Einführung in PHPs PEAR

Beitrag zum Chemnitzer Linuxtag 2003

Alexander Merz (alexmerz@php.net)

2. März 2003

Inhaltsverzeichnis

1 Copyright

2 Hintergründe und Geschichte

2.1 Was ist PEAR?

Das **PHP Extension and Application Repository** ist eine Bibliothek für Klassen, Erweiterungen und Anwendungen in PHP; kombiniert mit einer Infrastruktur und Werkzeugen zu ihrer Verteilung und Betreuung.

2.2 Wozu PEAR?

Mit dem wachsenden Erfolg von PHP wuchsen die Ansprüche der Anwender und die Anwendungsfelder. Komplexere Lösungen konnten nicht mehr einfach mit der Anreihung der eingebauten Befehle gefertigt werden.

Also begannen PHP-Programmier Klassen zu schreiben, um auch in solchen Fällen die Programmierung zu vereinfachen. Viele stellten ihre Klassen der Öffentlichkeit zur Verfügung, häufig in Skriptarchiven oder auf der eignen Homepage.

Die PHP Group erkannt Ende 1999, dass ein zentrales Archiv analog zu Perls CPAN und TeX CTAN ein wichtiges Instrument für den Erfolg von PHP darstellt. Verstärkt wurde diese Erkenntnis durch den Erfolg von Java und seiner Java Foundation Classes.

Neben der Einrichtung eines CVS-Repositorys für PEAR, begann der Aufbau der Webseite und die Entwicklung der PEAR Coding Standards.

Die Weiterentwicklung von PHP 4 ging seit dieser Zeit stetig voran; gleichzeitig wurden die Abstände zwischen den Releases neuer Versionen immer länger – bis hin zu 4 Monaten selbst bei kleineren Überarbeitungen. Der Grund lag in der engen Verknüpfung der Entwicklungszyklen des PHP-Kerns selber und seiner Erweiterungen.

Da jedes PEAR-Package über seinen eignen Entwicklungszyklus verfügt, nahm man dieses Modell als Vorlage auch für PHP-Erweiterungen; es entstand als Untergruppe von PEAR die **PHP Extension Code Library**, kurz **PECL**. Hatte PECL ursprünglich den Ruf einer „Verbannung nach Siberien“, so ist doch für PHP 5 geplant ein Gros der derzeitigen Erweiterungen nach PECL auszulagern. Dies wird die Entwicklung von PHP 5 deutlich beschleunigen.

2.3 Was ist ein Package?

Ein Package ist eine Zusammenfassung von Klassen und zugehöriger Daten für einen bestimmten Zweck. Sie bieten ein **Application Programming Interface** (API) an. Ein **Package-Archiv** enthält alle notwendigen Dateien, damit Sie diese API in ihren Programmen zu verwenden.

Packages sind organisiert in Kategorien und Unter-Kategorieren entsprechend ihrem Einsatzgebiet, z.B.: „Net“ für Netz-spezifische Aufgaben, „HTML“ für die Erzeugung und Verarbeitung von HTML usw.

3 Verwaltung der PEAR-Packages

Zentraler Dreh- und Angelpunkt für die Verwaltung von PEAR-Packages auf dem eigenen Computer/ Server ist der **PEAR Package Manager**, üblicherweise als **PEAR Installer** bezeichnet. Es ist ein Kommandozeilen-Programm, geschrieben in PHP, deshalb lauffähig auf jedem Computer, für das PHP zur Verfügung steht.

Neben dem Programm für die Kommandozeile, stehen zur Verfügung mittlerweile ein Installer als Web-Anwendung und mit grafischen Frontend auf PHP-GTK-Basis. Sie setzen nicht auf die Kommandozeilen-Version auf, sondern nutzen direkt die API der PEAR-Installer-Klassen.

3.1 Der PEAR-Installer

(Die nachfolgenden Befehle sind auf der Kommandozeile einzugeben.)

3.1.1 Hilfe!

Hilfe erhalten Sie mit

```
pear help
pear help <befehl>
```

Der erste Aufruf zeigt eine Liste der verfügbaren Befehle. Dieselbe Liste erscheint bei der Angabe von `pear` ohne weiter Parameter. Mit dem zweiten Aufruf erhalten Sie eine Übersicht der Optionen für einen bestimmten Befehl.

3.1.2 Konfiguration

Bevor Sie Installer verwenden, werfen Sie einen Blick auf seine Konfiguration:

```
pear config-show
```

Die wichtigsten Einstellungen betreffen die Pfad-Einstellungen – die erste Einträge in der angezeigten Liste. Sie müssen selbstverständlich auf existierende Verzeichnisse zeigen. Der `prefered_state` kann entweder auf `stable`, `beta`, `alpha` oder `devel` stehen. Dieser Wert spielt eine grosse Rolle bei der Installation und Upgrade von Packages. Der PEAR-Installer benötigt für einige Aufgaben Kontakt mit einem Package-Server – die URL des Servers erscheint unter `master_server`.

Um die Einstellungen zu ändern, verwenden Sie:

```
pear config-set <einstellung> <neuerWert>
```

z.B.: `pear config-set prefered_state beta`

3.1.3 Installation

```
pear install [optionen] <package>
```

Der Installer unterstützt drei Wege um ein Package zu installieren:

- vom Package-Server

  ```
  pear install [optionen] Package_Name
  ```

- einer lokalen Package-Datei

  ```
  pear install [optionen] Package_Name.0.4-beta.tgz
  ```

- von einer beliebigen URL

  ```
  pear install [optionen] http://www.example.com/Package_Name.0.4-beta.tgz
  ```

Bei der ersten Variante hängt es vom `prefered_state` ab, welche Version eines Packages vom Server installiert wird. Ist dieser `stable`, wird die aktuelle stable-Version installiert, existiert keine versucht der Installer die aktuelle beta-Version zu installieren usw.

Die wichtigsten Optionen für die Installation:

- `-f, -force`

  Das Package wird installiert – auch wenn bereits eine neuere Version vorhanden ist.

- `-n, -nodeps`

  Das Package wird installiert – auch wenn bestimmte Abhängigkeiten nicht erfüllt sind.

- `-r, -register-only`

  Das Package wird in der PEAR-Registry vermerkt, ohne es tatsächlich zu installieren.

- `-s, -soft`

  Wenn das Package bereits in einer neueren oder gleichen Version vorhanden ist, wird die Installation ohne weitere Kommentare abgebrochen.

- `-B, -no-build`

  Wird eine PECL-Erweiterung installiert, werden die Quelltexte zwar entpackt, aber die Erweiterung nicht aus ihnen erzeugt.

- `-Z, -nocompress`

  Bei der Installation direkt vom Package-Server wird das Package-Archiv nicht komprimiert.

- `-R verzeichniss, -installroot=verzeichniss`

  Falls ein anderes als das eingestellte Installationsverzeichniss verwendet werden soll

3.1.4 Neue Versionen

Neue Versionen spielen Sie ein mit:

```
pear upgrade [optionen] <package>
```

Es gelten die selben Optionen wie bei der Installation, analog die Angabe der Quelle für das Package.

Auch hier ist die Einstellung des `prefered_state` beim Bezug vom Package-Server von Bedeutung. Steht er auf stable und wird ein Upgrade versucht, schlägt er fehl, wenn nur eine neue beta-Version bereitsteht.

3.1.5 Entfernung

```
pear uninstall Package_Name
```

3.2 Package-Informationen

3.2.1 Lokal

Der PEAR-Installer hinterlegt die Daten eines installierten Packages in der *PEAR-Registry*

Eine Übersicht der Packages erhalten Sie mit:

```
pear list
```

Die Daten eines bestimmten Packages mit:

```
pear info <Package_Name>
```

Die Informationen umfassen u.a. eine kurze Beschreibung, die Version, die Lizenz und den Betreuer.

3.2.2 Remote

Die nachfolgenden Befehle werden mit Hilfe von XML-RPC an einen Package-Server weitergeleitet. Welcher ist abhängig von der Einstellung `master_server` der Installer-Einstellungen. Der Eintrag muss auf einem Server verweisen und es muss eine Verbindung zum Server bestehen.

Die ersten beiden Befehle funktionieren wie ihre lokalen Pendants:

```
pear remote-list
pear remote-info <Package_Name>
```

`remote-list` listet alle auf dem Server verfügbaren Packages auf; `remote-info` die Daten eines einzelnen Packages.

Daneben existieren noch die beiden Befehle `list-all` und `list-upgrade`. `list-all` zeigt alle bereitgestellten Packages einschliesslich Versionsnummer auf dem Server an und ob das Package bereits installiert ist – wenn ja auch dessen Versionsnummer. Der zweite Befehl `list-upgrade` funktioniert ähnlich, zeigt aber nur jene an, die bereits installiert sind und eine andere Versionsnummer/Status aufweisen.

3.2.3 Packages suchen

Neben der Möglichkeit nach Packages auf einer entsprechenden Webseite zu suchen, können Sie dazu auch den Installer nutzen. Der Befehl `pear search` startet eine Abfrage auf dem Package-Server. Zwei Suchvarianten werden unterstützt: Suche nach einem Package-Namen und Volltext-Suche nach einem Begriff in einer Package-Beschreibung.

Die Befehlssyntax:

```
pear search <package_name> <volltext>
```

Die Suche nach einem Package anhand eines Teil seines Namens:

```
pear search URL
```

Damit suchen Sie alle Packages, deren Namen „URL“ enthält.

Um einen Begriff zu finden, verwenden Sie folgenden Aufruf:

```
pear search * server
```

Es werden alle Packages aufgelistet, die den Begriff „server“ enthalten. Die Suche erfolgt unabhängig von Gross- und Kleinschreibung.

Sie können die Package- und Begriffssuche natürlich kombinieren:

```
pear search URL server
```

Beachten Sie: beide Begriffe werden *oder*-verknüpft; es werden alle Packages gefunden, die „URL“ im Namen enthalten oder „server“ in der Beschreibung.

3.3 Die grafischen Installer

Neben dem Installer für die Komandozeile stehen zwei GUI zur Verfügung; einer auf Basis von PHP-GTK und ein zweiter in Form eines Web-Frontends. Beide müssen zuerst installiert werden:

```
pear install PEAR_Frontend_Gtk
pear install PEAR_Frontend_Web
```

Der Gtk-Installer erfordert die Gtk-Version des PHP-Interpreters [1]. Er wird mit `pear -g` gestartet. Der Web-Installer erfordert zum Start ein angepasstes Skript für den Webserver; ein Beispiel-Skript findet sich im PEAR-Doc-Verzeichniss. Achten Sie darauf, den Installer mit einem Authentifizierungs-Mechanismus zu versehen – z.B. PEAR::Auth [2].

[1] http://gtk.php.net

[2] http://pear.php.net/package-info.php?pacid=2

3.4 Der Out-of-the-Box-Installer

Der oben erwähnte Web-Installer ist ideal für die Fern-Verwaltung von PEAR, insbesondere wenn kein Zugriff auf die Kommandozeile besteht. An sich praktisch bei der Pflege einer eigenen PEAR-Installation bei Massenhostern. Nur wie bringt man den Web-Installer unter, wenn eben die Installation per Komandozeile nicht funktioniert?

Die Antwort ist der Out-of-the-Box-Installer [3] von Christian Dickmann. Damit erhalten Sie eine PEAR-Basis-Installation inklusive Web-Installer. Nach dem Entpacken des Archivs und Verschieben des Verzeichnisses auf dem Webserver können Sie sofort loslegen.

3.5 Das Include-Path-Problem

Vorrausetzung für die problemlose Anwendung der PEAR-Klassen ist ein korrekt gesetzter Include-Path; er muss den Pfad zur PEAR-Verzeichniss beinhalten.

- Besteht voller Zugriff auf den Computer, so setzen Sie ihn in der `php.ini`, z.B.:

  ```
  include_path = ".:/usr/lib/php"
  ```

- Wenn diese Möglichkeit nicht besteht, können Sie versuchen – unter dem Apache Webserver – den Pfad in einer `.htaccess`-Datei zu setzen:

  ```
  php_value include_path .:/usr/lib/php
  ```

- Die dritte Variante ist die Zuweisung im Programm selber mit `ini_set()`:

  ```
  <?php

  ini_set("include_path",".:/usr/lib/php");

  require_once 'PEAR.php';
  require_once 'DB.php';
  ...
  ```

4 Anwendung

Einer der grossen Vorteile von PEAR sind seine *Codings Standards*. Sie schreiben u. A. vor wie der Quellcode aussieht, wie Funktionen benannt werden müssen und wie mit Fehlern umgegangen wird.

4.1 PEAR und PECL einbinden

4.1.1 PEAR-Packages

PEAR-Packages bestehen aus PHP-Klassen. Sie können Sie mit `require_once()` einbinden. Die notwendige Pfad- und Dateiangabe ergibt sich aus den Package-Name. Als Grundregel gilt: die Unterstriche im Namen werden durch Schrägstriche ersetzt und die Endung „.php“ angehängt.

Zum Beispiel, die Einbindung des Packages „Net_Ping“ geschieht mit

```
require_once 'Net/Ping.php';
```

[3] http://dickmann.homeunix.org/pear/

4.1.2 PECL-Erweiterungen

Bei PECL-Erweiterungen handelt es sich technisch um die gleichen Erweiterungen, wie jene, die bereits der Installation von PHP mitgeliefert werden - und werden deshalb genauso eingebunden. Entweder tragen Sie den Namen der Erweiterung in die `php.ini` ein:

```
// für Unix:
extension=erweiterung.so
// Windows
extension=erweiterung.dll
```

oder Sie laden die Erweiterung zur Programmlaufzeit: `dl('erweiterung.so');` bzw.
`dl('erweiterung.dll');`

4.2 Funktionsbezeichnungen

Für die Benennung der Klassen in PEAR gilt: *Kategorie_Klassenname* („Net_Ping“), eventuell zusätzlich eine Unterkategorie („HTML_Template_IT“). Der Name der Hauptklasse ist gleichzeitig der Name des gesamten Packages.

Die Bildung der Funktionsnamen erfolg nach dem Prinzip: erstes Wort komplett kleingeschrieben, bei alle weiteren wird der erste Buchstabe gross geschrieben. Z.B.: `connect()`, `toString()` oder `setCurrentBlock()`.

Für identische Funktion wird der gleiche Funktionsname benutzt. Ein Verbindungsaufbau wird immer mit `connect()`, die Trennung mit `disconnect()` durchgeführt; eine Rückgabe von HTML-Code erfolgt immer mit `toHTML()`, usw.. Dieser Standard wird erst seit kurzem forciert, Ausnahmen von dieser Regel sind leider noch zu erwarten.

4.3 Objektorientierung

4.3.1 Objekt-Methoden und statische Methoden

PEAR besteht aus Klassen; alle Funktionen sind Methoden von Klassen. Ursprünglich war die Kapselung in Klassen notwenig, da PHP 4 keine Möglichkeiten bot, zu verhindern, das eigene Funktionsnamen mit anderen kollidieren. Mit der Zeit aber wurden die Vorteile einer durchgängigen Objektorientierung in PEAR deutlich. Trotzdem merkt man diese Mischung der Philosopien noch deutlich an.

Die meisten Methoden sind Objekt-Methoden; es ist notwendig ein Objekt von einer Klasse zu erzeugen, um die Methode zu nutzen:

```
smtp = new Net_SMTP("smtp.example.com");
$smtp->connect();
```

Einige Packages bieten auch statische Methoden an – es ist kein konkretes Objekt notwenig. Der entsprechende Syntax in PHP für statische Aufrufe ist:

```
Klassenname::Methode(...)
```

z.B.:

```
DB::connect(...);
```

Einige Methoden lassen sich auf beide Arten ansprechen. Zu beachten ist nur, dass die Parameteranzahl varieren kann – abhängig von der Art des Aufrufs.

4.3.2 Software-Patterns

Zwei Software-Patterns werden in PEAR häufig eingesetzt: das Factory-Pattern und das Singleton-Pattern.

Das Factory-Pattern Anstatt ein Objekt einer Klasse mit `new` direkt zu erzeugen, wird das Objekt durch eine Factory-Funktion erzeugt.

Anwendung findet es z. B. bei der PEAR-Datenbank-Abstraktion PEAR::DB; statt direkt mit einer Klasse für ein bestimmtes DBMS zu arbeiten, sagen Sie der Factory-Funktion (hier `connect()`) was für ein Objekt Sie wünschen:

```
$dsn = "...";
$db = DB::connect($dsn);
```

`$db` enthält nicht etwa ein Objekt der Klasse `DB`, sondern ein Objekt einer Klasse die durch den Parameter `$dsn` bestimmt wird.

Das Singleton-Pattern Dieses Pattern verhält sich wie das Factory-Pattern – mit einem Unterschied: es kann das selbe Objekt mehrmals zurückgeben. Die obige `connect()`-Methode erfüllt auch die Anforderungen an das Singleton-Pattern.

```
$dsn = "...";
$db1 = DB::connect($dsn);
$db2 = DB::connect($dsn);
```

Die Methode wird zweimal mit den selben Parametern aufgerufen; `$db1` und `$db2` enthalten das gleiche Objekt. Es spielt keine Rolle, ob Sie z. B. `$db1->query()` oder `$db2->query()` verwenden.

4.4 Fehlerbehandlung

Eines herausragende Elemente von PEAR ist seine einheitliche Fehlerbehandlung. Damit unterscheidet sich PEAR von anderen Sammlung von PHP-Klassen. Es steht eine Reihe von Methoden zur Verfügung um zu kontrollieren, was geschehen soll, wenn innerhalb einer Methode ein Fehler auftritt und Methoden zum Umgang mit dem Fehler.

4.4.1 Die Fehler-Modi

Was im Fehlerfall passiert, wird durch den eingestellten Fehler-Modus bestimmt. Er wird mit der Methode `PEAR::setErrorHandling()` gesetzt. Sie übergeben als ersten Parameter die Angabe des Fehlermodus in Form einer Konstante; abhängig vom Modus ist ein zweiter Parameter notwendig, der spezifische Optionen enhält.

- `PEAR_ERROR_DIE`

  Das Programm wird im Fehlerfall beendet und die Fehlermeldung wird ausgegeben.

  Als zweiten, optionalen Parameter können Sie eine Zeichenkette übergeben, die beim Beenden ausgegeben wird: `'Fehler: %s'` - das %s wird durch die Fehlermeldung ersetzt

  Beispiel:

  ```
  PEAR::setErrorHandling(PEAR_ERROR_DIE, 'Fehler: %s');
  ```

- `PEAR_ERROR_PRINT`

  Es wird jedesmal die Fehlermeldung ausgegeben, wenn ein Fehler auftritt; das Programm läuft aber weiter.

  Mit dem zweiten Parameter kann ebenfalls die ausgegebene Zeichenkette beeinflußt werden.

  Beispiel:

  ```
  PEAR::setErrorHandling(PEAR_ERROR_PRINT, 'Fehler: %s');
  ```

- `PEAR_ERROR_RETURN`

  Im Fehlerfall wird die Methode beendet und ein Objekt der Klasse `PEAR_Error` wird zurückgegeben.

- `PEAR_ERROR_CALLBACK`

  Es wird im Fehlerfall eine Funktion oder Objektmethode aufgerufen. Jene erhält als Parameter ein `PEAR_Error`-Objekt.

  Der zweite Parameter bestimmt welche Funktion oder Objekt-Methode aufgerufen wird.

  Beispiel:

  ```
  PEAR::setErrorHandling(PEAR_ERROR_CALLBACK, 'meineFunktion');

  // Objekt:
  $fehlerklasse = new meineFehlerbehandlung();
  PEAR::setErrorHandling(PEAR_ERROR_CALLBACK,
                  array('fehlerklasse', 'meineFunktion'));
  ```

- `PEAR_ERROR_TRIGGER`

  Es wird im Fehlerfall ein Fehler erzeugt, als würde er von selber PHP direkt ausgelöst. Ob das Programm beendet und eine Fehlermeldung ausgegeben wird, ist von der `error_reporting`-Einstellung in der php.ini abhängig.

  Der zweite Parameter bestimmt welche Art von Fehler erzeugt wird.

  Beispiel:

  ```
  // Es wird eine Notiz erzeugt
  PEAR::setErrorHandling(PEAR_ERROR_TRIGGER, E_USER_NOTICE);
  // eine Warnung
  PEAR::setErrorHandling(PEAR_ERROR_TRIGGER, E_USER_WARNING);
  // ein Fehler
  PEAR::setErrorHandling(PEAR_ERROR_TRIGGER, E_USER_ERROR);
  ```

In den Beispielen rufen wir die Methode immer statisch auf. Für einige Klassen in PEAR kann diese Methode aber auch als Methode des Objektes aufgerufen werden, z.B. bei PEAR::Mail.

```
$mail = Mail::factory(...)
$mail->setErrorHandling(PEAR_ERROR_DIE);
...
```

Das bewirkt, dass der Fehlermodus nur für dieses Objekt allein gesetzt wird. Welche Klassen dies unterstützen, ist im PEAR-Manual vermerkt.

4.4.2 Das PEAR_Error-Objekt

In den Fällen von `PEAR_ERROR_RETURN`, `PEAR_ERROR_CALLBACK`, `PEAR_ERROR_EXECPTION` steht Ihnen ein PEAR_Error-Objekt zur Verfügung. Es enthält alle Informationen zu einem Fehler, z.B. Fehlercode und -nachricht.

Ob eine Variable ein Fehler-Objekt enthält, prüfen Sie mit `PEAR::isError()`:

```
$db = DB::connect(...)
if(PEAR::isError($db)) {
        // Fehler aufgetreten!
}
```

Einige Packages bringen eine eigene Fehlerklasse mit, diese ist immer von PEAR_Error abgeleitet. Auch diese können mit der Methode überprüft werden. Besitzt ein Package eine eigene Fehlerklasse, implementiert es meist auch ein eigenes `isError()`. Das ist beim PEAR::DB-Package der Fall: es bringt eine eigene Fehlerklasse und eigenes `isError()` mit; wir hätten genauso schreiben können:

```
$db = DB::connect(...)
if(DB::isError($db)) {
        // Fehler aufgetreten!
}
```

Diese Funktion prüft ob ein Objekt nur der Klasse DB_Error vorliegt; dieses Verhalten kann die schnelle Auswertung bei zentraler Fehlerverwaltung vereinfachen.

Unabhängig, ob es sich um ein orginales PEAR_Error-Objekt oder von einer abgleiteten Klasse handelt, stehen die folgenden Methoden zur Verfügung:

Methode	**Rückgabe**
`getMode()`	der Fehlermodus, mit dem das Objekt generiert wurde
`getCallback()`	Name der möglichen Fehlerbehandlungs-Funktion
`getMessage()`	Fehlernachricht
`getCode()`	Fehlernummer
`getType()`	Typ des Fehlers (Name der Fehlerklasse)
`getUserInfo()`	interne Fehlerinformationen
`toString()`	Alle Informationen des Fehlers in einer Zeichenkette

5 Abschluss

ich hoffe Ihnen einen kleinen Einblick in PEAR gegeben zu haben. Werfen Sie doch mal einen Blick auf die PEAR-Webseite: *http://pear.php.net*, dort erhalten Sie umfangreiche Informationen über PEAR.