom `__destruct()`. En PHP cette méthode n'est pas utilisée très souvent, elle peut être utilisée pour libérer des ressources : fermer une connexion à la base de données ou un descripteur de fichier par exemple.

Polymorphisme

Un autre aspect essentiel de la programmation par objets est appelé polymorphisme. L'idée consiste à pouvoir modifier un type pour obtenir une hiérarchie. Par exemple, on peut imaginer une classe `Vaisselle`, de

laquelle découlerait notre classe `Verre`. Le but est de permettre une programmation plus générique, en réutilisant les mêmes méthodes pour plusieurs types. On dit alors qu'une classe hérite d'une classe parent. Vous pouvez consulter la définition de la classe `Vaisselle` dans le Listing 3.

Pour préciser que la classe `Verre` hérite de la classe `Vaisselle`, on utilise le mot clé `extends` lors de sa déclaration : `class Verre extends Vaisselle`. Vous pouvez alors utiliser les membres de la classe `Vaisselle` directement en manipulant un objet Verre.

Cependant, un `Verre` est un type de vaisselle particulier, il faut donc spécialiser la classe. On y ajoute l'attribut `$volume` et la méthode `transvaser()`. La classe `Verre` sera donc :

```
class Verre extends Vaisselle {
  private $volume;
  public function transvaser($autre) {
    $this->remplir($autre->contenu);
    $autre->vider();
  }
}
```

Le constructeur de la classe `Verre`, héritant directement de celui de la classe `Vaisselle` est cependant incomplet : il est impossible de définir le volume de ce `Verre`. C'est donc l'occasion d'introduire la notion de redéfinition : quand une classe `fille` modifie le comportement d'une méthode de la classe mère. En PHP, redéfinir une fonction dans la classe `fille` se fait de la même manière qu'une définition normale.

Notre classe `Verre` contiendra alors la méthode :

```
public function _ _construct($forme, $volume,
$contenu = 'rien') {...}
```

Le constructeur de la classe `Vaisselle` n'est cependant plus exécuté, il faut donc l'appeler explicitement dans le constructeur de `Verre`. Pour appeler cette méthode, nous avons besoin de deux nouvelles notions : l'opérateur de résolution de portée `::` (appelé *Paamayim Nekudotayim* dans le folklore PHP, ce qui signifie simplement *double deux points*) permet d'accéder aux membres redéfinis par la classe `fille` ; d'autre part, le mot clé `parent` réfère à la classe mère la plus proche portant une définition de cette fonction.

```
public function _ _construct($forme, $volume,
$contenu = 'rien') {
  parent:: _ _construct($forme, $contenu);
  $this->volume = $volume;
}
```

Généralement, les constructeurs sont appelés dans l'ordre de la spécialisation, c'est à dire qu'on appelle le constructeur parent avant d'effectuer les opérations spécifiques à la classe `fille`. C'est le contraire pour les destructeurs : on appelle le destructeur parent une fois que les ressources propres au fils sont détruites. Cette pratique permet d'éviter les mauvaises surprises si lors de futurs développements les constructeurs et destructeurs parents sont modifiés.

Il est parfois nécessaire d'interdire la redéfinition d'une fonction. Pour cela, il existe le mot clé `final` à placer avant la définition de la méthode. On peut par exemple interdire la redéfinition de la méthode `vider()` qui n'a pas de raison de changer ainsi : `public final function vider()`. Il est également possible d'interdire d'utiliser une classe pour l'héritage en la rendant elle même `final` (`final class MaClasse`).

Il est par ailleurs possible d'effectuer plusieurs niveaux d'héritage. Par exemple, `Vaisselle` peut être parent de `Verre`, qui serait la classe mère de `VerreAPied`. Dans ce cas, utiliser `parent::__construct()` dans la classe `VerreAPied` référera au constructeur de la classe `Verre`. Si cependant vous souhaitez utiliser le constructeur définit dans `Vaisselle`, utilisez la syntaxe : `Vaisselle::__construct()`.

Il est temps d'introduire le troisième niveau de visibilité, situé entre public (`public`) et privé (`private`). Le niveau *protégé* (`protected`) rend le membre de la classe invisible en dehors de celle-ci mais visible pour les classes héritant de ce membre. Dans certains cas, la classe mère est une classe qui ne doit pas être instanciée. On parle alors de classe abstraite. Une classe abstraite contient généralement une ou plusieurs méthodes abstraites, ce qui signifie qu'elle n'ont pas d'implémentation. En clair, ces méthodes devront être redéfinies dans une classe concrète héritant de cette classe abstraite. L'abstraction est utilisée quand deux types enfants doivent proposer la même méthode mais ayant un comportement différent. Pour indiquer qu'une classe est abstraite, ont la définit ainsi : `abstract class MaClasse`, de même une méthode abstraite est déclarée ainsi : `abstract public function maMethode();` la méthode peut être publique, protégée ou privée, sa déclaration est suivie d'un point-virgule, puisqu'elle ne contient pas de corps.

Attention, une classe abstraite ne peut pas être *finale* puisque par définition, celle-ci doit être redéfinie. Sachez pour finir que bien que l'héritage à plusieurs niveaux soit possible, l'héritage multiple n'est quand à lui, pas permis dans PHP : une classe n'a qu'un parent direct.

Les interfaces

Les interface sont, d'un point de vue purement technique, semblables à des classes abstraites dont la totalité des méthodes seraient abstraites. Cependant, ces deux notions se distinguent d'un point de vue sémantique : une interface sert en effet à définir la liste des

Listing 4. *Manipuler des membres statiques*

```
class MaClasse {
  public static $nbObjets = 0;
  public function __construct() {
    MaClasse::$nbObjets++;
  }
  public static function nombreInstances() {
    return self::$nbObjets;
  }
  public function autresInstancesQueMoi() {
    return self::$nbObjets-1;
  }
}
```

Listing 5. *Les constantes __CLASS__ et __METHOD__*

```
class MaClasse {
  public static function test() {
    echo 'classe : '.__CLASS__.', méthode : '.__METHOD__;
  }
}
MaClasse::test(); // affiche classe : MaClasse, méthode
: MaClasse::test
```

méthodes publiques qu'une classe devra implémenter. Une interface ne contient donc aucune propriété et que des méthodes publiques.

Une interface permet de spécifier un ensemble de méthodes qui seront assurément implémentées dans une classe, afin de garantir dans d'autres développements que ces méthodes pourront être utilisées en ayant le même comportement même si elles n'ont pas le même fonctionnement interne.

On peut faire l'analogie suivante : la totalité des lecteurs ont une série de boutons : lecture, *pause*, *stop avancer*, *reculer*. Ces boutons constituent l'interface de l'appareil, et bien qu'un lecteur de DVD, un lecteur de CD ou un lecteur de MP3 n'aient rien de comparables à l'intérieur, ils proposent ces mêmes boutons qui déclencheront le même comportement pour l'utilisateur.

Une interface est déclarée ainsi :

```
interface MonInterface {
  public function maFonction();
}
```

Pour qu'une classe implémente une interface, on utilise le mot clé `implements` : `class LecteurMP3 implements interfaceLecteur`. Nous verrons plus tard un exemple concret classique d'utilisation des interfaces. Le mécanisme d'héritage existe aussi pour les interfaces, l'interface `interfaceLecteur` peut hériter de `interfaceAppareilElectrique` : `interface interfaceLecteur extends interfaceAppareilElectrique`. La classe implémentant `interfaceLecteur` comprendra alors les méthodes publiques de `interfaceLecteur` et `interfaceAppareilElectrique`. Une classe peut par ailleurs implémenter plusieurs interfaces, à condition que celles-ci ne possèdent pas de méthodes communes (source de confusion), la liste des interfaces est séparée par une virgule : `class LecteurMP3 implements interfaceLecteur, interfaceAppareilElectrique`. Et bien que l'héritage de plusieurs classes soit interdit, en PHP, une interface peut hériter de plusieurs interfaces.

Membres statiques et constantes de classes

Il est parfois nécessaire d'utiliser certains membres de la classe sans créer d'instance de celle-ci. Ces membres sont donc indépendants de toute instance de la classe qui les contient. On dit de ces membres qu'ils sont statiques. Un membre est déclaré statique à l'aide du mot clé `static`.

Un attribut statique s'appelle une variable de classe, et sa valeur est commune à l'ensemble des objets créés à partir de la classe. Une variable de classe est généralement utilisée pour partager une donnée entre les différentes instances de celle-ci, les compter par exemple. La manipulation des membres statiques est présentée au Listing 4.

Nous avons vu ici l'opérateur de résolution de portée utilisé de deux manières différentes, mais qui ont le même résultat : `MaClasse::$nbObjets` réfère à la variable `$nbObjets` de la classe `MaClasse`, `self::$nbObjets`, réfère à la variable `$nbObjets` de la classe courante : `self` est comparable à `$this` dans un contexte statique. Une méthode statique étant appelée indépendamment d'une instance, le mot clé `$this` ne peut être utilisé. Par ailleurs il est interdit d'accéder à un membre statique depuis un objet, mais une méthode le peut : `$objet->nbObjets` n'existe pas, mais vous pouvez utiliser la valeur de `MaClasse::$nbObjets` dans la méthode `$objet->autresInstancesQueMoi()`.

Les outils fournis par PHP

Pour développer vos applications en utilisant la POO, PHP met à disposition un ensemble de fonctions, classes et mots clés. Ceux-ci permettent notamment d'explorer la structure d'une classe de déterminer la classe dont est issu l'objet ou encore d'étendre les capacités de ce dernier.

Les constantes magiques

PHP introduit deux constantes magiques utilisables dans le contexte d'une classe : la constante __CLASS__ correspond au nom de la classe courante et __METHOD__ le nom de la méthode courante. Le Listing 5 présente un exemple de manipulation de ces constantes.

Le mot clé instanceof

PHP est un langage dit faiblement typé, une variable peut donc contenir n'importe quel type de donnée : un nombre, une chaîne de caractères ou encore un objet. Or, dans certains cas, il est nécessaire de vérifier que l'objet manipulé est bien d'un certain type, c'est à dire que cet objet est bien une instance d'une

classe donnée (ou d'une classe héritant de cette classe, ou implémentant cette interface). C'est le rôle du mot clé `instanceof` qui s'utilise comme un opérateur de comparaison classique. Par exemple, pour vérifier que l'objet `$unVerreEau` est bien un `Verre`, on utilisera la syntaxe `$unVerreEau instanceof Verre` dans une condition.

Les fonctions de manipulation d'objets

Le langage PHP met à votre disposition une série de fonctions vous permettant d'effectuer un certain nombre d'opérations sur les classes et objets, dont voici la liste des principales :

- `class_exists('MaClasse')` retournera vrai si la classe `MaClasse` est définie,
- `get_class_methods('MaClasse')` retournera un tableau de chaînes de caractères contenant les noms des méthodes visibles de la classe `MaClasse`,
- `get_class_vars('MaClasse')` retournera un tableau dont les clés sont les noms des propriétés et la valeur correspondante sera la valeur par défaut de la propriété à l'instanciation de l'objet (avant l'exécution du constructeur),
- `get_class($objet)` retournera le nom de la classe de l'objet passé en paramètre,
- `get_declared_classes()` retournera un tableau contenant la liste des classes existant au moment de l'exécution de la fonction,
- de la même manière, `get_declared_interfaces()` retournera la liste des interfaces disponibles,
- `get_object_vars($objet)` est similaire à `get_class_vars()`, mais les valeurs sont les valeurs courantes de l'objet,
- `interface_exists()` retournera vrai si l'interface `MonInterface` est déclarée,
- `is_subclass_of($objet, 'ClasseParent')` retournera vrai si l'objet `$objet` est l'instance d'une classe héritant de `ClasseParent`, mais attention, si `$objet` est une instance de `ClasseParent`, alors la méthode retournera faux.
- `method_exists('MaClasse', 'maMethode')` vaudra vrai si la méthode `maMethode` existe dans la classe `MaClasse`, qu'elle soit visible ou non dans la portion de code où la fonction est exécutée,
- et son équivalent pour tester l'existence d'une propriété dans une classe `property_exists('MaClasse', 'maPropriete')`.

Les fonctions et méthodes magiques

Des fonctions et méthodes magiques ont été introduites en PHP5. Elles sont appelées dynamiquement lors de certains événements, comme l'appel d'une classe ou d'une méthode inconnue.

Listing 6. *Le clonage d'objet et la gestion de membres partagés*

```
class Utilisateur { public $nom; }
class Message { public $auteur; }
$message = new Message();
$message->auteur = new Utilisateur();
$message->auteur->nom = 'Paul';
$autre_message = clone $message;
$autre_message->auteur->nom = 'Pierre';
echo 'Auteur : '.$message->auteur->nom.', autre auteur
: '.$autre_message->auteur->nom;
```

La fonction `__autoload($class_name)` est appelée lorsqu'une classe inconnue est utilisée dans le code, il est ainsi possible de déclarer la classe manquante avant que PHP ne renvoie une erreur. Le but de cette méthode est de limiter le nombre d'inclusion en tête de fichier, les développeurs utilisant généralement le formalisme *un fichier pour une classe*. La fonction `__autoload()` doit être définie dans le script courant : ce n'est pas une méthode, son seul paramètre et le nom de la classe. Voici un exemple sommaire d'implémentation de la fonction magique `__autoload()` :

```
function __autoload($class) {
  require '/lib/class.'.strtolower($class).'.php';
}
```

Bien que cette méthode soit très utile et fasse gagner du temps au développeur, elle ralentit cependant l'exécution globale du script. Réservez son utilisation pour les classes qui risquent d'être régulièrement déplacées dans la structure de fichier.

Avec l'arrivée des espaces de noms (ou *namespaces*) dans PHP 5.3, l'utilisation de cette fonction magique peut être amélioré. Je vous invite à consulter un article à ce sujet à l'adresse suivante *http://www.martiusweb.net/post/Chargement-automatique-de-classes-avanc%C3%A9-avec-PHP-5* pour en savoir plus.

Il existe également une série de méthodes magiques que l'on peut implémenter dans des classes.

La méthode magique `__clone()` permet de contrôler le clonage d'un objet, que nous avons vu en étudiant le cycle de vie d'un objet. En effet, lorsque l'on clone un objet, PHP crée une nouvelle instance et copie la valeur des propriétés de l'objet d'origine dans le clone. Cependant, si l'une des propriétés de cet objet est un objet, alors l'objet d'origine et le clone possèdent une propriété référençant la même instance d'un objet. Un exemple de code sera certainement plus clair, je vous invite donc à vous reporter au Listing 6.

Le résultat de ce code sera `Auteur : Pierre, autre auteur : Pierre`, car les deux objets `$message` et `$autre_message`, bien qu'étant différents, référencent le même objet `$auteur`. Il faut donc spécialiser le clonage :

Listing 7. *Une collection d'objets*

```
class Collection
{
  private $type = '';
  protected $collection = array();

  public function _ _construct($type) {
    $this->type = $type;
  }

  public function _ _set($obj, $valeur) {
    if($valeur instanceof $this->type)
      $this->collection[$obj] = $valeur;
    else
      throw new ErrorException('L\'objet proposé n'est pas
du bon type');
  }

  public function _ _get($obj) {
    if(isset($this->collection[$obj]))
      return $this->collection[$obj];
    else
      throw new ErrorException('L\'objet demandé n\'existe
pas');
  }

  public function _ _isset($obj) {
    return isset($this->collection[$obj]);
  }

  public function unset($obj) {
    if(isset($this->collection[$obj])
      unset($this->collection[$obj]);
  }

  public function _ _call($nom, $parametres) {
    if(is_callable($nom)) {
      $j = sizeof($parametres);
      if($j > 0) {
        for($i = 0; $i <= $j; ++$i)
          $parametres[$i] = $this->collection[$parametre
s[$i]];
        return call_user_func_array($nom,
$parametres);
      }
    else
      return call_user_func($nom, $this->collection);
    }
  }
}

$c = new Collection('Utilisateur'); // crée une collection
$c->pierre = new Utilisateur('Pierre'); // Ajoute un
nouvel utilisateur Pierre
$c->paul = new Utilisateur('Paul'); // Ajoute Paul
if(isset($c->pierre)) { // Si Pierre existe dans la
collection
  unset($c->pierre); // on le supprime
}
$c->marc = new Utilisateur('Marc'); // Ajout de Marc
$c->ksort(); // On trie la collections
```

```
class Message {
  public $auteur;
  public function _ _clone() {
    $this->auteur = clone $this->auteur;
  }
}
```

Lorsqu'un objet est cloné, si cette méthode magique est définie, alors le clone l'exécutera, ainsi, l'objet `$auteur` n'est plus le même dans les deux messages.

Les méthodes `__get()` et `__set()` sont appelées lorsqu'une propriété n'existant pas ou n'étant pas visible est utilisée. Les méthodes magiques `__isset()` et `__unset()` sont appelées lors de l'utilisation des fonctions `isset()` (pour tester si une variable est initialisée) et `unset()` (pour supprimer une variable de la mémoire) sur ces mêmes propriétés.

Enfin, la méthode `__call()` est exécutée quand une méthode inconnue ou invisible est appelée, `__callStatic()`, disponible depuis PHP 5.3 possède le même comportement, mais dans un contexte statique, comme son nom l'indique.

Cet ensemble de méthodes peut par exemple être utile pour gérer une collection d'objets et proposer une syntaxe plus simple pour les utilisateurs de votre classe. Le Listing 7 présente un exemple d'implémentation d'une telle collection.

Les méthodes `__sleep()` et `__wakeup()` permettent respectivement de contrôler le comportement de l'objet lorsqu'il est sérialisé (avec `serialize()`) et désérialisé (avec `unserialize()`), afin de pouvoir réutiliser l'objet entre deux scripts. Elles s'utilisent dans des cas d'utilisation précis, et sont progressivement remplacées au profit de l'interface `Serializable`.

La sérialisation (ou linéarisation) d'un objet permet d'obtenir une chaîne de caractères qui peut être stockée (par exemple dans une variable de session ou une base de données) pour le recharger plus vite lors d'une prochaine exécution. Par exemple, il peut être intéressant de stocker certains objets collectant des données dans une base pour limiter le nombre de requêtes pendant qu'un utilisateur navigue sur le site. Cependant, certaines données, comme les ressources, ne peuvent ou ne doivent pas être sérialisées, soit pour des raisons techniques, soit pour des raisons de sécurité : généralement, on évitera de conserver un mot de passe en mémoire par exemple.

La méthode `__sleep()` est appelée avant toute sérialisation, elle doit retourner un tableau contenant la liste des attributs de la classe qui doivent être conservés. La méthode `__wakeup()` est appelée après une déserialisation, et se charge de remettre en place les données qui ont disparues lors de la sérialisation. Le résultat linéarisé respectera une structure imposée par PHP, tandis que l'interface `Serializable` vous permettra de choisir votre propre format. Par ailleurs, l'utilisation de `__sleep()`, `__wakeup()` et `Serializable` simultanément est possible.

Reportez vous au Listing 8 pour découvrir un exemple de linéarisation avec `__sleep()` et `__wakeup()`.

Enfin, la méthode `__toString()` est appelée quand l'objet est utilisé comme une chaîne de caractères :

```
class Utilisateur {
  private $nom = 'Paul';
  public function _ _toString() { return
$this->nom; }
}
```

```
$u = new Utilisateur();
echo 'Je suis '.$u;
```

affichera *Je suis Paul*.

Les interfaces et classes de la SPL

La SPL, pour *Standard PHP Library*, est à PHP ce que la SDL est aux langages C et C++ : une librairie de fonctions et classes fournies dans toute installation de PHP répondant aux premiers besoins des développeurs. En utilisant la SPL, vous aurez à disposition des structures de données (liste chaînée, pile, queue...), des itérateurs, ou des interfaces qui vous permettrons de jongler entre les différents types de structures de données standard de PHP.

La documentation de la SPL est disponible dans un chapitre du manuel PHP : *http://fr.php.net/manual/fr/book.spl.php*, mais d'autres exceptions et interfaces standards sont présentées sur les pages *http://fr.php.net/manual/fr/reserved.exceptions.php* et *http://fr.php.net/manual/fr/reserved.interfaces.php*.

Plutôt que de décrire sommairement les classes et interfaces offertes dans l'installation standard de PHP, j'ai choisi d'en présenter une plus en détail : l'interface `ArrayAccess` qui permet d'utiliser la syntaxe PHP des tableaux associatifs pour manipuler un objet. Elle ne fait pas réellement partie de la SPL mais elle est disponible dans toute installation de PHP.

L'utilisation de cette interface est présentée dans le Listing 9. Cette classe hérite de la classe `Collection` créée précédemment, une fois implémentée, vous avez à disposition une nouvelle méthode vous permettant d'accéder aux éléments de votre collection. Attention, il n'est cependant pas possible de parcourir la collection avec la structure `foreach` dans l'état actuel : la classe devrait implémenter une seconde interface `Iterator` ou `IteratorAggregate`.

Les exceptions

Le mécanisme des exceptions a été introduit dans PHP 5 et repose sur le modèle objet. Comme son nom l'indique, une exception représente un comportement *anormal* du programme. Ce mécanisme est souvent utilisé pour gérer les situations qui apparaissent comme des erreurs pour l'utilisateur. Nous allons décrire ce mécanisme dans les paragraphes qui suivent, puis l'illustrer avec un extrait de code pour clarifier le tout.

Une exception est un objet envoyé à un moment (avec le mot clé `throw`), et elle est attrapée (mot clé `catch`) par un mécanisme surveillant la portion de code (appelé bloc `try`) dans laquelle elle est envoyée.

L'objet envoyé est nécessairement une instance de la classe `Exception` (dont vous pouvez consulter l'interface sur le manuel de PHP : *http://fr.php.net/manual/fr/class.exception.php*). Il est tout à fait possible de créer des classes qui héritent de celle-ci pour raffiner le type des exceptions lancées.

Lising 8. *La sérialisation avec __sleep() et __wakeup()*

```
Class MonObjet
{
  private $fichier;
  private $adresse;
  private $donnees;
  public function _ _construct($adresse) {
    $this->adresse = $adresse;
    $this->fichier = fopen($adresse, 'r+');
  }
  public function lireLigne() {
    if($this->fichier && !feof($this->fichier))
      $this->donnees .= fgets($this->fichier)."\n";
  }
  public function _ _sleep() {
    fclose($this->fichier); // le descripteur de fichier ne
peut être conservé
    return array('adresse', 'donnees');
  }
  public function _ _wakeup() {
    $this->fichier = fopen($this->adresse, 'r+'); //
réouverture du fichier
    fseek($this->fichier, strlen($this->donnees)); // on se
replace au même endroit dans le fichier
  }
  public function _ _destruct() {
    fclose($this->fichier);
  }
}

$monObjet->lireLigne();
$ser = serialize($monObjet);
unset($monObjet);
$ressucite = unserialize($ser);
// $ressucite n'est pas à proprement parler le même
objet,
// mais il sera identique à $monObjet lorsqu'il a été
linéarisé.
```

Listing 9. *Un exemple d'utilisation de l'interface ArrayAccess*

```
class AsArrayCollection extends Collection implements
ArrayAccess
{
  public function offetSet($offset, $valeur) {
    $this->collection[$offset] = $valeur;
  }
  public function offsetGet($offset) {
    if(isset($this->collection[$offset]))
      return $this->collection[$offset];
    else return null;
  }
  public function offsetUnset($offset) {
    unset($this->collection[$offset]);
  }
  public function offsetExists($offset) {
    return isset($this->collection[$offset]);
  }
}

$c = new AsArrayCollection();
$c['pierre'] = new Utilisateur('Pierre'); // Ajouter Pierre
$c['paul'] = new Utilisateur('Paul'); // Ajouter Paul
if(isset($c['pierre']))// Si Pierre existe dans la
collection
  unset($c['pierre']); // on le supprime
```

Dans un programme, et ici dans un script PHP, des fonctions en appellent souvent d'autres et ces appels s'empilent. Lorsqu'une exception est lancée, l'exécution du segment de code dans lequel elle est envoyée est interrompue, et l'exception remonte jusqu'au premier bloc `try` qui l'englobe.

Tout comme plusieurs appels de fonctions peuvent être imbriqués, les bloc `try` peuvent l'être. Une ou plusieurs structures `catch` suivent directement le bloc `try`.

Listing 10. *Les exceptions*

```
// Définition d'exceptions personnalisées
class ActionException extends Exception {}
class TraitementException extends Exception {}

// Fonction de traitement
function traiterNombre($nombre) {
  if($nombre < 0)
    throw new TraitementException('Le nombre doit être
positif');

  // Placer ici le traitement du nombre

  return $nombre;
}

// Action pour l'utilisateur
function monAction()
{
  if(!isset($_GET['nombre']) || !is_numeric($_
GET['nombre']))
    throw new ActionException('Aucun nombre renseigné :
action annulée');

  try {
    $nombre = traiterNombre($_GET['nombre']);
  }
  catch(TraitementException $e) {
    // Transformation de TraitementException en
ActionException
    throw new ActionException($e->getMessage());
  }
}

// Exécution du programme
try {
  monAction();
}
catch(ActionException $e) {
  // Si l'exception est de type ActionException,
  // on affiche un message à l'utilisateur
  include('erreurAction.php');
}
catch(Exception $e) {
  // Les autres exceptions sont masquées et provoquent
  // une erreur 500 (code http pour les erreurs internes)
  header('HTTP/1.0 500 Internal Server Error');
}
```

Elles permettent de choisir le comportement adopté par le programme quand une exception est attrapée.

L'utilisation de plusieurs `catch` pour un même bloc `try` permet de faire varier le comportement du programme en fonction du type d'exception à traiter.

Ce n'était peut-être pas très clair, mais le Listing 10, devrait remettre ces notions en ordre. Le script du listing appelle une fonction `monAction()`, qui vérifie qu'un utilisateur a bien renseigné le paramètre `$_GET['nombre']`, pour lui faire subir un traitement par la fonction `traiterNombre()`. Cette dernière ne peut pas traiter de nombre négatif. Dans ce script, deux types dérivés de la classe Exception on été créés : `ActionException` et `TraitementException` : ces deux classes permettent de différencier la nature des exceptions reçues.

Dans notre script, les exceptions `ActionException` sont affichées dans une page d'erreur (`erreurAction.php`) ; les autres exceptions reçues généreront une erreur renvoyée par le serveur pour éviter d'effrayer l'utilisateur avec des erreurs qu'il ne peut comprendre. La fonction `monAction()` attrape les `TraitementException` pour les transformer en `ActionException`, les autres ne sont pas attrapées ici et remonteront au bloc de niveau supérieur. Dans le bloc de plus haut niveau, le deuxième `catch` attrapera toute exception qui n'a pas été attrapée par avant : une instance d'une classe fille de `Exception` en est aussi une.

Si aucun bloc n'intercepte une exception, une erreur fatale est renvoyée par PHP.

Pour finir, PHP propose plusieurs types dérivés d'exceptions dans la SPL dont la liste est dans le manuel : *http://fr.php.net/manual/fr/spl.exceptions.php*. Il faut également ajouter *ErrorException* qui fait partie de PHP mais n'est pas intégrée à la SPL.

Un peu de conception orientée objet

Vous connaissez maintenant la plupart des notions relatives à la POO existant dans PHP 5. Il est possible que vous ne soyez pas encore à l'aise avec l'ensemble de ces notions, c'est pourquoi vous ne devez pas hésiter à tester les syntaxes vues et prendre le temps de vous familiariser avec ce nouveau vocabulaire. Mais malgré tous ces efforts, connaître la syntaxe de la programmation par objet n'est pas encore tout à fait suffisant. En effet, les concepts de la POO sont issus de méthodes et mécanismes de conception logicielle, et il est important de les connaître pour éviter les erreurs communes ou les fautes de conception pouvant parfois rendre vos classes non fonctionnelles.

Nous allons donc aborder quelques notions de conception courantes, malheureusement tout ceci restera limité puisqu'il faudrait plusieurs livres pour couvrir convenablement ce sujet !

Différents modèles de classes

On peut distinguer trois grandes familles de classes : les dialogues, les contrôles et les entités.

Les dialogues sont des classes représentant les interfaces communicantes. Dans une application client/serveur PHP, cette classe va généralement intercepter et analyser les paramètres passés dans l'URL de la page, les données entrées dans un formulaire par l'utilisateur ou même des données échangées entre deux serveurs (un fichier RSS par exemple).

Les contrôles sont des *classes outils*, qui ne contiennent généralement pas ou peu de propriétés. Ces classes regroupent un ensemble de méthodes représentant des actions, généralement liées au dialogue.

Enfin, les classes entités représentent généralement les données manipulées par l'application, les données persistantes, comme par exemple des données issues d'une base de données. Une entité possède généralement de nombreuses propriétés et des méthodes permettant de les manipuler.

Généralement il est intéressant de concevoir une interface de dialogue pour standardiser les méthodes

permettant d'accéder aux données que les objets dialogue vont analyser. Ainsi, l'objet contrôle pourra manipuler la ou les entités à utiliser à partir des données extraites du dialogue, sans avoir à changer le code si le formulaire utilisateur est modifié ou que l'adresse de la page change de format.

Les liens entre les objets

Les objets d'une application forment un système complexe au sein duquel ils communiquent. C'est pourquoi ils sont généralement liés les uns aux autres et s'envoient des messages sous différentes formes. Ces liens ne s'expriment pas toujours directement dans le code de définition d'une classe, mais apparaissent lorsqu'elle est manipulée.

Les exemples de liens que nous allons décrire à présent se retrouvent dans les méthodes de modélisation telles que UML (*Unified Modeling Language*), qui offre des méthodes très concrètes pour concevoir une application. Il est difficile de concevoir un logiciel sans ces outils.

La plupart des liens ont une direction et une cardinalité précisant combien d'objets participent à une relation.

L'association entre deux objets la plus simple et courante est une association *sémantique*, qui précise le sens du lien entre deux objets : ces liens représentent généralement des actions d'un objet sur un autre. Dans votre code, ceci se traduit généralement par l'utilisation d'une classe dans une autre.

L'agrégation est un lien fort qui permet de montrer qu'un objet est une composante d'un autre, sans pour autant lui appartenir. Les objets agrégés ont une raison d'exister de manière indépendante, mais l'objet agrégeant peut en avoir besoin pour fonctionner correctement. Par exemple, des objets `Personne` peuvent être agrégés par un objet `Entreprise` : l'entreprise fonctionne difficilement si personne n'y participe, pourtant, les personnes y travaillant vivent même sans être dans l'entreprise.

Dans votre code, votre classe `Entreprise` contiendra une propriété `$employes` qui sera un tableau de `Personne` (un tableau puisque dans notre cas plusieurs employés seront liées à l'entreprise). Les objets `Personne` seront créés en dehors de l'`Entreprise`, elle doit donc proposer des méthodes publiques permettant d'embaucher du personnel. Il est courant, pour permettre à l'objet `Personne` de communiquer avec l'objet `Entreprise` qu'il possède lui aussi un attribut `$employeur`.

La composition, enfin, est une agrégation forte, puisque le cycle de vie des objets participant à la relation est dépendant. Par exemple, des objets `Produit` sont des composant d'un objet `Entreprise` : ils n'ont pas de raison d'exister sans l'entreprise, qui les créé, les maintient, et décide de ne plus les commercialiser (donc de les détruire) : les objets `Produit` doivent être détruits si l'`Entreprise` est détruite, par ailleurs, l'entreprise à l'exclusivité sur ses Produits, il ne peut normalement pas y avoir d'agrégation (ou composition) entre `Produit` et un autre objet.

Listing 11. *Les classes Personne, Entreprise et Produit : exemples d'agrégation et composition*

```
class Personne
{
  private $id = 0;
  private $nom = '';
  private $employeur;

  public function _ _construct($id, $nom) {
    $this->id = $id;
    $this->nom = $nom;
  }

  public function setEmployeur(Entreprise $e) {
    $this->employeur = $e;
  }

  public function travaille() {
    if(!is _ null($this->employeur)) {
      $this->employeur->creerProduit(); // Envoie un
message
    }
  }

  public function getId() { return $this->id; }
  public function getNom() { return $this->nom; }
  public function getEmployeur() { return $this-
>employeur; }
}

class Entreprise
{
  private $produits;
  private $employes;

  public embaucher(Personne $p) {
    $this->employes[$p->getId()] = $p;
    $p->setEmployeur($this);
  }

  public licencier(Personne $p) {
    unset($this->employes[$p->getId()]);
    $p->setEmployeur(null); // Envoi d'un message
  }

  public function creerProduit() {
    $id = sizeof($this->produits)+1;
    $p = new Produit($id);
    $this->produits[$id] = $p;
    return $p;
  }

  public function arreterProduit(Produit $p) {
    unset($this->produits[$p->getId()]);
  }
}

class Produit
{
  private $id;

  public function _ _construct($id) {
    $this->id = $id;
  }

  public function getId() { return $this->id; }
}
```

Dans votre code, votre classe `Entreprise` contiendra une propriété `$employes` qui sera un tableau de `Personne` (un tableau puisque dans notre cas plusieurs employés seront liées à l'entreprise). Les objets `Personne` seront créés en dehors de l'`Entreprise`, elle doit donc proposer des méthodes publiques permettant d'embaucher du personnel. Il est courant, pour permettre à l'objet `Personne` de communiquer avec l'objet `Entreprise` qu'il possède lui aussi un attribut `$employeur`.

La composition, enfin, est une agrégation forte, puisque le cycle de vie des objets participant à la relation est dépendant. Par exemple, des objets `Produit` sont des composant d'un objet `Entreprise` : ils n'ont pas de raison d'exister sans l'entreprise, qui les créé, les maintient, et décide de ne plus les commercialiser (donc de les détruire) : les objets `Produit` doivent être détruits si l'`Entreprise` est détruite, par ailleurs, l'entreprise à l'exclusivité sur ses Produits, il ne peut normalement pas y avoir d'agrégation (ou composition) entre `Produit` et un autre objet.

Un exemple simple d'implémentation de ces concepts est présenté au Listing 11.

Les design pattern

Les *design patterns* (ou motifs de conception, ou encore masques) sont des architectures de classes courantes

Listing 12. *Présentation du pattern singleton*

```
class UnSingleton {
  private static $instance = null;
  final private function _ _construct($parametre) {}
  final private function _ _clone() {}
  final public static function getInstance($parametre =
null) {
    if(is_null(self::$instance))
      self::$instance = new _ _CLASS_ _($parametre);
    return self::$instance;
  }
}
```

Listing 13. *Un exemple d'adaptateur*

```
interface iSGBD
{
  public function connect($host, $user, $pass, $db);
  public function close();
}

class SGBD_Postgre implements iSGBD
{
  private $ressource = null;
  public function connect($host, $user, $pass, $db)
  {
    $this->ressource = pg_connect('host='.$host.'
user='.$user.' pass='.$pass.' dbname='.$db);
  }
  public function close()
  {
    pg_close($this->ressource);
  }
}

class SGBD_Mysql implemes iSGBD
{
  private $ressource = null;
  public function connect($host, $user, $pass, $db)
  {
    $this->ressource = mysql_connect($host, $user, $pass);
    mysql_select_db($db, $this->ressource);
  }
  public function close()
  {
    mysql_close($this->ressource);
  }
}

class SGBD_factory {
  public static function getSGBD()
  {
    // on peut aller chercher le SGBD à utiliser dans
un fichier de configuration
    $sgbd = lire_dans_config('sgbd_choisi');

    switch($sgbd) {
      case 'MySQL':
        $r = new SGBD_MySQL(); break;
      case 'PostgreSQL':
        $r = new SGBD_Postgre(); break;
      default:
        $r = null;
    }
    return $r;
  }
}
```

répondant à des besoins de conception. Ce sont des modèles connus des développeurs et certains sont très souvent utilisés. Il existe une infinité de design patterns, certains n'existent que dans certains langages, d'autres n'ont généralement pas à exister dans une application client/serveur en PHP.

Nous avons déjà vu un design pattern : la collection, celle-ci n'est pas toujours utile en PHP, puisque les tableaux sont très souples, dans d'autres langages, elle est généralement déjà implémentée. Nous pouvons voir quelques exemples de plus, très courants en PHP : le *singleton* et l'adaptateur (*adapter* en anglais).

Le singleton est une classe qui doit être instanciée pour fonctionner, mais qui ne doit l'être une seule fois dans toute l'exécution du programme. Pour créer un singleton, il faut pouvoir contrôler la construction et le clonage de l'objet en rendant les méthodes `__construct()` et `__clone()` privées et en proposant une méthode gérant cette instanciation (qui sera logiquement statique) et utilisera une variable de classe privée qui contient la seule instance autorisée. Le Listing 12 montrera l'implémentation de base d'un singleton.

La méthode `getInstance()` teste si l'instance existe déjà, si ce n'est pas le cas, elle la crée : le constructeur étant privé, il est toujours visible depuis la classe. Si des paramètres doivent être transmis au constructeur, alors la méthode `getInstance()` doit pouvoir les recevoir, mais elle doit pouvoir être appelée sans paramètre également.

L'adaptateur est utilisé pour uniformiser l'utilisation d'un outil quand plusieurs librairies concurrentes peuvent êtres utilisées. Par exemple, votre application PHP peut travailler avec plusieurs systèmes de gestion de bases de données (SGBD) : MySQL, PostgreSQL, SQLite.... Avant l'introduction de PDO, la connexion avec ces bases de données se faisait avec des extensions différentes. Par exemple, pour se connecter à la base, il fallait utiliser *mysql_connect()* pour MySQL et *pg_connect()* pour PostgreSQL. L'utilisation d'une de ces méthodes dans votre code le rend impossible à utiliser sur les deux SGBD sans modifications. L'adaptateur permet de supprimer ce problème.

Il consiste en la création d'une interface d'accès aux fonctions communes des deux SGBD qui sera implémentée par deux classes : l'une pour MySQL, l'autre pour PostgreSQL. Enfin, vous utiliserez une *fabrique* (qui est un design pattern à part entière) pour sélectionner le système à utiliser.

Dans certains cas, l'interface est remplacée par une classe abstraite. Je vous recommande, pour assurer la pérennité de votre développement, d'utiliser une interface et, si besoin, une classe abstraite implémentant cette interface.

Le Listing 13 présente un exemple d'adaptateur pour les fonctions de connexion à ces SGDB. Notez cependant que l'harmonisation de l'utilisation des SGBD est un problème complexe pour lequel il existe des réponses puissantes et approuvées par les développeur, telles que PDO, Propel ou Doctrine.

Les frameworks

Depuis plusieurs années, des frameworks pour PHP sont apparus. Ces outils ont pour but de rendre le développement plus fiable et plus rapide. La plupart de ces frameworks reposent sur des notions de programmation orientée objet comme le motif de conception MVC (Modèle, Vue, Contrôleur). Ils proposent un grand nombre de

fonctions, classes et méthodes de travail afin de limiter les tâches répétitives des développeurs. Par exemple, la gestion des bases de données et les actions liées les plus courantes (insertion, édition, sélection et suppression) peuvent être réalisées de manière sécurisée en quelques lignes de code.

Les trois framework PHP les plus connus sont Zend Framework, CakePHP et Symfony. Ils sont tous les trois distribués gratuitement sous des licences Open Source plus ou moins restrictives et peuvent être utilisés pour des applications à caractère commercial.

Zend Framework est développé par Zend (l'entreprise développant le moteur interne de PHP : Zend Engine) et supporté par IBM. Il est essentiellement basé sur un ensemble de composants couvrant de nombreux aspects et pratiques de programmation avec PHP. Il est très souvent utilisé en entreprise car les briques logicielles qui le composent sont faiblement couplées (elles peuvent être utilisées plus ou moins indépendamment les unes des autres).

CakePHP est selon ses créateurs, un équivalent en PHP du célèbre Ruby On Rails (pour le langage Ruby). Enfin, Symfony, né au sein d'une agence française, propose de nombreux outils, tels que la génération de code permettant de mettre en place une application rapidement, pour laquelle le développeur ne se concentrera que sur du développement spécifique. Il est utilisé, entre autres, par Yahoo! et DailyMotion. C'est un framework *full-stack* : c'est à dire que les contrôleurs et de nombreux comportements (comme la gestion des sessions de l'utilisateur) sont gérés directement par le framework.

L'apprentissage d'un framework nécessite du temps et de l'investissement de la part du développeur, cependant, les gains sont nombreux une fois que l'outil est maîtrisé : une économie de temps et de moyens mais aussi un ensemble de conventions facilitant la maintenance et garantissant le bon fonctionnement de l'application. Ceux-ci sont essentiellement utilisés dans des projets dont la taille est importante. En effet, sur des projets de taille raisonnable, l'usage d'un tel outil peut s'avérer être une perte de temps.

Du bon usage de la POO

Il me reste un dernier point à aborder avant de vous lâcher dans la nature : le développement par objet demande plus de ressources à l'ordinateur lors de l'exécution que du code fonctionnel, c'est pourquoi vous devez savoir quand utiliser la POO et quand l'éviter en PHP !

Il existe plusieurs écoles : certains partisans du *tout-objet* vous diront que le code fonctionnel se limite à l'instanciation d'un objet et l'appel d'une de ses méthodes : le point d'entrée du programme est donc un fichier `index.php` contenant *grosso-modo* ces quelques lignes :

```
<?php
require('sources/controler.php');
$mainObjet = new Controler();
$mainObjet->start();
```

Ce choix doit être motivé et répondre à des besoins précis de conception. C'est l'architecture choisie par le framework *symfony*, elle est justifiée par le besoin de contrôler l'ensemble de l'exécution du script en interne.

Les concepteurs de PHP, quand à eux, recommandent l'utilisation de la programmation par objet pour le développement de la *couche métier* de votre application, c'est à dire les outils qui constituent le cœur de votre logiciel. Le déroulement global de l'application doit donc se faire dans le *script courant*. C'est l'architecture choisie pour des applications comme *spip* ou *dotclear*.

Cette architecture à l'avantage de permettre de contrôler plus facilement les ressources employées par le script : vous n'utiliserez en effet probablement pas toutes les méthodes de la totalité des objets que vous manipulerez, et limiter le nombre de classes que PHP doit analyser à chaque exécution améliore nettement les performances. Par ailleurs, l'instanciation d'un objet a un coût en mémoire et en temps non négligeable, tout comme l'appel en série de méthodes qui multiplient les passages à la pile d'exécution.

Conclusion

J'espère vous avoir mis toutes les cartes en main pour vous permettre de vous lancer dans le monde de la programmation orientée objet. C'est un style de programmation puissant et complexe, mais qui devient rapidement un allié de taille une fois maîtrisé. N'ayez donc pas peur de passer du temps à le travailler.

Sur le réseau

- *http://fr.php.net/manual/fr/language.oop5.php* – Le chapitre sur le paradigme objet en PHP5 du manuel de référence de PHP est certainement le plus complet ; n'hésitez pas à le consulter,
- *http://php.developpez.com/cours/?page=langage#poo* – Les ressources sur la PHP et la POO du site developpez.com peuvent apporter des réponses précises à vos questions.

Bibliographie

- *Best Practices PHP5* (Guillaume Ponçon) aux éditions Eyrolles – L'un des livres de référence en France sur le développement de logiciels en PHP5, celui-ci complète parfaitement l'excellent *PHP5 avancé*.

MARTIN RICHARD

L'auteur est étudiant en informatique et collabore à de nombreux projets d'applications pour le web ou de communication. Ses travaux et activités sont résumées sur son site internet www.martiusweb.net où il publie articles et cours sur ses découvertes.

Manipuler les sessions avec PHP

La gestion des niveaux de privilèges d'un internaute dans un wiki ou un CMS et la réalisation d'un panier de commande, sont deux exemples d'applications rendues possibles par le mécanisme des sessions. Dans cet article vous apprendrez à créer et manipuler des sessions depuis un script PHP.

Cet article explique :

- Ce que sont les sessions.
- Comment établir une session et manipuler des variables de session.

Ce qu'il faut savoir :

- Vous devez connaître les bases du langage PHP.

La gestion d'un panier de commande et la navigation dans une application web à accès restreint gérant plusieurs niveaux de privilèges, ne peuvent fonctionner que si le serveur reconnaît l'internaute lorsque celui-ci lui demande une URL. Les clients (navigateurs) et les serveurs web communiquent par le biais du protocole de transport HTTP (*HyperText Transfer Protocol*). Aucune information n'est conservée entre deux connexions. Lorsque le serveur reçoit une demande de ressource, les seules informations dont il dispose sont l'adresse IP qui a émis la requête, et la requête HTTP elle-même (ressource demandée, informations sur le navigateur, et éventuellement données et méta-informations sur ces données). L'adresse IP ne permet pas d'identifier un utilisateur unique car plusieurs clients peuvent partager une même adresse IP (proxy, translation d'adresse). De plus un ordinateur peut changer d'adresse IP entre deux visites à un site web, voire même au cours d'une même navigation (DHCP).

Le seul moyen pour un serveur de reconnaître un internaute est que le navigateur transmette une information dans la requête HTTP. L'information transmise est un jeton de session, c'est-à-dire un identifiant unique qui permettra à l'application de reconnaître l'internaute pendant toute la durée de sa session. La gestion de ce jeton de session peut être réalisée par divers procédés : placer le jeton dans un paramètre de l'URL, transmettre le jeton par le biais d'un champ caché de formulaire HTML, ou en utilisant un cookie. La dernière solution est la plus simple à mettre en place, la plus efficace et la plus sûre.

Dans un précédent numéro de ce même magazine, vous avez appris le principe et la manipulation des cookies depuis un script PHP. Dans cet article, vous n'aurez pas à générer vous même les cookies car PHP intègre en standard, depuis la version 4, un mécanisme de session qui se charge de l'envoi automatique des cookies, du stockage des informations de session sur le serveur et de leur manipulation. Bien que le mécanisme des sessions soit très simple d'utilisation, il est utile d'avoir des connaissances sur les cookies pour redéfinir les paramètres utilisés par défaut (durée de vie, chemin, transfert sécurisé) et comprendre le fonctionnement des sessions.

Vous allez apprendre dans cet article à régler les propriétés des sessions, à créer et restaurer une session, à manipuler des données dans des variables dont la portée est celle des scripts de l'application, et à détruire une session depuis un script PHP.

Régler les propriétés

Lors de la création d'une session, PHP génère un jeton de session, c'est-à-dire un identifiant unique attribué et envoyé à un internaute. Le navigateur de l'internaute transmet ensuite ce jeton lors de chaque requête. C'est ce qui permet de reconnaître un internaute et de placer dans le bon panier de commande les différents articles commandés pendant toute la durée de la navigation. Le suivi de session peut être réalisé en transmettant

ce jeton par cookie, ou dans la partie *querystring* de l'URL.

Le comportement des sessions dépend d'un ensemble de directives du fichier de configuration PHP, présentées dans cette partie. Pour obtenir les valeurs des directives PHP, vous pouvez directement consulter la section `[Session]` du fichier *php.ini* si vous y avez accès. Dans le cas contraire, exécutez le script PHP `<?php phpinfo(); ?>`. La fonction `phpinfo` retourne des informations sur le support des sessions et indique les valeurs des directives. Pour des raisons de sécurité cette fonction est souvent désactivée sur les serveurs en production. Si vous n'avez accès ni au fichier de configuration, ni à la fonction `phpinfo`, vous pouvez obtenir la valeur d'une directive en la passant en argument à la fonction `ini_get`.

Nom du jeton

Le nom du jeton de session par défaut est `PHPSESSID`, il est défini dans la directive `session.name` du fichier de configuration PHP. La fonction `session_name`, appelée sans argument, retourne le nom du jeton pour la session courante.

Il est possible de modifier le nom du jeton dans le fichier de configuration, ou en donnant le nom de jeton en argument à la fonction `session_name`. Dans le premier cas, les jetons de toutes les applications web du serveur prendront le nom placé dans la directive. Dans le second cas, la modification du nom de jeton est locale au script. Il faudra donc appeler la fonction `session_name` dans tous les scripts de l'application web, avant de démarrer la session.

Valeur du jeton

La valeur du jeton de session est une chaîne de caractères générée aléatoirement par PHP, par exemple : `eh26jtuab7g1fsu1mthi1gbs55`. La fonction `session_id`, appelée sans argument, retourne la valeur du jeton pour la session courante.

La valeur du jeton de session peut être modifiée en passant en paramètre la valeur souhaitée à la fonction `session_id`. Cette fonction doit être appelée avant l'ouverture de la session, dans chaque script de l'application web. Il est cependant fortement déconseillé pour des raisons de sécurité de modifier la valeur du jeton.

Transmission du jeton

Le moyen de transmission de l'identifiant de session est dépendant des valeurs de directives du fichier *php.ini*. Lorsque la directive `session.use_cookies` a la valeur 1 (valeur par défaut), l'application web transmet le jeton de session dans un cookie. Le cookie placé sur le disque dur du client a pour nom `PHPSESSID` et pour valeur l'identifiant de session, par exemple :

```
PHPDESSID= eh26jtuab7g1fsu1mthi1gbs55.
```

Si le navigateur de l'internaute n'accepte pas les cookies, PHP ne peut donc pas placer un cookie de session chez le client. Il utilise alors automatiquement une session basée sur l'URL, à condition que la directive `session.use_trans_sid` soit activée dans le fichier de configuration `php.ini`. Pour des raisons de sécurité, il est préférable de ne pas utiliser cette fonctionnalité, les sessions ne devraient reposer que sur une transmission par cookie. Il est conseillé de laisser cette directive désactivée (valeur 0) et de donner la valeur 1 à la directive `session.use_only_cookies`, qui indique à PHP de ne jamais accepter de jeton de session transmis dans la partie *querystring* de l'URL. Un prochain article de ce magazine expliquera en détail les réglages de sécurité des sessions.

Si vous n'avez pas accès au fichier de configuration PHP, vous pouvez régler localement les valeurs des directives soit dans un fichier *.htaccess* si vous avez les droits nécessaires, soit avec la fonction `ini_set`. Cette fonction prend en premier argument le nom de la directive à modifier (chaîne de caractères) et en second argument la valeur à lui affecter. Dans ce cas il est recommandé de créer un fichier PHP contenant les réglages des directives et démarrant la session et de l'inclure en première ligne de chaque script de l'application web.

Cookie de session

La validité, le chemin, le domaine et la transmission sécurisée ou non du cookie de session sont définis dans le fichier de configuration *php.ini*. Ces propriétés ont été décrites dans l'article intitulé *Manipuler les cookies avec PHP* du numéro 4/2010 de PHPSolutions, pour plus de détails veuillez vous référer à cet article disponible en ligne : *http://phpsolmag.org/fr/magazine/1068-ajax-avec-jquery-et-zend-framework*. La directive `session.cookie_lifetime` fixe la durée de validité du cookie, elle a la valeur `0` par défaut. Les directives `session.cookie_domain` et `session.cookie_path` précisent respectivement le domaine (vide par défaut) et le chemin de validité du cookie (`/` par défaut). Enfin la directive `session.cookie_secure` indique si le cookie doit être transmis uniquement lorsque la communication est cryptée (0 par défaut). Par défaut le cookie est donc valable sur tout le domaine et pour tous les scripts du serveur, il est détruit lorsque l'internaute quitte le navigateur, et est transmis par HTTP ou HTTPS.

Ces propriétés peuvent être modifiées dans le fichier *php.ini* ou directement dans un script PHP en utilisant la fonction `session_set_cookie_params`. Dans le premier cas elles seront appliquées à toutes les applications web du serveur qui reposent sur les sessions. Dans le second cas, seul le cookie de session de l'application courante sera concerné. La fonction `session_set_cookie_params` prend en arguments la durée de

Listing 1. *init_session.php*

```
<?php

// session par cookies uniquement
ini_set("session.use_cookies", 1);
ini_set("session.use_only_cookies", 1);
// validite : 1 h (3600s)
session_set_cookie_params(3600);
// modifier le nom du jeton
session_name("test");
session_start();
echo "Votre navigation est suivie par la session ";
echo session_name(), "=", session_id();
?>
```

Listing 2. *memoriser_var.php*

```
<?php

// creer ou restaurer la session
session_start();
// recuperer le parametre passe dans l'URL
$nb = isset($_GET['nb']) ? (int)$_GET['nb'] : 0;
// stocker les donnees dans des variables de session
$_SESSION['quantite'] = $nb;
$_SESSION['ref'] = 'A-06-18';
echo 'Variable quantite memorisee';
?>
```

Listing 3. *informations.php*

```
<?php

// restaurer la session
session_start();
// afficher les variables de session
foreach ($_SESSION as $cle=>$valeur){
    echo "Variable de session $cle, contenu =
$valeur<br>";
}
?>
```

Listing 4. *deconnexion.php*

```
<?php

// restaurer la session
session_start();
// supprimer le cookie de session
setcookie(session_name(), '', time()-3600);
// supprimer les variables de session
$_SESSION = array();
// fermer la session
session_destroy();
echo '<br>Deconnexion reussie';
?>
```

vie du cookie (entier), le chemin (chaîne de caractères), le domaine (chaîne de caractères) et deux booléens dédiés à la sécurité. L'appel à cette fonction doit être réalisé avant l'ouverture de la session.

Dans l'exemple du Listing 1, les directives `session.use_cookies` et `session.use_only_cookies` sont activées localement avant d'ouvrir la session, indiquant au script courant d'utiliser les cookies et uniquement les cookies pour la transmission du jeton de session. Dans les autres Listings de cet article, ces deux directives ainsi que le nom du jeton (test) ont été directement modifiés dans le *php.ini*, afin de simplifier les codes. L'exemple du Listing 1 modifie également la durée de validité du cookie de session (la session est détruite au bout d'une heure) et le nom du jeton de session (valeur `test`), puis affiche le nom et la valeur de l'identifiant de session.

Ouvrir une session

La fonction `session_start` crée ou restaure une session, tout script qui utilise les sessions doit appeler cette fonction. Elle ne prend aucun argument et retourne un booléen. Lors de la création cette fonction génère un jeton de session. Dans cet article, le suivi de session est réalisé en transmettant le jeton de session uniquement par cookie. Dans l'exemple du Listing 1, un cookie dont le nom est `test` et la valeur est définie aléatoirement, est envoyé au navigateur.

Selon la configuration du serveur les sessions peuvent être démarrées automatiquement, sans faire appel à la fonction `session_start`. C'est le cas si la directive `session.auto_start` du fichier de configuration *php.ini* a la valeur `1`.

Stocker des données en session

Des informations peuvent être mémorisées dans des variables dont la portée est celle de l'application web. C'est-à-dire que les valeurs stockées dans ces variables pourront être lues et modifiées par tout script faisant partie de l'application. Cette partie explique comment créer et utiliser ces variables de session.

Créer une variable de session

La variable super-globale `$_SESSION`, disponible depuis PHP 4.1, permet de créer une ou plusieurs variables de session. C'est un tableau associatif dont les clés sont les noms des variables de session, et les valeurs le contenu de ces variables. Tout script qui stocke une donnée en session doit préalablement créer ou restaurer une session en utilisant la fonction `session_start`.

Le script *memoriser_var.php* du Listing 2 crée une première variable de session dont le nom est `quantite` et la valeur est celle de la donnée `nb` reçue par la méthode GET. Par exemple, *memoriser_var.php?nb=8* affectera la valeur 8 à `quantite` dans le tableau `$_SESSION`. Une seconde variable est placée en session (clé `ref`, valeur `A-06-18`).

Afin de mémoriser les variables de session pendant toute la durée de navigation d'un internaute, ces variables sont stockées dans un fichier sur le disque dur du serveur web. Ce comportement est fixé par la directive `session.save_handler` dans le fichier de configuration *php.ini*. Lorsque cette directive prend la valeur `files` (valeur par défaut), les variables sont stockées dans un fichier. L'application web gérant simultanément plusieurs utilisateurs, un fichier est créé par session. Le nom du fichier de session contient la valeur du jeton de session, préfixé par `sess`. Ces fichiers sont généralement stockés dans un répertoire temporaire, dont l'emplacement est défini par la directive `session.save_path` du fichier *php.ini*.

Dans l'exemple ci-avant, l'identifiant de session a la valeur `eh26jtuab7g1fsu1mthi1gbs55`, le fichier de session, stocké dans le répertoire `/tmp`, est donc nommé `sess_eh26jtuab7g1fsu1mthi1gbs55`. Ce fichier contient les noms des variables de session et leurs valeurs, ces informations sont sérialisées avant d'être stockées, c'est-à-dire que des méta-informations sont ajoutées (type et taille des données). Par exemple, le fichier de session de l'exemple du Listing 2 contient les données suivantes :

```
quantite|i:8;ref|s:7:"A-06-18";
```

Les données de session sont stockées dans le fichier sous la forme `nom_variable|type:valeur;`. Le type de la variable de session est représenté par un caractère : `s` = string, `b` = boolean, `i` = int, `a` = array. Lorsque le type est une chaîne de caractères ou un tableau, il est de la forme `caractere_type:taille`. La taille indique le nombre de caractères dans la chaîne, ou le nombre de cases du tableau. Dans cet exemple, deux variables de session sont stockées, `quantite` et `ref`. La variable de session `quantite` est de type entier (`i`) et a la valeur 8. La variable `ref` est une chaîne de caractères (`s`), de 7 caractères, dont la valeur est la chaîne indiquée entre guillemets (`A-06-18`).

Lire les données en session

La fonction `session_start`, présentée précédemment, crée ou restaure une session. Lorsqu'une session existe, l'appel à cette fonction rend disponible les variables de session pour le script courant. A partir du jeton de session envoyé par l'utilisateur, PHP récupère les informations stockées dans le fichier de session adéquat et les place dans le tableau `$_SESSION`. Pour accéder au contenu de la variable de session `quantite`, il suffit d'écrire `$_SESSION['quantite']`. Afin d'éviter des erreurs ou avertissements PHP, vous devez vérifier l'existence d'une variable de session avant de l'utiliser avec les fonctions `isset` ou `array_key_exists`, qui renvoient toutes les deux le booléen `true` si la variable de session est définie.

Le Listing 3 affiche le nom et le contenu de toutes les variables de session pour la session courante. Ce script suppose que les données stockées en session sont scalaires. Si vous mettez en session des variables de type tableau, il n'affichera pas leur contenu. Une manière simple de contrôler toutes les données placées en session est d'utiliser la fonction `print_r`, qui affiche automatiquement tout le contenu d'un tableau :

```
<?php
session_start();
print_r($_SESSION);
?>
```

L'exécution de ce code produit le résultat suivant pour l'exemple pris dans la section précédente :

```
Array (
   quantite => 8,
    ref => A-06-18
)
```

Modifier les données

La modification d'une variable de session est effectuée simplement en affectant une nouvelle valeur à la clé correspondante dans le tableau `$_SESSION`. Cette opération doit être réalisée après avoir restauré la session avec la fonction `session_start`.

Lorsque le script `memoriser_var.php` du Listing 2 est exécuté une seconde fois avec `nb=12` dans la chaîne de requête de l'URL, le script affecte un nouveau contenu à la variable `quantite` (la valeur `8` est remplacée par la valeur `12`), et affecte à nouveau la valeur `A-06-18` à la variable `ref`.

Supprimer une variable de session

Pour supprimer une variable de session il faut utiliser la fonction `unset`. Celle-ci prend en paramètre la variable à supprimer. L'instruction ci-après supprime la variable `ref` du tableau `$_SESSION` :

```
unset($_SESSION['ref']);
```

Fermer une session

Lorsqu'un utilisateur se déconnecte d'une application web reposant sur les sessions, il faut détruire la session pour des raisons de sécurité. Cette opération est réalisée en plusieurs étapes :

- supprimer le cookie de session en envoyant un nouveau cookie de session dont la valeur est vide et dont la date d'expiration est dépassée,
- supprimer les variables de session en affectant un tableau vide à la variable super-globale `$_SESSION`,
- supprimer les données liées à la session en utilisant la fonction `session_destroy`.

Ces opérations sont réalisées dans le Listing 4. Lorsque le script de déconnexion est exécuté, il restaure la session, puis la détruit. Il est important de ne pas oublier de restaurer la session avant de la supprimer, sinon les données de session et le cookie de session ne seront pas détruits. Si vous exécutez le script *deconnexion.php*, puis que vous exécutez le script *informations.php*, vous obtiendrez l'affichage d'un tableau vide.

Erreurs fréquentes

Lors de la manipulation de sessions, il peut arriver que les données de session ne soient pas disponibles

dans un script. Si c'est le cas suivez les étapes ci-dessous afin de déterminer la source du problème et de la corriger.

Vérifiez tout d'abord que vous avez un cookie de session dans votre navigateur, si ce n'est pas le cas vérifiez que votre navigateur accepte les cookies. S'il les accepte, assurez-vous que le script qui crée les variables de session appelle la fonction `session_start`, et que cette instruction est placée avant toute instruction envoyant des données vers la sortie standard. Si la fonction `session_start` est absente, le cookie de session n'est pas envoyé au navigateur. Si elle est présente mais située après l'envoi du début du corps de la requête HTTP, le cookie de session ne peut plus être envoyé car il doit être placé dans l'en-tête de la requête HTTP, la session ne peut donc pas fonctionner. En fonction des réglages du niveau d'erreur dans le fichier *php.ini*, vous ne serez pas avertis de l'échec d'envoi du cookie. Tant que vous n'aurez pas obtenu un cookie de session, votre application ne pourra pas fonctionner correctement.

Si vous avez un cookie de session dans votre navigateur, mais que les variables de session ne sont pas disponibles dans le script (tableau `$_SESSION` vide), le problème peut venir comme précédemment de l'oubli de l'appel à la fonction `session_start`. Si vous avez le droit d'accéder aux fichiers du serveur web, vérifiez le contenu du fichier de session correspondant à l'identifiant de session de votre cookie de session. Si le fichier ne contient pas les variables de session, vérifiez si le script qui affecte des données dans ces variables fait un appel à la fonction `session_start`. Vous pouvez avoir un cookie de session, placé par un script de l'application, mais aucune variable stockée car le stockage des variables n'est pas réalisé par le script qui a créé la session. C'est souvent le cas lorsqu'un premier script réalise l'authentification de l'internaute, et un autre script crée une variable.

Si le fichier de session sur le serveur contient les variables de session, mais que ces variables ne sont pas disponibles dans le script, vérifiez si votre script restaure la session avec la fonction `session_start`.

Un mauvais chemin de cookie peut également être la source des problèmes cités précédemment. Vérifiez le chemin du cookie dans les informations fournies par le navigateur et assurez-vous que ce chemin correspond à celui du script qui crée les variables ou qui souhaite accéder à leur contenu. Si le chemin ne correspond pas, le cookie n'est pas envoyé au script et la restauration de session n'est pas réalisée. Une manière de vérifier que le cookie est bien envoyé par le navigateur est d'utiliser l'onglet *Net* du *plugin Firebug* de *Firefox*, ou d'utiliser le *plugin TamperData* qui permet d'intercepter la requête HTTP et de vérifier les données envoyées par le navigateur au serveur web. Si le cookie de session n'apparaît pas dans l'en-tête de la requête HTTP, alors le serveur n'a aucun moyen de faire le lien entre les demandes de l'internaute et les données ne seront pas placées en session, ou ne pourront pas être lues.

Si vous avez des valeurs inattendues dans les variables de session ou que des données manquent, le problème peut être dû à un conflit entre deux applications web. Lorsque deux applications sur un même serveur ont un cookie de session valable sur tout le serveur (chemin `/`), alors un utilisateur qui a accès aux deux applications aura un cookie commun pour ces deux applications. Les variables de session des deux applications seront donc stockées dans le même fichier de session sur le serveur. Ceci peut poser des problèmes d'écrasement de valeurs de variables, et d'accès aux variables (destruction du fichier de session lors de la déconnexion d'une des deux applications, remplacement de valeurs).

Sur Internet

- *http://php.net/manual/fr/book.session.php* – Manuel de la gestion des sessions sur le site officiel de PHP.

Conclusion

Vous avez appris les bases de la manipulation des sessions. Vous savez à présent démarrer et restaurer une session, créer et utiliser des variables de session et supprimer une session et ses données. Les sessions permettent notamment de réaliser des paniers de commande et des applications web gérant plusieurs niveaux de privilèges. Le mécanisme des sessions reposant sur un identifiant unique transmis par un cookie de l'utilisateur, il est primordial de connaître les problèmes de sécurité posés par l'utilisation de cet identifiant, et d'apprendre à sécuriser les applications qui les utilisent. La sécurité des sessions fera l'objet d'un prochain article dans ce magazine.

CÉCILE ODERO, MAGALI CONTENSIN

Cécile Odero est spécialisée dans la conception et le développement d'applications web en PHP. Elle est développeur web freelance.
Contact : cecile.odero@gmail.com
Magali Contensin, auteur du livre Bases de données et Internet avec PHP et MySQL, est chef de projet en développement d'applications au CNRS. Elle enseigne depuis plus de dix ans le développement d'applications web à l'Université.
Contact : http://magali.contensin.online.fr

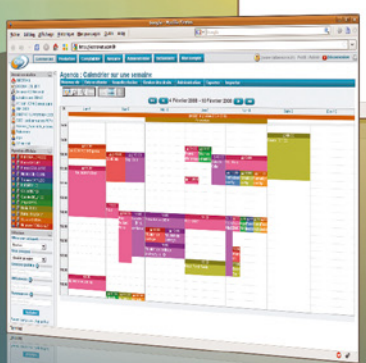
LINAGORA

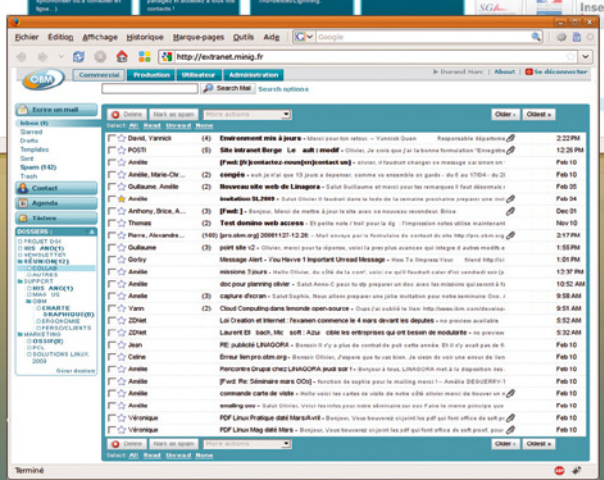
présente

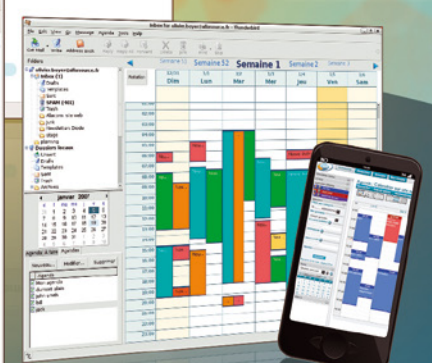
OBM Online

LA MESSAGERIE COLLABORATIVE OPEN SOURCE HÉBERGÉE !

NEW







http://online.obm.org

WEB Gazelle

Génialement Simple !

Cet homme
essaie de mettre son
site Internet à jour

Cet homme
a mis son site
Internet à jour avec
WebGazelle CMS 2.0

WebGazelle CMS 2.0 rencontre l'AJAX

www.webgazelle.net

Webgazelle.net, une marque de COGNIX systems