ATOUM

Le framework de tests unitaires pour PHP 5.3+ simple, moderne et intuitif

PMSIpilot, Lyon, le 18 avril 2011

Mojo technique d'ATOUM

- Modernité.
- Rapidité.
- Simplicité.
- Fiabilité.

Modernité

- PHP ≥ 5.3 :
  - Espaces de noms.
  - Late State Binding.
  - Fonctions anonymes et fermetures.
  - PHAR.

Dépendances optionnelles

- `Xdebug` pour la couverture de code.
- `XML` pour le rapport `xunit`.
- `SVN` pour l'intégration continue.

Rapidité

- Lors de l'installation :
  - Une archive PHAR à télécharger.
- À l'utilisation :
  - Une archive PHAR à inclure dans le fichier de test.
  - Fichier de tests exécutable directement.

Exemple de test

```
namespace vendor\project\tests\units;
use mageekguy\atoum, vendor\project;

require_once(__DIR__ . '/../../vendor/atoum.phar');

class foo extends atoum\test {
    public function testBar() {
        $foo = new project\foo();

       $this->assert
         ->object($foo->bar())->isIdenticalTo($foo)
       ;
    }
}
```

Exécution d’un test

```
# php path/to/test.php
> Atoum version 325 by Frédéric Hardy (/Users/fch/Atoum)
> PHP path: /Users/fch/PHP/Install/5.3.6/bin/php
> PHP version:
=> PHP 5.3.6 (cli) (built: Apr  3 2011 17:53:34)
=> Copyright (c) 1997-2011 The PHP Group
=> Zend Engine v2.3.0, Copyright (c) 1998-2011 Zend
Technologies
> vendor\project\tests\units\foo:
[S_____________________________________________][1/1]
=> Test duration: 0.14 second.
=> Memory usage: 1.00 Mb.
> Total test duration: 0.1 second.
> Total test memory usage: 1.00 Mb.
> Running duration: 0.5 second.
...
```

Alternatives pour l'exécution

```
# php atoum.phar -t path/to/tests/units/foo.php
> Atoum version 325 by Frédéric Hardy (/Users/fch/Atoum)
...
# php atoum.phar -d path/to/tests/units
> Atoum version 325 by Frédéric Hardy (/Users/fch/Atoum)
...
# ./atoum.phar path/to/test.php
> Atoum version 325 by Frédéric Hardy (/Users/fch/Atoum)
...
# ./atoum.phar -d path/to/tests/units
> Atoum version 325 by Frédéric Hardy (/Users/fch/Atoum)
...
```

Simplicité

- Simplifier l'écriture et la maintenance des tests
  - Grâce à une API explicite.
  - Grâce à des assertions expressives.
  - Grâce à la notion d'adaptateur.
  - Grâce à des bouchons simples à utiliser.
  - Grâce à des annotations.

Assertions expressives

```
...
$this->assert
  ->object($foo->bar())->isIdenticalTo($foo)
  ->string($foo->getBar())->isEqualTo('bar')
  ->integer($foo->countBar()
    ->isGreaterThan(100)
    ->isLessThan(1000)
;
...
```

- Miment le langage «naturel».
- Utilisent une interface fluide.
- Permettent plusieurs tests sur une même variable.
- Indépendantes de la classe de test.

Alias et labels

```
...
$this->define
  ->alias('int', 'integer')
  ->alias('str', 'string')
  ->object($foo)->is('foo')
;
$this->define->alias

$this->assert
  ->foo->isIdenticalTo($foo->getBar())
  ->str($foo->getBar())->isEqualTo('bar')
  ->int($foo->countBar()
    ->isGreaterThan(100)
    ->isLessThan(1000)
;
...
```

Adaptateur

- Proxy indépendant et générique pour :
  - Les appels aux fonctions de PHP :
    - `fopen()`, etc.
  - Les appels aux super-globales de PHP :
    - `$_SERVER`, `$_GET`, etc.
- Assertion dédiée.

Principe des adaptateurs

- Valeur des fonctions ou des super-globales :
  - Définissable par valeur.
  - Définissable via l'exécution d'une fonction anonyme.

```
$adapter = new project\adapter();
$adapter->_SERVER = array('foo' => 'bar');
$adapter->_GET = function() use ($request) {
  return $request->getValues();
};
$adapter->fopen = false;
$adapter->fopen[1] = true;
$adapter->md5 = function() { return md5(''); };
```

Définition d'un adaptateur

```
namespace vendor\project\adapter;
use \mageekguy\atoum;

interface adapter implements atoum\adapter\definition {}

namespace vendor\project\tests\units;
use \vendor\project\adapter, \mageekguy\atoum\test;

class adapter extends test\adapter
                implements adapter\definition {}

namespace vendor\project;
use \vendor\project\adapter, \mageekguy\atoum;

class adapter extends atoum\adapter
                implements adapter\definition {}
```

Utilisation de l'adaptateur dans une classe

```
namespace vendor\project;

class foo { ...
  public function __construct(adapter $adapter) {
    $this->adapter = $adapter;
  }

  public function write($string, $path) {
    if ($this->adapter->fopen($path) === false) {
      throw new \exception('Unable to open');
    } ...
  }
}
```

Utilisation de l'adaptateur dans les tests

```
namespace vendor\project\tests\units;
...
class foo ... { ...
  public function testWrite() {
    $foo = new project\foo($adapter = new test\adapter());
    $adapter->fopen = false;
    $this->assert->exception(function() use ($foo) {
        $foo->write(uniqid(), uniqid());
      }
    )
      ->isInstanceOf('\exception')
      ->hasMessage('Unable to open')
  }
}
```

Bouchonnages

- Extension de l'adaptateur.
- Possible de bouchonner :
  - des classes, des classes abstraites ou des classes inconnues,
  - des interfaces.
- Espace de nom dédié.
- Assertion dédiée.

Exemple de bouchonnage

```
namespace vendor\project\tests\units;

use \mageekguy\atoum\mock;
...
class foo ... { ...
   public function testWrite() {
      $this->mock('\vendor\project\file');

      $file = new mock\file();
      $file->getMockController()->open = false;

      $foo = new foo($path = uniqid());

      $this->assert
         ->exception(function() use ($foo)
          { $foo->open($file); }
         )
            ->isInstanceOf('\exception')
            ->hasMessage('Unable to open')
         ->mock($file)->call('open', array($path))
      ;
   }
}
```

Annotations

- Utilisées pour l'instant pour :
  - Ignorer une classe de test.
  - Ignorer une méthode de test dans une classe.

Fiabilité

- Isolation des tests.
- Traçage correct des erreurs et des exceptions.
- Rapports d’exécution simples et évolutifs.

Isolation des tests

- Un processus PHP « vierge » est utilisé pour exécuter chaque méthode d'une classe de test.
  - Aucune interaction entre les tests.
    - Pas de corruption de l'environnement.
    - Retour significatif en cas de crash de PHP.
    - Un crash ou une erreur dans une méthode ne perturbe pas les autres tests.
  - En contrepartie, perte de performance.

Traçage des erreurs et des exceptions

- Location précise des erreurs et des exceptions :
  - Dans la classe de tests d'origine par :
    - Méthode,
    - Numéro de ligne,
    - Assertion.
  - Dans la classe d'origine dans la mesure du possible.

Exemple de trace d’erreur

```
> Atoum version XXX by Frédéric Hardy (/path/to/atoum)
> vendor\project\foo:
=> Test duration: 0.00 second.
=> Memory usage: 0.00 Mb.
> Total test duration: 0.00 second.
> Total test memory usage: 0.00 Mb.
> Running duration: 0.08 second.
> Success (1 test, 1 method, 0 assertion, 1 error, 0
exception) !
> There is 1 error:
=> vendor\project\tests\units\foo::testBar():
==> Error 255 in /path/to/project/vendor/tests/units/classes/
foo.php on unknown line, generated by file /path/to/project/
vendor/tests/units/classes/foo.php on line 15:
Fatal error: Call to undefined function niqid()
```

Rapports simples et évolutifs

- Un rapport est un ensemble de champs.
- Un champ correspond à un type de données :
  - Durée d'exécution,
  - Assertion non vérifiée,
  - Erreur,
  - Etc.
- Remplis en fonction d'événements déclenchés lors de l'exécution des tests.

Deux familles de rapports

- Temps réel :
  - Le rapport est « écrit » au fur et à mesure de l'exécution des tests.
    - Utilisé par défaut en ligne de commande.
- Asynchrone :
  - Le rapport est « écrit » à la fin de l'exécution des tests.
    - xunit

Utilisation des rapports

- Possible de générer plusieurs rapport simultanément :
  - `CLI`,
  - `CLI` + `xunit`.
- Possible d’écrire sur plusieurs médias un même rapport :
  - `CLI`,
  - `CLI` + mail,
  - Fichier + mail.

Exemple de mise en œuvre des rapports

```
...
use \mageekguy\atoum;

$mailer = new atoum\mailers\mail();
$mailer
  ->to('dev@atoum.org');
  ->from('runner@atoum.org')
;

$report = new atoum\reports\asynchronous\vim();
$report
  ->addWriter(new atoum\writers\std\out());
  ->addWriter(new atoum\writers\mail($mailer))
;

$runner->addReport($report);
...
```

Exemple de rapport

```
# php path/to/test.php
> Atoum version 325 by Frédéric Hardy (/Users/fch/Atoum)
> PHP path: /Users/fch/PHP/Install/5.3.6/bin/php
> PHP version:
=> PHP 5.3.6 (cli) (built: Apr  3 2011 17:53:34)
=> Copyright (c) 1997-2011 The PHP Group
=> Zend Engine v2.3.0, Copyright (c) 1998-2011 Zend Technologies
> vendor\project\tests\units\foo:
[S_____________________________________][1/1]
=> Test duration: 0.14 second.
=> Memory usage: 1.00 Mb.
> Total test duration: 0.1 second.
> Total test memory usage: 1.00 Mb.
> Running duration: 0.5 second.
```

Quelques statistiques

- > 800 Ko de code.
- > 22 000 lignes de code.
- 2 développeurs.
- Utilisé chez F-Secure.

Ressources

- http://downloads.atoum.org/nightly.
- https://svn.mageekbox.net/repositories/atoum/trunk.
- Documentation en cours de rédaction.
- Site en cours de réalisation.
- Sortie publique en cours de planification.

Questions ?

Cette conférence est maintenant terminée,

vous pouvez reprendre une activité normale.