Feature

Zend Framework

Filtern für Fortgeschrittene

Die Komponente *Zend\_Filter* des Zend Framework filtert Daten nach bestimmten Kriterien.

In diesem Workshop werfen wir einen Blick auf ein paar Filter im Lieferumfang von *Zend\_Filter*, die eingebaute PHP-Funktionalität pfiffig kapseln und einen praktischen Zusatznutzen bieten können. Wie üblich untermauern zahlreiche lauffähige Codebeispiele die verfügbaren Möglichkeiten.

Systemcheck

Wie bei allen vorherigen Artikeln aus unserer Zend-Framework-Serie gelten auch dieses Mal einige Systemvoraussetzungen und Konventionen. Basis ist die aktuellste zum Redaktionsschluss vorliegende Version von Zend Framework; diesmal ist es Nummer 1.10.0. Einige der Beispiele dieses Workshops erfordern auch mindestens diese Ausgabe von Zend Framework (Bild 1).

Unter framework.zend.com gibt es mehrere Varianten des Zend Framework zur Auswahl und zum Download; die minimale Version ist für unsere Zwecke absolut ausreichend. Im heruntergeladenen Archiv steckt ein Verzeichnis namens *library*; dieses muss in der *include\_path*-Einstellung von PHP auftauchen. Alle eigenen Codebeispiele liegen im virtuellen Verzeichnis */php/ZendFramework*, das Zend Framework wiederum in */php/ZendFramework/ZF/library*. Die folgende Anweisung passt den *include\_path* zur Laufzeit an; Sie müssen nur noch Ihre möglicherweise abweichende lokale Pfadstruktur einfügen:

set\_include\_path(

$\_SERVER['DOCUMENT\_ROOT'] .

'/php/ZendFramework/ZF/library' .

PATH\_SEPARATOR .

get\_include\_path());

Zwei der (zahlreichen) neuen Klassen von Zend Framework 1.10.0 sind *Zend\_Filter\_Compress* und *Zend\_Filter\_Decompress*. Der Name ist Programm: damit ist es möglich Daten zu komprimieren und wieder zu entkomprimieren. Abhängig von der PHP-Installation und -Konfiguration unterstützt das Zend Framework eine ganze Reihe von Formaten und auch von Möglichkeiten des Packens beziehungsweise Entpackens. Beginnen wir zunächst mit den Datenformaten. Die folgenden werden unterstützt:

((Aufzählung))

Bz2

Gz

Lzf

Rar

Tar

Zip

((Ende Aufzählung))

Wundern Sie sich nicht über die ungewöhnlichen Schreibweisen (beispielsweise schreibt man in der Regel ZIP statt Zip) – die oben angegebenen Vorgaben werden so an späterer Stelle vom Zend Framework auch erwartet. Nicht all diese Formate funktionieren jedoch ohne weiteres Eingreifen; die zugehörige PHP-Erweiterung (etwa Bz2 für die BZip2-Unterstützung oder Zlib für die GZ-Unterstützung) muss vorhanden sein. Einzige Art von Ausnahme ist das TAR-Format, das nur bei Vorhandensein des PEAR-Pakets Archive\_Tar zur Verfügung steht.

Wie bereits zuvor erläutert sind die beiden zentralen Klassen *Zend\_Filter\_Compress* und *Zend\_Filter\_Decompress*. Beginnen wir mit dem Komprimieren von Daten. Diese Daten können Dateien oder auch einfach Strings sein. Das in Listing 1 gezeigte HTML-UI stellt die Grundlage für ein kleines Beispielskript dar.

((Listingkasten))

Listing 1: HTML-UI

<form method="get" action="">

<input type="text" name="q" /><br />

<input type="radio" name="typ" value="Bz2" />BZip2

<input type="radio" name="typ" value="Gz" />GZip

<input type="radio" name="typ" value="Lzf" />Lzf

<input type="radio" name="typ" value="Rar" />Rar

<input type="radio" name="typ" value="Tar" />Tar

<input type="radio" name="typ" value="Zip" />ZIP<br />

<input type="submit" value="Komprimieren" />

</form>

((Ende Listingingkasten))

Auf Mausklick soll der eingegebene Text im gewählten Format komprimiert werden – wobei Rar und Tar nur der Vollständigkeit halber aufgeführt sind, denn sie unterstützen keine Zeichenketten als Eingabe, lediglich Dateien und Verzeichnisse (dazu später mehr).

Das eigentliche Komprimieren wiederum ist schnell geschehen. Zunächst muss die Klasse *Zend\_Filter\_Compress* instanziiert werden; als Argument geben Sie das Format gemäß obiger Liste an.

$filter = new Zend\_Filter\_Compress('Lzf');

Im nächsten Schritt wird die *filter()*-Methode aufgerufen. Ergebnis: der komprimierte Text.

$filter->filter($q)

Wie in Bild 2 zu sehen, lohnt sich das Vorgehen natürlich nur ab einer gewissen Textmenge, damit die komprimierten Informationen auch wirklich weniger Speicherplatz benötigen als die Quelldaten. Listing 2 zeigt den relevanten Code für die HTML-Oberfläche.

((Listingkasten))

Listing 2: HTML-Oberfläche

$typen = array('Bz2', 'Gz', 'Lzf', 'Rar', 'Tar', 'Zip');

if (isset($\_GET['q']) && is\_string($\_GET['q']) &&

isset($\_GET['typ']) && in\_array($\_GET['typ'], $typen)) {

require 'Zend/Filter/Compress.php';

$q = $\_GET['q'];

$filter = new Zend\_Filter\_Compress($\_GET['typ']);

echo 'Ergebnis: ' . htmlspecialchars($filter->filter($q));

}

((Ende Listingingkasten))

*Zend\_Filter\_Compress* kann aber auch Dateien erzeugen, die dann die komprimierten Informationen enthalten. Sie müssen dazu nur noch ein paar zusätzliche Daten angeben, wenn Sie mit der Klasse arbeiten – und zwar in Form eines Arrays. Zwei Werte sollten Sie angeben:

((Aufzählung))

*adapter*: Das Dateiformat

*options*: Ein Unter-Array mit weiteren Informationen wie dem Dateinamen des Archivs (zugehöriger Schlüssel: *archive*)

((Ende Aufzählung))

Hier ein Beispiel:

$optionen = array(

'adapter' => 'Lzf',

'options' => array(

'archive' => '/pfad/zu/datei.lzf'));

Das Beispiel auf der PHP-Journal-Website arbeitet mit derselben Oberfläche wie das vorangegangene Listing. Auf Basis des Komprimierungstypen wird die Dateiendung ermittelt und dann die Datei *datei.xyz* erzeugt – *xyz* steht für das Dateiformat, beispielsweise *zip*. Beachten Sie unbedingt, dass der Code lediglich zu Erklärungszwecken dient. Sollte es die angegebene Datei bereits geben, wird sie ohne Rückfrage überschrieben. Und PHP benötigt natürlich auch Schreibrechte für den Zielort (Bild 3).

Genug der Vorrede, Listing 3 zeigt den relevanten zugehörigen Code.

((Listingkasten))

Listing 3: Komprimierung

require 'Zend/Filter/Compress.php';

$q = $\_GET['q'];

$dateiname = 'datei.' . strtolower($\_GET['typ']);

$optionen = array(

'adapter' => $\_GET['typ'],

'options' => array('archive' => $dateiname));

$filter = new Zend\_Filter\_Compress($optionen);

$filter->filter($q);

echo "<a href=\"$dateiname\">Zur Datei</a>";

((Ende Listingingkasten))

Die umgekehrte Richtung – Daten entpacken statt packen – geht natürlich auch. Zuständige Klasse ist wie oben bereits erwähnt *Zend\_Filter\_Decompress*. Als Eingabeparameter für die *filter()*-Methode kommen diesmal die komprimierten Daten bzw. der zugehörige Dateiname zum Einsatz:

$filter->filter('datei.zip');

Für den Klassenkonstruktor jedoch benötigen wir wieder ein paar Optionen, die in Form eines Arrays übergeben werden. Ein alter Bekannter, *adapter*, gibt erneut an was für ein Komprimierungsformat vorliegt. Im Konfigurationsarray *options* instruiert der Wert *target* das Zend Framework, wohin die Datei entpackt werden soll. Die zuvor schon vorgebrachten Warnhinweise gelten auch hier: PHP benötigt Schreibrechte in das angegebene Verzeichnis, und man sollte nicht wie im Beispiel einfach Dateien ohne zusätzliche Prüfungen entpacken.

Für unsere Zwecke reicht aber ein vereinfachtes Szenario bereits aus. Der Nutzer kann per HTML-Datei-Upload ein Archiv an den Server übertragen sowie das zugehörige Dateiformat auswählen. Letzteres könnte man zwar auch aus der Dateiendung versuchen zu ermitteln, aber so vereinfacht sich das Testen ein wenig (Listing 4).

((Listingkasten))

Listing 4: Datei-Upload

<form method="post" action="" enctype="multipart/form-data">

<input type="file" name="datei" /><br />

<input type="radio" name="typ" value="Bz2" />BZip2

<input type="radio" name="typ" value="Gz" />GZip

...

<input type="submit" value="Dekomprimieren" />

</form>

((Ende Listingingkasten))

Der serverseitige Code (Listing 5) greift auf die übertragene Datei zu und entpackt sie mithilfe des Filters.

((Listingkasten))

Listing 5: Entpacken auf dem Server

require 'Zend/Filter/Decompress.php';

$dateiname = $\_FILES['datei']['tmp\_name'];

$optionen = array(

'adapter' => $\_POST['typ'],

'options' => array('target' => './entpackt'));

$filter = new Zend\_Filter\_Decompress($optionen);

$filter->filter($dateiname);

echo '<p>Archiv entpackt.</p><p>';

((Ende Listingingkasten))

Sie können dann im Dateisystem des Servers überprüfen ob der Dekomprimierungsvorgang erfolgreich war. Zusätzlich gibt der Code in Listing 5 noch den Inhalt des Zielverzeichnisses an. Bedenken Sie aber dass bei mehrmaligem Entpacken in dasselbe Verzeichnis sowohl noch Altdateien vorhanden oder gar überschrieben worden sein könnten (Bild 4).

((Listingkasten))

Listing 5: Check des Dateisystems

$dir = dir('./entpackt');

while (($datei = $dir->read()) !== false) {

if ($datei != '.' && $datei != '..') {

echo htmlspecialchars($datei) . '<br />';

}

}

$dir->close();

echo '</p>';

((Ende Listingingkasten))

Ver-/Entschlüsseln

Die ursprüngliche Intention der Filter-Klasse von Zend Framework war es, Sicherheitsfeatures anzubieten. Wir gerade gesehen ist die Funktionalität mittlerweile allgemeiner einsetzbar. Einige der Klassen unterhalb von *Zend\_Filter* tangieren jedoch durchaus sicherheitsrelevante Themen. Zwei davon stellen die folgenden Codebeispiele vor: *Zend\_Filter\_Encrypt* und *Zend\_Filter\_Decrypt*. Wie der Name bereits andeutet können damit Dateien ver- und wieder entschlüsselt werden. Die Implementierung der Klasse basiert dabei im Wesentlichen auf anderen PHP-Erweiterungen oder –Funktionalitäten. Prinzipiell gibt es zwei technische Ansätze für die Datenverschlüsselung bzw. –entschlüsselung, die von *Zend\_Filter\_En*/*Decrypt* unterstützt werden: Die *mcrypt*-Erweiterung von PHP und die *OpenSSL*-Erweiterung von PHP. Im Folgenden werfen wir einen Blick auf erstere Option. Die HTML-Oberfläche unseres Beispiels ermöglicht es einen Text zu verschlüsseln:

<form method="post" action="">

Text: <input type="text" name="text" /><br />

Schlüssel: <input type="text" name="key" /><br />

<input type="submit" value="Verschlüsseln" />

</form>

Mcrypt benötigt drei Informationen für die Verschlüsselung: Zunächst einmal die Zeichenkette, die zu verschlüsseln ist. Zweitens benötigen wir ein geheimen Schlüssel, und drittens einen Vektor der bei der Ver- und Entschlüsselung zum Einsatz kommt. Sie können im Code den Vektor selbst auch angeben (die Methode heißt *setVector()*), aber einfacher ist es wenn Sie das Zend Framework selbst einen zufälligen Vektor bestimmen lassen. Wichtig dabei: die Methode *getVector()* liefert diesen Vektor zurück. Ohne diesen Vektor kann die Verschlüsselung nicht mehr rückgängig gemacht werden.

Im Codeschnipsel in Listing 6 sehen Sie die Funktionsweise des Filters, die ganz ähnlich zu den anderen Filter-Klassen ist. Im Konstruktor werden per Arrays ein paar Informationen angegeben, beispielsweise der geheime Schlüssel. Auch die zu verwendende Verschlüsselungsmethode kann dort angegeben werden; wir setzen alternativ auf die Methode *setAdapter()*. Anschließend gibt der Code nicht nur das Ergebnis der Verschlüsselung aus, sondern auch den verwendeten Vektor – wir wollen ja später noch einmal an unsere Daten heran.

((Listingkasten))

Listing 6: Funktionsweise des Filters

if (isset($\_POST['text']) && is\_string($\_POST['text']) &&

isset($\_POST['key']) && is\_string($\_POST['key'])) {

require 'Zend/Filter/Encrypt.php';

$text = $\_POST['text'];

$filter = new Zend\_Filter\_Encrypt(array('key' => $\_POST['key']));

$filter->setAdapter('mcrypt');

$vector = $filter->getVector();

echo 'Ergebnis: ' . $filter->filter($text) . '<br />';

echo 'Verwendeter Vektor: ' . $vector;

}

((Ende Listingingkasten))

Die HTML-Oberfläche in Listing 7 wird uns dabei behilflich sein, wieder die ursprünglichen Daten zu erhalten: ein Benutzer kann die verschlüsselte Zeichenketten, den verwendeten Schlüssel und natürlich den Vektor eingeben:

((Listingkasten))

Listing 7: Formular für Abfrage

<form method="post" action="">

Verschlüsselter Text: <input type="text" name="text" value="<?php

echo htmlspecialchars($text);

?>" /><br />

Schlüssel: <input type="text" name="key" /><br />

Vektor: <input type="text" name="vector" value="<?php

echo htmlspecialchars($vector);

?>" /><br />

<input type="submit" value="Entschlüsseln" />

</form>

((Ende Listingingkasten))

Die zugehörige Klasse im Zend Framework heißt *Zend\_Filter\_Decrypt*. Das weitere Prozedere kennen Sie jetzt ja: Klasse instanziieren, Adapter angeben, Vektor angeben, und filtern (Listing 7).

((Listingkasten))

Listing 7: Entschlüsseln

require 'Zend/Filter/Decrypt.php';

$text = $\_POST['text'];

$vector = $\_POST['vector'];

$filter = new Zend\_Filter\_Decrypt(array('key' => $\_POST['key']));

$filter->setAdapter('mcrypt');

$filter->setVector($vector);

echo 'Ergebnis: ' . htmlspecialchars($filter->filter($text));

((Ende Listingingkasten))

Bei Verwendung von OpenSSL ändern sich einige der Rahmenparameter. Statt Schlüsseln und Vektoren benötigen Sie Schlüsseldateien und eine Passphrase. Letzten Endes läuft es aber wie zuvor ab: Sie geben ein paar Daten und Informationen an und rufen abschließend die Methode *filter()* auf.

Fazit

Das Zend Framework liefert über zwei Dutzend verschiedene Filterklassen mit – und dabei sind Einzelfilter wie etwa die für die verschiedenen Komprimierungsformate gar nicht erst mitgezählt. Der Ordner *Filter* im Zend Framework enthält über 50 Dateien.

((Autorenkasten))

((autorenbild.jpg))

Autor

**Christian Wenz** ist Autor, Trainer und Berater mit den Schwerpunkten Webentwicklung und Webtechnologien. Er bloggt sporadisch unter

www.hauser-wenz.de/blog/

((Ende Autorenkasten))

((Kasten))

Links zum Thema

Download des Zend Framework und zugehöriger Dokumentation

framework.zend.com/

Dokumentation zu *Zend\_Filter*

framework.zend.com/manual/en/zend.filter.html

PHP-Handbuchseite zur mcrypt-Erweiterung

php.net/mycrpt

PHP-Handbuchseite zur OpenSSL-Erweiterung

php.net/openssl

((Ende Kasten))

((Bildunterschriften))

((0406\_ framework10.png))

**Brandneu:** Version 1.10.0 des Zend Framework erschien Ende Januar 2010 (Bild 1)

((0406\_bz-string.png))

**Fettkur:** Bei kleinen Daten ist das Komprimierungsergebnis zu groß Bild 2

((0406\_bz-file.png))

**Kleindatei:** Das Archiv enthält tatsächlich die eingegebenen Informationen (Bild 3)

((0406\_fileupload.png))

**Ausgepackt:** Die Archivinhalte wurden extrahiert (Bild 4)