INTELLAGENCE

Comment améliorer PHP ?
Avec un préprocesseur !

Présentation des avantages significatifs apportés par l'insertion d'un préprocesseur de code au sein de la fonction __autoload()

Nicolas Grekas et Yann Bogdanovic
IntellAgence Technologies

Vos conférenciers

- Nicolas Grekas

  Après ses études, Nicolas a fondé IntellAgence Technologies, une société spécialisée dans les services en ligne pour les chercheurs. Il est l'auteur du framework Patchwork, résultat de huit années de pratiques et de réflexions en PHP/Web. Cette conférence est pour lui l'occasion de contribuer à la communauté PHP et de sortir ses travaux de leur confidentialité.

- Yann Bogdanovic

  Yann est ingénieur R&D chez IntellAgence Technologies. Il y a deux ans, il est devenu le premier membre de la "communauté Patchwork". L'opportunité s'étant présentée de concilier centres d'intérêts et vie professionnelle, il a rejoint l'entreprise en septembre dernier.

Plan de la conférence

- **Introduction**
  - Présentation de *Patchwork*
  - Mécanisme du préprocesseur
- **Améliorations à l'exécution**
  - Vérifier l'encodage des sources
  - Analyse anticipative
  - Substituer des fonctions
  - Casser l'opérateur de silence
  - Booster l'autoload
  - Spécialiser en fonction du contexte d'exécution
- **Améliorations du langage**
  - Héritage multiple d'applications
  - Superposition de classes
  - Constructeurs et destructeurs statiques
  - Libérer le constructeur PHP4
- **Conclusion**

Présentation de *Patchwork*

- Protocole HTTP et gestion du cache
- Portabilité des applications
- Internationalisation
- Sécurité
- Bibliothèque de code
- Structuration du code

Mécanisme du préprocesseur

- Transformer le code source automatiquement
- Déroulement
  1. Contrôleur frontal (1er fichier exécuté)
  2. Charger `__autoload` et le code du préprocesseur
  3. Transformer les codes chargés par __autoload
  4. Exécuter le code source compilé
  5. Mettre en cache

Améliorations à l'exécution

- Vérifier l'encodage des sources
- Analyse anticipative
- Substituer des fonctions
- Casser l'opérateur de silence
- Booster l'autoload
- Spécialiser en fonction du contexte d'exécution

Vérifier l'encodage des sources

- UTF-8
- Normalisation canonique composée (NFC)

NFD	NFC
A ◌̊	Å
0041 030A	00C5

- Suppression des BOM parasites
- Normalisation des fins de ligne (CRLF, CR, **LF**)
- Amélioration possible :

```
<?php declare(encoding='ISO-8859-1'); // Backport PHP6
```

INTELLAGENCE

Analyse anticipative

- Divergences des tables de localisation

```
<?php T("Bonjour $nom") /*vs*/ sprintf(T('Bonjour %s'), $nom)
```

- Pré-remplissage des tables de localisation

INTELLAGENCE

Substituer des fonctions

- Remplacer `fonction()` par `maFonction()`
- Usages concrets
  - remplacer par un équivalent plus performant
  - contourner des bugs
  - greffer des fonctionnalités
  - ajouter des fonctionnalités
- Limité par la résolution de nom à l'exécution
- Améliorations possibles : *runkit*

Substituer des fonctions

- Remplacer par un équivalent plus performant

```
<?php

rand => mt_rand
md5  => hash('md5',
```

- Contourner des bugs

```
<?php

getcwd              => patchwork_getcwd
mb_encode_mimeheader => trigger_error('mb_encode_mimeheader()
   is bugged. Please use iconv_mime_encode() instead.',
```

Substituer des fonctions

- Greffer des fonctionnalités

```
<?php

// contrôle de la casse des noms de fichiers sous Windows
file_exists => win_file_exists

// intercepter et prendre le contrôle de certaines entêtes
header      => patchwork::header
```

- Ajouter des fonctionnalités

```
<?php

// charger un substitut PHP d'iconv à la demande
iconv                => utf8_iconv::iconv

// backporter des fonctions de PHP 5.3
normalizer_normalize => Normalizer::normalize
```

Casser l'opérateur de silence

- Idée originale : extension *scream*

  "L'extension scream *donne la possibilité de désactiver l'opérateur de contrôle d'erreur, de manière à ce que toutes les erreurs soient rapportées."*

- Voir toutes les erreurs,
  même celles qui veulent se cacher

```
<?php

// cherche bien le pourquoi de cette satanée page blanche :)
@include 'parse_error.php';
```

- Transformation : remplacer les @ par un espace
- Limitation : pas avant l'initialisation du préprocesseur

Booster l'*autoload*

- **Charger les classes, deux extrêmes**
  - Manuellement : `require`
  - Automatiquement : `__autoload()` + *include_path*
- **Le plus vite possible**
  - `__autoload()` : alourdit l'exécution mais dépendances faciles
  - `require` : des chargements superflus car gestion difficile
- **Le meilleur des deux ?**
  Substituer `__autoload()` par des `require` à l'exécution
  - permettre à `__autoload()` de connaître son point d'appel
  - insérer un `require` à cet endroit

Booster l'*autoload*

- **Performance**
  - Perte due aux marqueurs
  - Gain grâce aux substitutions
- **Benchmark**
  - Fonction `reject` précédente : +15%
  - Génération d'une page : -20%
- **Discussion**
  - PHP creux *vs* accès disques, BDD, *etc.*
  - Réalisable par les compilateurs d'opcode ?

Spécialiser en fonction du contexte d'exécution

- Du code statique résolu à l'exécution

```php
<?php

define('IS_WINDOWS',       '\\' === DIRECTORY_SEPARATOR     );


    if (!function_exists('utf8_decode'))
    {
        function utf8_decode($string)
        {
            // ...
        }
    }


    $a = file_get_contents('data/utf8/quickChecks.txt');
    $a = explode("\n", $a);
define('UTF8_NFC_RX',       '/' . $a[1] . '/u'     );
```

Spécialiser en fonction du contexte d'exécution

- Au code statique résolu à la compilation

```
<?php

define('IS_WINDOWS', /*<*/'\\' === DIRECTORY_SEPARATOR/*>*/);


/**/if (!function_exists('utf8_decode'))
/**/{
        function utf8_decode($string)
        {
            // ...
        }
/**/}


/**/$a = file_get_contents('data/utf8/quickChecks.txt');
/**/$a = explode("\n", $a);
define('UTF8_NFC_RX', /*<*/'/' . $a[1] . '/u'/*>*/);
```

Améliorations du langage

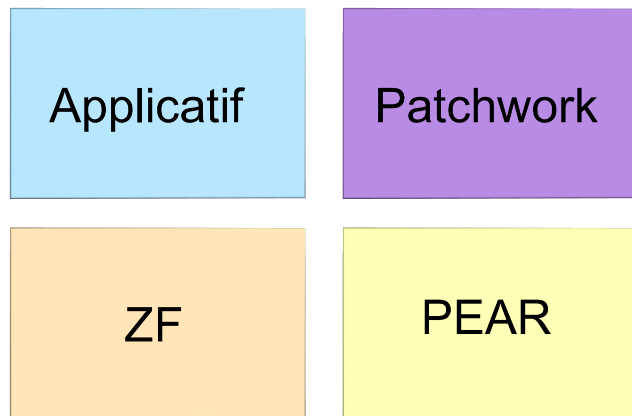
- Héritage multiple d'applications
- Superposition de classes
- Constructeurs et destructeurs statiques
- Libérer le constructeur PHP4

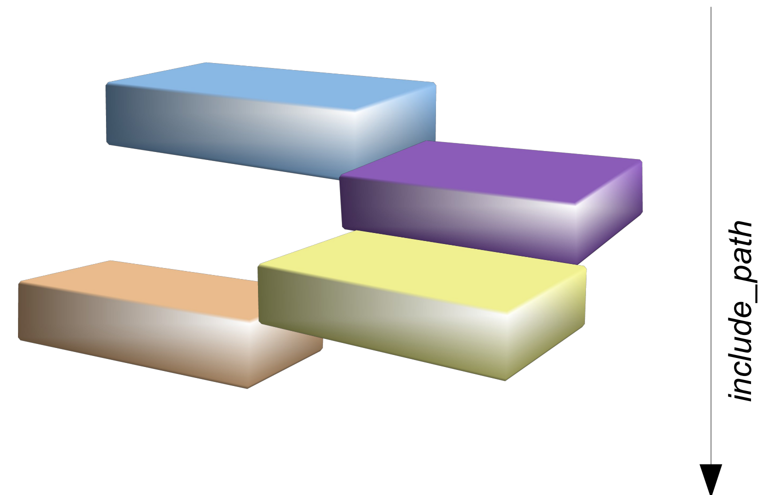
Superposition de classes

- Le sujet : modulariser son code
- Classiquement :
  décomposition fonctionelle + *include_path*
- Une nouvelle dimension :
  superposition de classes
- Programmation orientée aspect
  et injection de dépendances
  made easy

Héritage multiple d'applications

- Retour sur l'*include_path*

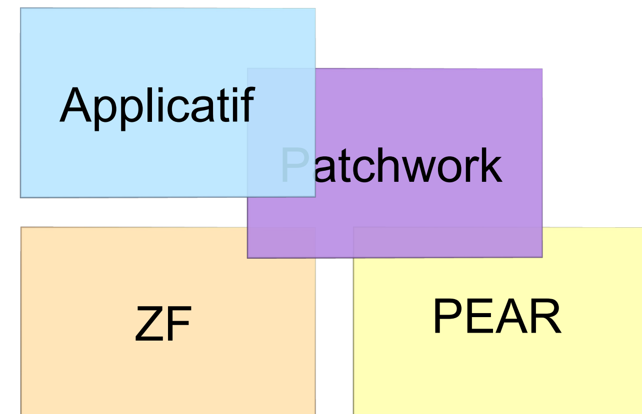




- Un développement de l'*include_path*
  - Un fichier de configuration par application
  - C3MRO : héritage multiple

Superposition de classes

- Superposition de fichiers
- Pas assez granulaire
- Superposition de classes




```php
<?php

class toto extends self
{
    // ...
}
```

INTELLAGENCE

Superposition de classes

- Patchwork - class/toto.php

```php
<?php

class toto
{
    function isAuth() {...};
}
```

- Applicatif – class/toto.php

```php
<?php

class toto extends self
{
    function isAuth()
    {
        $is_auth = parent::isAuth();
        $this->logAuth($is_auth);
        return $is_auth;
    };

    function logAuth() {...};
}
```

Superposition de classes

- Chaque méthode devient un *hook*

  Programmation orientée aspect

- Découple interface et implémentation

  Injection de dépendance

- Compatible avec les codes existants
- Réhabilite les méthodes statiques
- Transformations du préprocesseur :

  renommage des classes et des `self`

Superposition de classes

- Cas concrets
  - Sur-couche fonctionnelle
  - Système de plugin
  - Contournement de bug
  - Thème
  - Aspect
  - *etc.*
- Bonus à l'usage :

un excellent guide de factorisation !

Constructeurs et destructeurs statiques

- Même intérêt que pour les objets, mais pour les classes

```
<?php

class toto
{
    static function __constructStatic() {...}
    static function __destructStatic() {...}
}
```

- Exemples
  - Initialiser les variables statiques
  - Mettre en cache les tables de localisation
- Encore plus indispensable
  pour la superposition de classes

Libérer le constructeur PHP4

- La rétro-compatibilité du constructeur PHP4

```
<?php

class toto
{
    function toto(...) {...}
}
```

- Libérée grâce au préprocesseur !

Conclusion

- Un bénéfice mesurable
- Utilisé en production
- Au cœur de *Patchwork*
- Importer certaines idées dans PHP ?
- Une sous-couche pour les autres *framework* ?
- A vous de jouer !

Merci de votre attention

À vous la parole !

p@tchwork.com
http://pa.tchwork.com/

Plan de la conférence

- **Introduction**
  - Présentation de *Patchwork*
  - Mécanisme du préprocesseur
- **Améliorations à l'exécution**
  - Vérifier l'encodage des sources
  - Analyse anticipative
  - Substituer des fonctions
  - Casser l'opérateur de silence
  - Booster l'autoload
  - Spécialiser en fonction du contexte d'exécution
- **Améliorations du langage**
  - Héritage multiple d'applications
  - Superposition de classes
  - Constructeurs et destructeurs statiques
  - Libérer le constructeur PHP4
- **Conclusion**

Annexes

Booster l'*autoload*

- Code source - class/guardian.php

```
<?php

class guardian
{
    static function reject($type = '')
    {

        if ($type)
        {
            $type = 'guardian_' . $type;
            $type = new $type;
        }
        else
        {
            $type = new guardian_basic;

        }

        // ...
    }
}
```

Booster l'*autoload*

- Code compilé - .class_guardian.php

```
<?php

class guardian
{
    static function reject($type = '')
    {global $a,$b,$c;static $d=1;($e=$b=$a=__FILE__.'*572793636')&&$d&&$d=0;

        if ($type)
        {
            $type = 'guardian_' . $type;
            $type = (($a=$b=$e)?new $type:0);
        }
        else
        {
            $type = ((isset($c['guardian_basic']))||$a=__FILE__.'*121851464')?
new guardian_basic:0);
        }

        // ...
    }
}$GLOBALS['c']['guardian']=__FILE__.'*1431979276';
?>
```

Booster l'*autoload*

- Code compilé substitué - .class_guardian.php

```
<?php

class guardian
{
    static function reject($type = '')
    {global $a,$b,$c;static $d=1;($e=$b=$a=__FILE__.'*1863339583')&&$d
&&($c['guardian_robot']=16)&&$d=0;
        if ($type)
        {
            $type = 'guardian_' . $type;
            $type = (($a=$b=$e)?new $type:0);
        }
        else
        {
            $type = ((isset($c['guardian_basic'])||patchwork_include('.class_
guardian_basic.php')||1)?new guardian_basic:0);
        }

        // ...
    }
}$GLOBALS['c']['guardian']=1;
?>
```

Libérer le constructeur PHP4

- En PHP 5

```
<?php

class toto
{
    function __construct(&$un, $deux = 2) {...}
}
```

- En PHP 4

```
<?php

class toto
{
    function toto(&$un, $deux = 2) {...}
}
```

Rétro-compatibilité : fonctionne aussi en PHP 5

Libérer le constructeur PHP4

- Superposition de classe

```
<?php

class toto__3
{
    function toto(&$un, $deux = 2) {...}
}
```

- Compatibilité PHP 4

```
<?php

class toto__3
{
    function __construct(&$a0, $a1 = 2) {$this->toto($a0, $a1);}
    function toto(&$un, $deux = 2) {...}
}
```

En réalité un peu plus compliqué à cause du nombre de paramètres potentiellement variable