Formation

Smile
OPEN SOURCE SOLUTIONS

Formation eZ Publish

Techniques de cache

Smile
OPEN SOURCE SOLUTIONS

Plan

Formation

- Introduction
- Les types de cache eZ Publish
- Caches liés aux contenus
  - Caches de contenus
  - Les blocs de cache
- Structure des pages
- Cache en mémoire
- Le stale cache
- Le mode cluster
- Le cache statique
- Reverse Proxy

Smile OPEN SOURCE SOLUTIONS

→ Un sujet relativement complexe

- Beaucoup de spécificités eZ Publish
- Vocabulaire précis et spécifique
- Ambiguïtés liées aux traductions (anglais / français)

→ Un point essentiel pour les performances

- eZ Publish génère beaucoup de cache
- Il est indispensable de maîtriser la chaîne complète

La clé

→ Anticipation **ET** Réalisation

Les moyens

→ Connaître les objectifs du site (audience, métier)

→ Connaître la théorie

→ Identifier les problématiques de cache au plus tôt (non cachages, bloc dynamiques, …)

Les types de cache eZ Publish

Formation

- **Cache local**

  configuration, identifiants, traductions, surcharges, rôles, images ...

- **Cache de gabarits**

  Compilation des gabarits
  → Directive TemplateCompile

- **Caches liés aux contenus**
  - **Cache de contenu**
    ou View cache / Cache de vue
  - **Blocs de cache**
    ou cache-blocks

  → Politiques de cache personnalisées
  → Défini dans les gabarits
  → Directive TemplateCache

- **Caches en mémoire**

Smile OPEN SOURCE SOLUTIONS

Cache local

Formation

Niveau intégralement **géré par l'outil**.

Permet d'améliorer la performance en fournissant une image exploitable directement.

- Configuration ini

  Par défaut il est régénéré à la modification d'un fichier, ce comportement peut être modifié par positionnement de la valeur **EZP_INI_FILEMTIME_CHECK** dans le fichier *config.php* introduit en PHP 4.1.

- Fichiers de traductions

  → Longs à régénérer

- Variations d'images

  Générées à la **publication**.

- Mappings, gestion des rôles

- ...

Smile OPEN SOURCE SOLUTIONS

Cache de contenu

Formation

- **Essentiel pour de bonnes performances**
  - Activé par défaut (ViewCaching)
    Aucune raison d'être désactivé en production

    Le paramétrage s'effectue dans le fichier site.ini
    section [ContentSettings]
- **Stocke le résultat de la visualisation d'un contenu**

  → Contenu de la variable **$module_result.content**

  → Calculé à la première visualisation du contenu

  → Stockage brut
  (ne sera pas réinterprété lorsqu'il sera resservi)
- Ne s'applique qu'à la **visualisation**,
  vue « **content/view** »
  - Ne s'applique pas aux autres modules / vues
    (user/login, user/register, content/draft, ...)
  - Ne s'applique qu'aux modes de visualisation définis par la variable **CachedViewModes**
    par défaut : « full;sitemap;pdf »

Smile OPEN SOURCE SOLUTIONS

Cache de contenu

Formation

- **Les paramètres d'identification des caches**
  - Le **contexte** de visualisation (le noeud, le mode)
  - La **collection de droits** associée à l'utilisateur
    Ses « rôles », un cache par profil sera donc généré.
  - Les **paramètres de vues**
    Le contenu de la variable $view_parameters

    Optionnellement
  - Les préférences utilisateur
    via la variable CachedViewPreferences[]
- Possibilité de personnaliser ces paramètres, par noeud

  ViewCacheTweaks[<node_id>]=<setting>
- Attention, **$_GET**, **$_POST** ou **$_SESSION ne sont pas** des paramètres **valides**.

  → Il faut utiliser les **paramètres de vue**
- La politique de régénération des caches à la publication peut être personnalisée avec les fonctionnalités de **SmartCache**

  → Le paramétrage s'effectue dans le fichier *viewcache.ini*

Smile OPEN SOURCE SOLUTIONS

Cache de contenu

Formation

- Lors de la **modification d'un objet** (publication), **le cache de contenu est vidé** pour
  - **Le noeud courant**
  - **Ses parents directs**
  - Les objets en **relation inverse**
    (qui ont une relation vers l'objet courant)

    → **Attention**, seules les relations au niveau objet sont actives par défaut *(relations simples ou embedded)*.
    Les relations de type *link* ou *attribute* non actives par défaut.

    ClearRelationTypes[]
    Dans la section [ViewCacheSettings] du fichier viewcache.ini
  - Les objets qui ont un **mot-clé commun** (attribut *ezkeyword*)
- Possibilité de modifier le comportement sur un mode de visualisation

  → ComplexDisplayViewModes (sitemap)

Smile OPEN SOURCE SOLUTIONS

Cache de contenu

Formation

- Le cache de contenu est **spécifique à un siteaccess**

  → La variable *RelatedSiteAccessList[]* permet de **lier les caches de contenu** entre les sites partageant une même base de données.

- **Le cache de contenu n'expire pas naturellement**

  Si **nécessaire** il doit être explicitement supprimé :

  - Directive TTL
    {set-block scope=root variable=cache_ttl}XXX{/set-block}
    ou définition dédiée (tweak)
  - Script de vidange ciblé (cronjob)
    → Plus élégant

- Possibilité de pré-générer certains caches lors de la contribution
  - Par *siteaccess*
  - Par *utilisateur*

  Configuration ***PreviewCache***

  → Peu utilisé, quelques anomalies gênantes (traductions, …)
  → Ralentit la contribution (temps de génération)

Variable persistante

Formation

- Utilisation possible d'une variable persistante

  $persistent_variable dans les gabarits de vue

  La valeur sera rendue disponible dans le layout

  $module_result.content_info.persistent_variable

  → Outrepasse le cache de vue, pratique pour faire transiter des **informations structurantes**

Smile OPEN SOURCE SOLUTIONS

Blocs de cache

Formation

- **Notion de « cache-block »**
- **Principal outil d'optimisation** d'un site
- Prend la forme d'une **instruction** à intégrer au gabarit
- Le bloc définit les **variantes**, composées par **la clé** du bloc

  → La clé constitue **l'intelligence**
- Pour chaque affichage du bloc, l'existence préalable d'un cache valide est vérifiée, il est régénéré si nécessaire puis servi.

  → Il reste moins performant que le cache de vue

  → Il subsiste un coût de calcul,
  il est donc à réserver aux blocs de code qui constituent des traitement conséquents (requêtes, ...).

  → Il ne faut pas tomber dans l'excès inverse,
  L'abus de blocs peut nuire gravement aux performances

Smile OPEN SOURCE SOLUTIONS

Blocs de cache

- *En pratique*

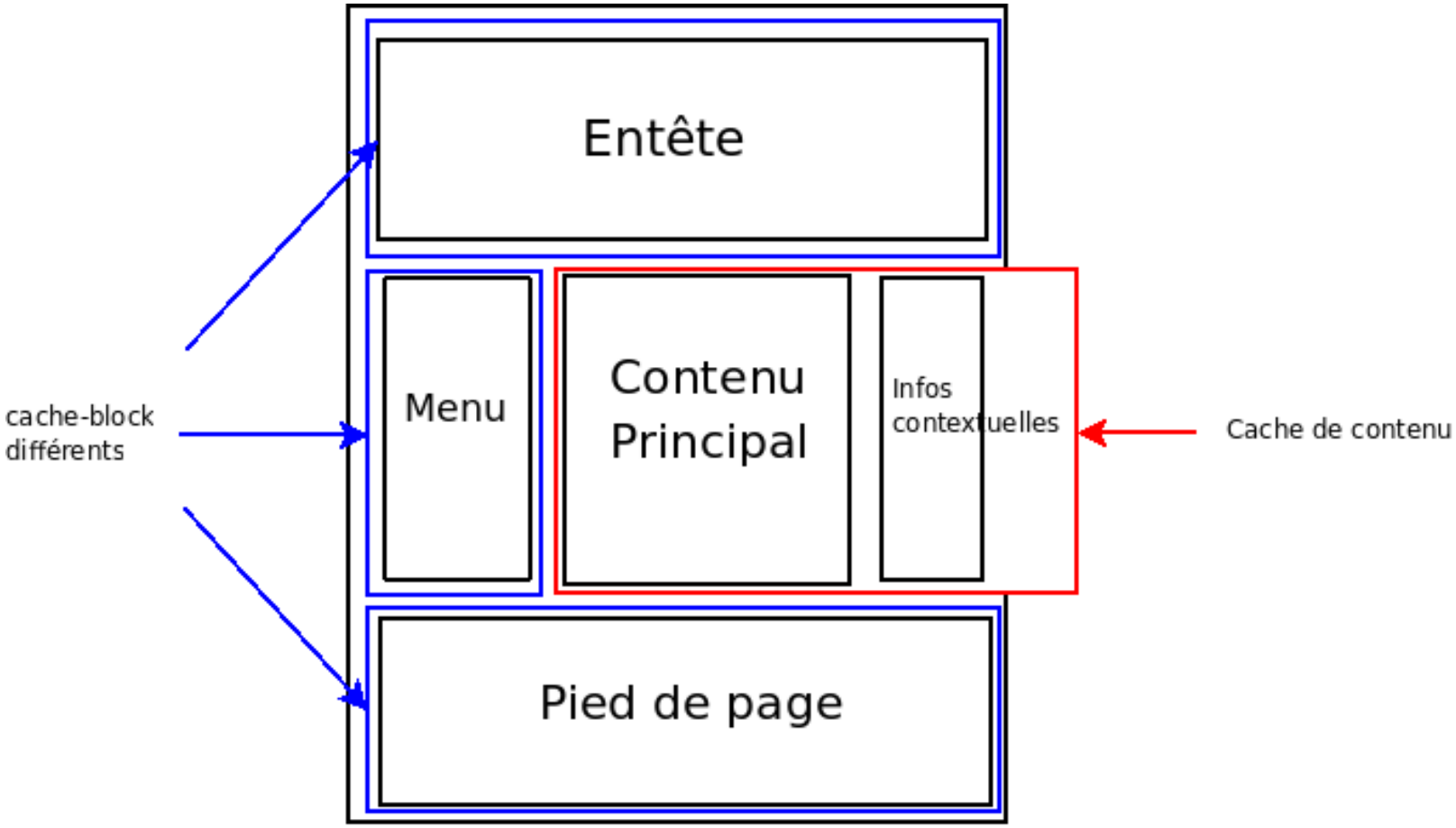
  {cache-block keys=$key expiry=$ttl subtree_expiry=/a/node/path ignore_content_expiry}

- **La clé**
  - Elle peut être une chaîne ou un tableau.
  - Pour avoir un bon cache-block, il faut **limiter le nombre de variantes** du bloc de cache, tout en couvrant l'ensemble des besoins fonctionnels.
  - Elle peut être parfois très complexe... un commentaire est toujours le bienvenu.

- **L'expiration** du bloc
  - Expire à **chaque publication d'un contenu**, sauf mention contraire (ignore_content_expiry)
  - Définie selon un **TTL** (en secondes)
    La valeur par défaut est de 2h (7200)
  - Expire à la modification d'un noeud ou une arborescence définie (subtree_expiry)

Structuration d'une page

Formation



Les métriques

→ Moins de **10 requêtes** (sql) par page, tous caches générés

Smile OPEN SOURCE SOLUTIONS

Caches en mémoire

Formation

- Évite de charger plusieurs fois les mêmes objets depuis la base de données.

  Utilisation d'une variable globale
  $eZContentObjectContentObjectCache[$id]

- Sa durée de vie limitée est limitée à l'**exécution de la page** (processus)

- Le cache mémoire peut engendrer des comportements inattendus ou des **problèmes de dépassement de mémoire**, notamment lors des traitements lourds (imports, scripts, ...)

  → La méthode **eZContentObject::clearCache**([$id]) permet de vider un cache.
  C'est la technique préconisée sur les modules ou scripts lourds

Smile OPEN SOURCE SOLUTIONS

Le moteur de stockage

Formation

- Permet **l'extensibilité** de l'application

  → Multiplier les serveurs applicatifs
- Notion de **FileHandler**
  - Implémentation générique du moteur de stockage
  - A permis la mise en place du « stalecache »
  - Étendu avec un handler pour les systèmes de fichiers distribués (NFS, ...) en 4.2.

- Implémentations natives
  - eZFSFileHandler
  - eZFS2FileHandler
  - eZDBFileHandler
  - eZDFSFileHandler

  → Possibilité d'implémenter son propre moteur de stockage

Smile OPEN SOURCE SOLUTIONS

Mode « cluster »

Formation

Mode cluster par défaut : eZDBFileHandler

- Les fichiers binaires sont stockés en base de données
  - **Fichiers de contenus** (pdf, txt, ...)
  - **Images** et variations
  - **Caches** liés aux contenus

  → Assure la **cohérence entre les frontaux**

- Les caches locaux subsistent sur le disque
  (compilation des gabarits, ...)

  → Nécessité de passer par la commande **bin/php/ezcache.php**

Smile OPEN SOURCE SOLUTIONS

« Stale cache »

- Depuis eZ Publish 4.1, possibilité de servir du cache « obsolète » si la régénération est déjà en cours.

  → Évite les verrous de générations concurrentes (dead locks)

- Améliore sensiblement les performances lors des mises à jour.
- N'est pas activé par défaut (limites Windows)

  → Doit être activé sur tous les projets en production Linux ou PHP >= 5.3.

```
[ClusteringSettings]
FileHandler=eZFS2FileHandler
```

→ Attention, bug avec le cache_ttl=0 sur les premières 4.1.

Le cache statique

Formation

- Génération des pages sous forme de **fichiers HTML** directement exploitables par le serveur web

  → Améliore sensiblement les **performances**

- Peut être configuré dans le fichier **staticcache.ini**
  - Siteaccess concernés, pages ou arborescences à générer
  - Possibilités de générations systématiques
- Nécessite l'application de **règles de réécriture** Apache

```
RewriteCond  %{REQUEST_METHOD}      !^POST$
RewriteCond  /path/to/site/www/static$1/index.html -f
RewriteRule  (.*) /static$1/index.html [L]
```

- Depuis eZ Publish 3.10, la génération peut être effectuée périodiquement (cronjobs)
- Inconvénients
  - **Ralentissement** BO en synchrone ou temps de génération en asynchrone
  - Pas d'identification directement possible sur le site (statique)
  - Assez compliqué d'assurer la cohérence du site
    → Généralement à réserver sur quelques pages stratégiques

Reverse proxy

Formation

- **Outil de cache** extrêmement performant, en amont du serveur web
  - Squid
  - Varnish
- Principe de fonctionnement avec eZ
  - eZ Publish distribue les pages avec des **entêtes spécifiques**
    - Configuration possible dans site.ini / [HTTPHeaderSettings]
    - Configuration par pages / zones / arborescences
    - URL + wildcard
  - Le reverse proxy utilise ces entêtes pour décider du cachage / validité de la page
- Préconisé sur les les **sites à fortes charges**
- Doit être paramétré obligatoirement pour servir les fichiers binaires en mode cluster (éviter la surcharge de la base)
- Induit des limites applicatives importantes,

→ Le développement doit tenir compte de ces contraintes dès le démarrage du projet

Et après?

- Pour assurer un fonctionnement optimal
  - Soigner **l'architecture système**

    → Appliquer les bonnes pratiques **d'optimisation**
  - Soigner la **conception** / penser le projet en amont
  - Soigner la **réalisation**

- D'autres systèmes de caches peuvent être mis en oeuvre pour des besoins plus spécifiques
  - Contraintes de délais de publication / fort trafic

    → projets Sport24, 01Informatique
    - Plusieurs millions de pages servies
    - Publication garantie en 3 minutes
  - Gestion de blocs et dépendance (SSI)
  - La version d'eZ Publish 4.2 intègre des fonctionnalités de génération ESI (Edge Side Includes)

    → **Extension eZSI**

Questions?

Formation

Merci de votre attention

À vous de jouer, on passe à la pratique... ;)

Smile
OPEN SOURCE SOLUTIONS