

AspectPHP

An Extension for Aspect-Oriented Programming to the PHP Programming Language

Sebastian Bergmann <sb@sebastian-bergmann.de>

1 Introduction

1.1 PHP

PHP [1] is a widely-used general-purpose programming language that is especially suited for Web development. It is dynamically typed and supports both procedural and object-oriented programming. For the latter it provides a class-based object model that is similar to those of C# or Java.

1.2 AspectPHP

The goal of the AspectPHP [2] project is to provide an extension to the PHP programming language that supports aspect-oriented programming using dynamic weaving of aspect- and base-code at runtime.

2 Characteristics of AspectPHP

AspectPHP can be characterized as follows:

1. Possible Quantifications (Join Points): Attribute read and write access, call and execution of a method, execution of a catch-block, initialization of an object.
2. Possible Interactions
 - (a) Advice: before-, after- and around-advice for the above mentioned join points.
 - (b) Introduction: Attributes and methods can be added to existing PHP classes, inheritance relations and interface implementations of existing PHP classes can be altered.

3 Implementation

3.1 Possible Implementation Approaches

An extension to the PHP programming language that supports aspect-oriented programming using dynamic weaving should be implemented in one of two ways: The Zend Engine, the compiler and interpreter of PHP that is written in C, can be extended. Alternatively, the language extension can also be written in the PHP programming language itself by leveraging its meta-programming capabilities.

3.2 Implementation of AspectPHP

The current prototype for AspectPHP is implemented in the PHP programming language itself. It uses the Runkit [3] extension to the PHP interpreter to manipulate the PHP-internal bytecode of classes and methods.

Aspects are plain PHP classes that contain pointcut, advice, or introduction annotations in their code comments.

AspectPHP uses a classloader that handles the loading of the source files that contain the code for aspects and classes. When loading the source file for an aspect, the class representing the aspect is searched for annotations and the pointcuts declared in the aspect are registered. When loading the source file for a class, the necessary hooks for the join points as well as introductions declared by aspects are inserted into the PHP-internal bytecode of the class and its methods.

At runtime, the previously introduced hooks for the join points check whether or not an aspect has a pointcut with an advice for the current join point. If that is the case, AspectPHP's Dispatcher class handles the execution of the corresponding advice and passes the current context to the advice's method(s).

The variable `$joinPoint` is available in the advice methods and contains context information for the current join point. This information includes, for instance, the calling object as well as the object called for a method call join point.

4 Example: Logging Method Calls

```
<?php
/**
 * Declare a new pointcut named callPointCut using the @pointcut annotation that captures
 * join points on a method regardless of modifier, class, method, or number of parameters.
 * @pointcut callPointCut() : call(* *->*(..));
 *
 * Bind the after() method of this aspect as an after-advice to the
 * previously declared pointcut.
 * @after callPointCut : LoggingAspect->after();
 */
class LoggingAspect {
    public function after($joinPoint) {
        printf(
            "%s->%s() called %s->%s()\n",
            $joinPoint->getSource()->getDeclaringClass()->getName(),
            $joinPoint->getSource()->getName(),
            $joinPoint->getTarget()->getDeclaringClass()->getName(),
            $joinPoint->getTarget()->getName()
        );
    }
}
```

5 Related Work

The author of this poster knows of four existing approaches to facilitate aspect-oriented programming with the PHP programming language:

- *PHPAspect* [4] uses a compiler, written in the PHP programming language, that performs static weaving using source code transformations. A downside of this approach is that advantages that stem from PHP's interpreted nature are lost.
- *Aspect-Oriented PHP* [5] uses a preprocessor for the PHP programming language written in Java that is responsible for the weaving of aspect- and base-code. Due to its Java implementation this approach does not integrate seamlessly with the PHP platform.
- *aspectPHP* [6] is a reimplement of Aspect-Oriented PHP in C, available as a patch against (not as an extension to) PHP 4.3.10.
- The *AOP Library for PHP* [7] requires manual changes to the base-code and thus does not provide obliviousness.

6 About the Author

Sebastian Bergmann is a Computer Science student in his final year at the University of Bonn. He spends his free time with the development of Free Software, is a member of the PHP and Gentoo Linux development teams and author of a variety of PHP software projects such as PHPUnit.

7 References

1. <http://www.php.net/>
2. <http://www.sebastian-bergmann.de/AspectPHP/>
3. <http://pecl.php.net/package/runkit>
4. <http://www.phpaspect.org/>
5. <http://www.aopphp.net/>
6. <http://www.cs.toronto.edu/~yijun/aspectPHP/>
7. <http://pear.php.net/pepr/pepr-proposal-show.php?id=315>