

Atelier Zend Framework : Eviter les boucles infinies de dispatching dans un modèle MVC à contrôleur frontal

par Julien Pauli ([Tutoriels, article et conférences PHP et développement web](#)) ([Blog](#))

Date de publication : 14/12/2007

Dernière mise à jour :

Le modèle MVC de Zend Framework fait intervenir une "boucle de dispatching". Et il est possible, par mégarde, négligence, absence de tests; de tomber alors dans une boucle infinie de l'application.

Un très simple plugin résout ceci.

- I - Rappels sur le modèle MVC et le dispatching dans ZendFramework
- II - Le plugin anti boucles infinies
- III - Une intégration frontController ?

I - Rappels sur le modèle MVC et le dispatching dans ZendFramework

Le modèle MVC de ZF, fait en général intervenir le contrôleur frontal (abrégié FC). Dans celui-ci se situe une boucle : la boucle de dispatching.

Dans cette boucle, les plugins sont traités, puis l'action principale, munie de ses 'helpers', puis à nouveau les plugins.

Si à la fin de ses 3 scénarii, le "jeton de dispatching" (un simple indicateur true false, mais on le nomme ainsi) est remis à false, alors la boucle recommence intégralement (on dit aussi "obtenir un nouveau jeton"). Il est donc tout à fait possible de tomber dans une boucle infinie, plutôt désastreuse.

Bien entendu les tests, unitaires et d'intégration notamment, ont pour but de détecter ces comportements. Il faut dire qu'ils sont heureusement rares, mais possibles, car l'erreur reste humaine.

L'ajout d'un plugin tout simple permet de repérer une boucle infinie de dispatching, configurable en plus, et de lever une exception.

II - Le plugin anti boucles infinies

Le code du scénario le plus simple d'une boucle de dispatching infinie (et donc le plus bête) :

Un contrôleur idiot

```
<?php
class indexController extends Zend_Controller_Action
{
    public function indexAction()
    {
        $this->getRequest()->setDispatched(false);
    }
}
```

Le bootstrap est carrément le plus simple qui soit, on a juste pour l'occasion mis le contrôleur d'action avec le bootstrap (c'est peu commode, mais ça passe pour cet atelier ;-)) :

Le bootstrap le plus simple

```
<?php
$appPath = realpath(dirname(__FILE__));

set_include_path($appPath . PATH_SEPARATOR . get_include_path() );

require_once 'Zend/Loader.php';
Zend_Loader::registerAutoload();

Zend_Controller_Front::run($appPath);
```

On lance ceci, et le serveur web tire la langue. Pour parler français on dit (dans une boucle assez longue) :

"Mets le jeton à true : met le jeton à false - tant que le jeton est à false". Je pense que si certains d'entre vous n'avaient pas compris, c'est plutôt clair ...

Même avec beaucoup de tests, une petite erreur peut passer inaperçue. Car la boucle, dans une application, est réellement plus complexe : nombreux plugins qui peuvent toucher à ce jeton (error_handler en fait partie), actions qui peuvent demander le rendu d'autres actions (et donc remettre le jeton à false), et ceci quelques fois à l'aide de 'helpers', qui sont donc encore d'autres classes susceptibles de toucher à ce paramètre dit "jeton".

La solution passe par un plugin. Celui-ci va agir en postdispatch, et va simplement compter le nombre de fois qu'il est exécuté alors que le jeton est mis à false (le nombre de boucles donc). Si un seuil paramétrable est franchi : il lève une exception.

Il faut faire en sorte de lever une exception, mais un simple throw ne va pas suffire!

2 possibilités : on a spécifiquement dit à FC de rendre les exceptions (**throwExceptions(true)**), celui-ci va donc vous la faire suivre dans l'immédiat. Soit alors, on utilise le comportement par défaut : le plugin error_handler capte donc cette exception pour nous. Mais si on a désactivé ce plugin, et qu'on a pas dit à notre FC d'envoyer les exceptions, alors la boucle infinie est bel et bien là.

Nous devons donc gérer tous ces cas-là. J'ai géré ceci avec le debuggeur, je n'ai pas écrit de tests d'intégration (j'aurais pu sur une appli bidon, certes). J'ai en revanche écrit un petit contrat, sous forme de tests unitaires (PHPUnit); voyons un peu le contrat de notre plugin :

maxDispatchPluginTest, autoload supposé activé, comme d'habitude

```
<?php
```

maxDispatchPluginTest, autoload supposé activé, comme d'habitude

```
if (!defined('PHPUnit_MAIN_METHOD')) {
    define('PHPUnit_MAIN_METHOD', 'maxDispatchPluginTest::main');
}

class maxDispatchPluginTest extends PHPUnit_Framework_TestCase {

    private $fixture;
    private $request;
    private $response;
    private $loops;

    public static function main()
    {
        $suite = new PHPUnit_Framework_TestSuite('maxDispatchPluginTest');
        $result = PHPUnit_TextUI_TestRunner::run($suite);
    }

    protected function setUp()
    {
        $this->request = new Zend_Controller_Request_Http();
        $this->response = new Zend_Controller_Response_Http();

        $this->fixture = new maxDispatchPlugin($this->loops = 30);
        $this->fixture->setResponse($this->response);
        $this->fixture->setRequest($this->request);
        try{
            Zend_Controller_Front::getInstance()->dispatch(clone $this->request, clone
$this->response);
        }catch(Exception $e) { }
    }

    public function testPluginCorrectementInitialise()
    {
        $this->assertType('Zend_Controller_Response_Http', $this->fixture->getResponse());
        $this->assertType('Zend_Controller_Request_Http', $this->fixture->getRequest());
        $this->assertType('Zend_Controller_Plugin_Abstract', $this->fixture);
        $this->assertEquals($this->loops, $this->fixture->getMaxDispatchLoops());
    }

    public function testSetMaxDispatchLoops()
    {
        $this->assertEquals($this->fixture, $this->fixture->setMaxDispatchLoops($this->loops));
        $this->assertEquals($this->loops, $this->fixture->getMaxDispatchLoops());
    }

    public function testGetMaxDispatchLoops()
    {
        $this->assertEquals($this->loops, $this->fixture->getMaxDispatchLoops());
    }

    public function testPostDispatchWithErrorHandlerActivated()
    {
        for($i=0;$i<=$this->loops;$i++)
        {
            $this->fixture->postDispatch($this->request);
            if ($this->fixture->getResponse()->hasExceptionOfType('Zend_Controller_Exception'))
            {
                return;
            }
        }
        $this->fail('Response object does not contain expected Zend_Controller_Exception');
    }

    public function testPostDispatchWithErrorHandlerNotActivated()
    {
        Zend_Controller_Front::getInstance()->unregisterPlugin('Zend_Controller_Plugin_ErrorHandler');
```

maxDispatchPluginTest, autoload supposé activé, comme d'habitude

```
for($i=0;$i<=$this->loops;$i++)
{
    try{
        $this->fixture->postDispatch($this->request);
    }catch(Zend_Controller_Exception $e){
        return;
    }
    $this->fail('Zend_Controller_Exception not met');
}

if (PHPUnit_MAIN_METHOD == 'maxDispatchPluginTest::main')
{
    maxDispatchPluginTest::main();
}
```

Cette classe de test n'utilise aucun objet Mock, de ce fait, nous supposons toutes les classes utiles (Zend_Controller_Front, Request, Response ...) correctement écrites et testées :-)

De plus, il nous faut initialiser chacun des objets, et passer au moins une fois dans une boucle de dispatching de FC, avant tout test, afin que celui-ci initialise bien ses plugins et tout son environnement.

On voit bien que la programmation guidée par les tests définit facilement le contrat de la classe. Il ne reste plus qu'à l'écrire :

maxDispatchPlugin

```
<?php
class maxDispatchPlugin extends Zend_Controller_Plugin_Abstract
{
    private $_counter = 1;
    private $_maxLoop;

    public function __construct($loops = 10)
    {
        return $this->setMaxDispatchLoops($loops);
    }

    public function getMaxDispatchLoops()
    {
        return $this->_maxLoop;
    }

    public function setMaxDispatchLoops($loops)
    {
        $this->_maxLoop = abs((int)$loops);
        return $this;
    }

    public function postDispatch(Zend_Controller_Request_Abstract $request)
    {
        if (!$request->isDispatched()) {
            ++$this->_counter;
            if ($this->_counter > $this->_maxLoop) {
                $e = new Zend_Controller_Exception('Maximum dispatch loop count reached : ' .
                $this->_maxLoop);
                if
                (Zend_Controller_Front::getInstance()->hasPlugin('Zend_Controller_Plugin_ErrorHandler')) {
                    $this->getResponse()->setException($e);
                } else {
                    Zend_Controller_Front::getInstance()->throwExceptions(true);
                    throw $e;
                }
            }
        }
    }
}
```

maxDispatchPlugin

```
}  
}  
}  
}
```

Il est très important, si le plugin `error_handler` n'est pas activé, de demander au FC de renvoyer l'exception, explicitement (**`throwExceptions()`**), sinon par défaut, l'exception va être rajoutée en pile à la réponse, et ne sera jamais envoyée.

III - Une intégration frontController ?

Ce plugin n'est en réalité que le fruit d'une idée que j'ai proposée à l'origine pour être intégrée au FC. Ce n'est actuellement pas dans les priorités, car les tests doivent détecter ce genre de problème.

Cependant, étant donné qu'il s'agit d'un problème très grave (une boucle infinie dans une application est d'une grande sévérité), et que même avec des tests très poussés, sur une application très lourde, ce scénario n'a pas une probabilité nulle d'être rencontré, il peut être intéressant d'intégrer un tel système au sein même du FC.

C'est un jeu d'enfant à propos duquel je vous laisse méditer ^^ . Faites attention tout de même à ne pas avoir la main trop légère : une valeur maximale de boucles, de 10, par exemple, est insuffisante. Il n'est pas rare, surtout sur des applications complexes, qu'une action puisse se décomposer en plusieurs sous-actions.

Celles-ci sont alors "forwardées", via la méthode `_forward()` de `Zend_Controller_Action`, ce qui a pour effet de relancer un jeton. Idem pour certains plugins (dont c'est un des rôles, après tout).

