PHP-Autoloading ohne Fehler-Kontroll-Operator

Ein Tipp von unserem Entwickler Claudio Kressibucher

Köln/Bergisch Gladbach

www.kennziffer.com GmbH
Friedrich-Ebert-Straße 75
51429 Bergisch Gladbach

Telefon 02204-84266 0

Münster

www.kennziffer.com GmbH
Königstraße 32-33
48143 Münster

Telefon 0251-590 595 81

Aachen

team in medias GmbH
Karl-Friedrich-Straße 74
52077 Aachen

Telefon 0241-980 998 70

Kapstadt/Südafrika

Marc Koebler
3 Sillery Rd
7935 Bergvliet-Cape Town

Telefon 02204-84266 2

PHP-Autoloading ohne Fehler-Kontroll-Operator

In PHP gibt es diese schöne Möglichkeit, Fehler (und Warnungen, etc.) mit einem @ vor dem entsprechenden Ausdruck zu unterdrücken. Es gilt mittlerweile als schlechte Praxis, diesen Operator zu verwenden, zumindest, wenn es alternative Möglichkeiten gibt. Das Problem ist in erster Linie, dass man damit alle möglichen Fehler, und nicht nur die erwarteten, unterdrückt. Und dann: Happy debugging...! (Ein Beispiel folgt am Ende des Artikels)

Ein Beispiel, bei dem der Operator noch manchmal verwendet wird, sind autoloader Funktionen. Die Idee ist hier, ein File per include zu laden -- falls es exisitiert -- oder aber nichts zu tun, wenn die Datei nicht existiert. Simple Lösung: ein @include $file.

Als komplettes Code-Beispiel:

```
function ($className) {
    $filePath = strtr(
        ltrim($className, '\\'),
        array(
            '\\' => '/',
            '_'  => '/'
        )
    );
    @include $filePath . '.php';
}
```

Dabei gibt es nun potentiell mehrere include-Pfade. Eine naheliegende Alternative ohne Verwendung des @ Operators wäre, alle include-Pfade zu prüfen:

```
function ($className) {
    $filePath = strtr(
        ltrim($className, '\\'),
        array(
            '\\' => '/',
            '_'  => '/'
        )
    );
    $paths = explode(PATH_SEPARATOR, get_include_path());
    foreach ($paths as $p) {
        if (file_exists($path . DIRECTORY_SEPARATOR . $filePath)) {
            include($path . DIRECTORY_SEPARATOR . $filePath);
        }
    }
}
```

Diese Lösung ist nicht wirklich ideal. Einerseits ist der Code deutlich komplexer geworden, andererseits ist vermutlich auch die Performance schlechter (ich denke, dass PHP mit dem include statement effizienter alle Pfade prüfen kann, als mit mehreren file_exists -- allerdings habe ich das weder gemessen, noch mir den source code angeschaut, um das mit Sicherheit sagen zu können).

Die Lösung: stream_resolve_include_path

Auf den PHP-Doc Seiten bin ich auf eine Funktion gestoßen, die scheinbar zur Lösung dieses Problems geschaffen
wurde: stream_resolve_include_path. Sie versucht, einen Dateipfad unter Berücksichtigung der include-Pfade aufzulösen. Im Unterschied
zu include macht sie jedoch nichts anderes, als den aufgelösten Pfad zurückzugeben.

Kann der Pfad nicht aufgelöst werden, wird false zurückgegeben. Damit lässt dich der Autoloader wie folgt umbauen:

```
function ($className) {
    $filePath = strtr(
        ltrim($className, '\\'),
        array(
            '\\' => '/',
            '_'  => '/'
        )
    );
    $res = stream_resolve_include_path($filePath . '.php');
    if ($res) {
        include $res;
    }
}
```

Im Vergleich zur ersten, schlechten, Lösung ist hier ein if Statement hinzugekommen. Was die Performance betrifft, ist ebenfalls kein wesentlicher Unterschied zu erwarten. Falls der Vorteil noch nicht verständlich ist, hier ein Beispiel aus dem Entwickler-Alltag:

Es gibt eine abstrakte Klasse A.

```
abstract class A {

  // some code...

  abstract protected function doSomething();
}
```

Diese wird nun erweitert durch die Klasse B. Dummerweise wird dabei vergessen, die Methode doSomehing zu implementieren.

```
class B extends A {

    // some code... no doSomething method
}
```

In einem Unit-Test versuche ich die Klasse zu instanziieren:

```
protected function setUp() {
    $this->model = new B();
}
```

Resultat mit einem @include autoloader? PhpUnit wird mit exit-status 255 beendet (eben weil die implementierende Klasse B die Methode doSomething nicht implementiert). Keine weitere Information dazu. Ein "Step-Debugging" mit XDebug zeigt mir, dass der Fehler beim autoloading auftritt. Allerdings ist der Pfad, welcher der include Funktion übergeben wird, korrekt.

Es gibt also einen Fehler beim include Statement, jedoch nicht den Fehler, den der Entwickler bei Verwendung von @ in seinen Gedanken hatte. Genau das ist das Problem mit dem @ Operator. Es gibt zwar die Möglichkeit, die unterdrückten Fehler-Meldungen abzufragen, allerdings

muss dafür in der PHP-Konfiguration track_errors eingeschaltet sein, und man muss die Fehler dann auch wirklich abfragen...

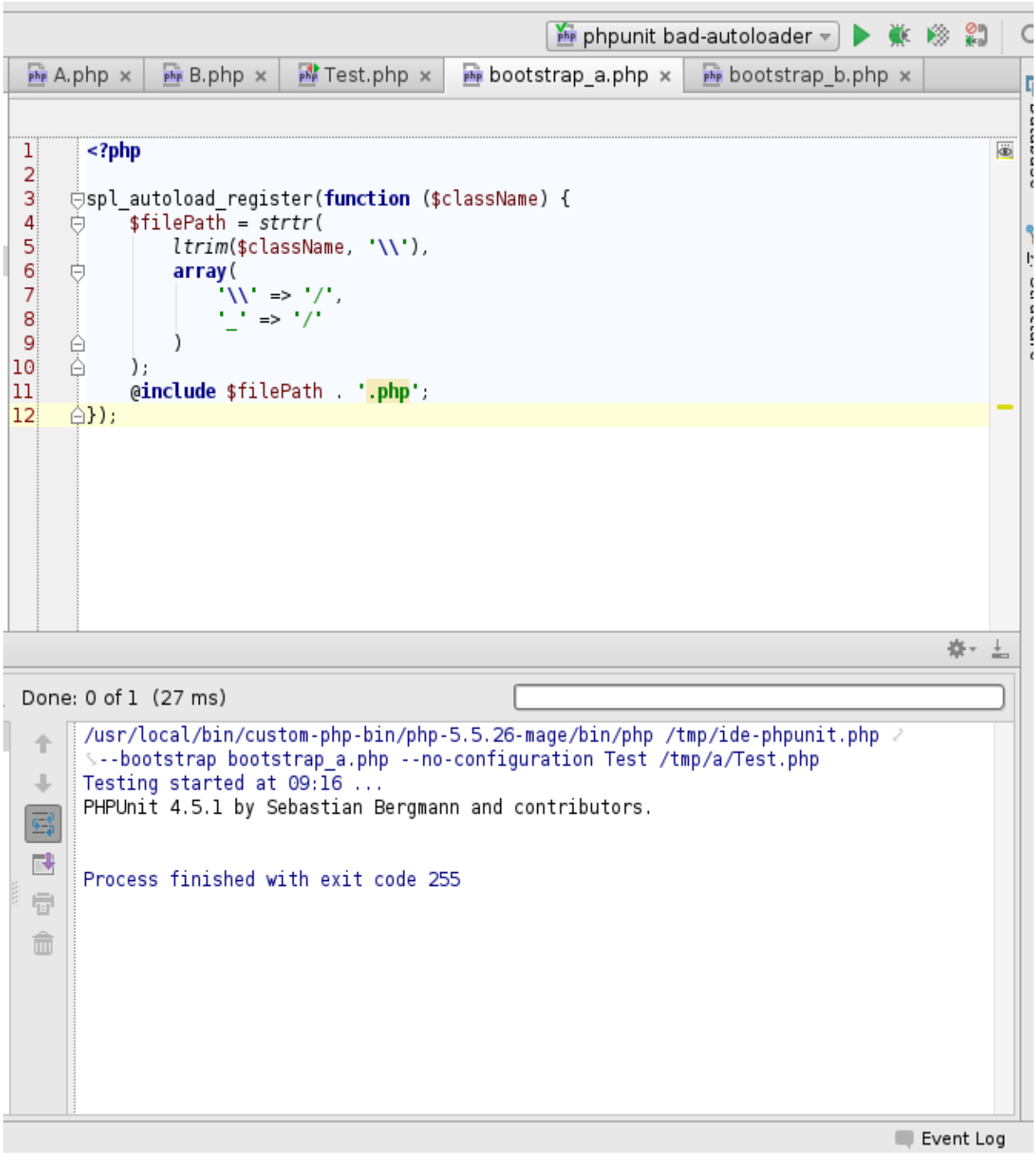

Abbildung 1: Beispiel mit @include Autoloader

Zum Vergleich, was passiert in derselben Situation mit dem überarbeiten Autoloader? Die Funktion stream_resolve_include_path gibt einen gültigen Pfad zurück (die Datei mit der Klasse existiert).

Dann wird include aufgerufen, ohne Fehler zu unterdrücken. Der "erlaubte" Fehler, dass die Datei nicht existiert, ist durch die if Anweisung ausgeschlossen. Unser Source-Code Fehler wird jedoch erkannt, und bewirkt einen Fatal Error mit einem aussagekräftigen Hinweis:

```
PHP Fatal error: Class B contains 1 abstract method and must
therefore be declared abstract or implement the remaining
methods (A::doSomething) in /tmp/a/B.php on line 4
```

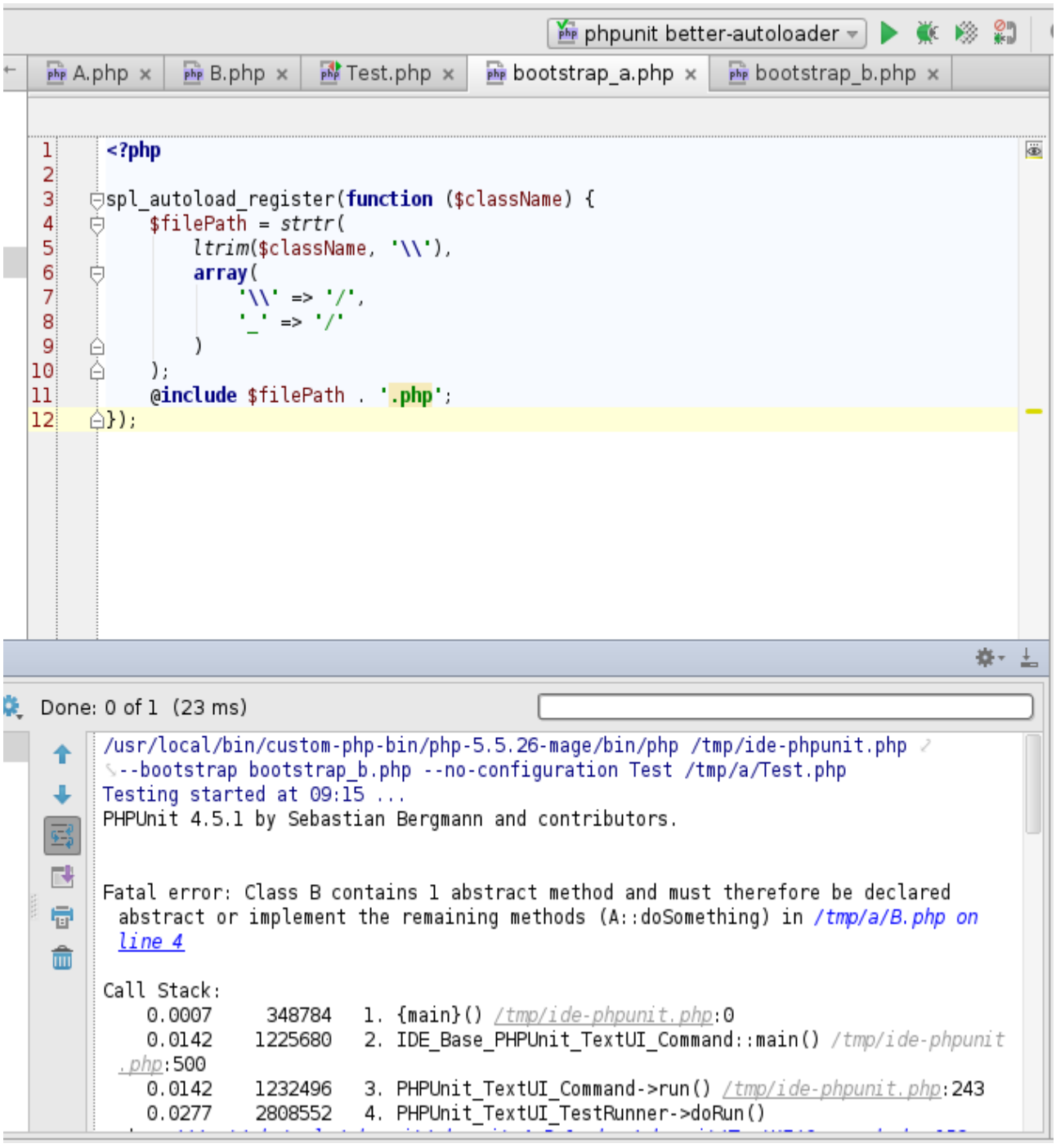

Abbildung 2: Beispiel mit stream_resolve_include_path Autoloader

Fazit

Es gibt meistens eine bessere Lösung, als den @ Operator zu verwenden.
Bei autoloader Funktionen heißt diese: stream_resolve_include_path.

Der Autor

Claudio Kressibucher

Entwickler

claudio.kressibucher@inmedias.de

Kein Shop kann komplex genug für ihn sein. Mit schweizer Präzision bringt der Informatiker jede Extension in Magento zum Laufen.

Als Sicherheitsjunkie sorgt er nicht nur bei Magento für Beruhigung beim Kunden, sondern auch bei der Einrichtung seiner Soundanlage ist jedes Kabel an seinem Platz.