Das Dreamteam
Pear und Smarty

von Bernd Ott (bernd@quantendrehung.de)

Inhaltsverzeichnis

Einleitung

Damals, als ich jung war, habe ich viele Fehler gemacht; nun bin ich alt und mache noch immer viele Fehler. Aber hoffentlich jeden nur einmal. In diesem Vortrag möchte ich zeigen, wie leicht es ist, dynamische Inhalte auf Webseiten anzuzeigen. Dazu verwende ich PEAR (http://pear.php.net) für den Datenbankzugriff und die Templateengine Smarty (http://smarty.php.net/).

Template was ?

Am Einfachsten demonstriere ich einen Fehler von mir. Als PHP-Neueinsteiger kam ich auf die Idee eine Newspage zu programmieren. Selbstverständlicherweise sollte diese datenbankgestützt sein. Nach viel Mühe war der nachfolgende Code das Ergebnis.

```
<?php
     include("./db.php");
     $link = generic_dbopen();

     $query="SELECT n.dat_datum, n.str_news, n.int_news, n.str_author,
          count(c.int_comment)
          FROM `allstars_news` n left
          join `allstars_news_comments` c on (n.int_news=c.int_news)
          where n.dat_datum<=\"".date("Y-m-d") ."\"
          GROUP BY n.int_news, n.dat_datum, n.str_news, n.str_author
          order by dat_datum DESC, int_news DESC
          LIMIT 0, 10";

      $result = mysql_query($query) or die("Anfrage fehlgeschlagen");

        while ($row = mysql_fetch_array($result, MYSQL_NUM))
        {
           $dat = explode(" ",$row[0]);
           $dat = explode("-",$dat[0]);
           $news_datum=$dat[2].".".$dat[1].".".$dat[0];

           echo "<a name=\"$row[2]\">";
           echo "<table class=\"maintab\" width=\"99%\" border=\"0\" cellspacing=\"0\"
cellpadding=\"2\">";
              printf ("<tr class=\"TD1\"><td><b>%s</b></td><td align=\"right\"><font
size=-2>%s</font></td></tr>", $news_datum, $row[3]);
          printf ("<tr><td colspan=2>%s</td></tr>",stripslashes($row[1]));
              printf ("<tr class=\"TD1\"><td colspan=\"2\"><font size=-2><a
href=\"news_showcomment.php?int_news=$row[2]\">Kommentare lesen ($row[4]) /
schreiben</a></font></td></tr>");
          echo "</table></a><br>\n\n";
        }
        mysql_free_result($result);
        mysql_close($link);
?>
```

Die Idee das Tabellenlayout der Webseite zu ändern führte mich in eine Kriese. Mein WYSIWIG-Editor konnte mit dem Mix aus PHP und HTML nichts mehr anfangen. Auch die Portierung von MYSQL auf Interbase habe ich nicht vorgenommen, da an vielen Stellen im Code native MYSQL-Aufrufe vorhanden waren.

Sebastian Röbke von der PHPUG-Hannover berichtete bereits am 05.2.2002 von der PEAR, daher werde ich im Folgenden nur auf die PEAR DB eingehen.

PEAR

PEAR steht für PHP Extension and Application Repository. PEAR ist ein Framework und ein Installations/Verteilungssystem für wiederverwendbare PHP Komponenten. PEAR DB ist ein allgemeines Framework für Datenbankzugriff. Dieses ist nicht auf eine bestimmte Datenbank wie Interbase festgelegt. Wenn man PEARDB allerdings mit verschieden SQL-Datenbanken nutzen möchte, ist man auf Standart SQL festgelegt.

Datenbank öffnen

Um eine Verbindung mit einer Datenbank herzustellen, muss ein DSN erzeugt werden. Dieser besteht aus Datenbanktyp, Username, Passwort, Server und Datenbankname. Mögliche Datenbanktypen sind MSSQL, MYSQL, ACCESS, DB2 oder INTERBASE.

Beispiel:

```
$datenbank = &DB::connect("mssql://user:passwort@dbserver/datenbankname");
```

Datenmenge öffnen

In der Variable $datenbank ist die Verbindung gespeichert. Diese kann genutzt werden um auf Datenmengen zuzugreifen. Um eine Datenmenge zu öffnen, die an das Smarty übergeben wird, nutze ich meistens den Befehl getAll. Dieser liefert mir alle Datensätze in einen Array zurück. Der Vorteil von getAll ist, das man nicht erst durch die Datensätze interieren muss. Der Parameter DB_FETCHMODE_ASSOC gibt an, dass eine Datenbankrow in einen Assozativen Array gespeichert wird. Das macht später die Nutzung der Daten im Smarty einfacher. In $recordset ist nach dem Aufruf ein Array gespeichert, welches die Tabellenrows enthält. Jede Row ist wieder ein Array, welches die Spalten der Tabelle enthält.

Beispiel:

```
$recordset=&$datenbank->getAll("select * from Tabellenname", array(),
DB_FETCHMODE_ASSOC);
```

Smarty

Smarty ist eine Templateengine. Durch ein einfaches Pluginsystem kann diese schnell und einfach erweitert werden. Hier sind einige Vorteile von Smarty als Liste vorweg:

- Caching – um die Anzeigegeschwindigkeit zu erhöhen und die Datenbanklast zu verkleinern
- Sicherheit – PHP-Code wird von Anzeige getrennt, damit Designer es leichter haben
- Variablen Modifiert – erleichter die Ausgabe von Variablen z.B. für Formatierungen
- Templatefunktionen – erleichtern die Templategestaltung z.B. für Optionslisten, Formatierung
- Filter – pre-filters, post-filters und output-filters
- Resourcen – Templates können durch einen Resourcehandler von verschiedensten Quellen geladen werden
- Plugins – mit wenig Aufwand ohne Ende erweiterbar

Die Klasse nutzen

Zunächst brauchen wir einen „require" um die Klassendatei einzubinden. Als Nächstes instanzieren wir die Klasse. Abschließend müssen noch drei Einstellungen vorgenommen werden.

```
require 'lib/smarty/Smarty.class.php';

// hier wird die instanz erzeugt
$smarty = new Smarty;
// compilecheck an, damit die template neu kompiliert wird, wenn wir noch entwickeln
$smarty->compile_check = true;
// debugfenster einschalten, damit wir sehen was die templateengine macht.
$smarty->debugging = true;
// caching ausschalten, da wir in der entwicklung immer das aktuelle sehen wollen
$smarty->caching = false;
```

Als Standardeinstellung nutzt Smarty geschweifte Klammern um Tags/Platzhalter zu markieren. Da aber in Javascript auch geschweifte Klammer vorkommen, empfielt es sich, die Delimiter in andere umzuschalten. Die Delimiter können auf fast beliebige Zeichen(-folgen) umgestellt werden.

```
$smarty->right_delimiter = "*}";
$smarty->left_delimiter = "{*";
```

Das erste Template

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
     <head>
          <meta http-equiv="content-type" content="text/html;charset=iso-8859-1">
     </head>

     <body>
          Hallo Welt ich bin ein {*$platzhalter*}.
     </body>
</html>
```

Um das Template nun anzeigen zu können, muss die Methode „Display" aufgerufen werden. Diese compiliert das Template in ein PHP Script, welches anschließend durch das Smarty auf dem Response ausgegeben wird.

```
$smarty->display('beispiel.html');
```

Ist eine Ausgabe nicht gewüscht, kann stattdessen „fetch" aufgerufen werden. Als Rückgabewert gibt es den Smartyoutput in eine Variable. Das kann z.B. für automatisch generiert Emails genutzt werden.

```
$email = $smarty->fetch('beispiel.html');
```

Beide Methoden haben zwei weitere optionale Parameter. Der Erste ist die Cache ID. Der Zweite die Compile ID. Die Cache ID lässt sich nutzen um die Seite in einen bestimmten Kontext zu Cachen.

Tags ersetzen

In unserem Beispiel Template haben wir ein Tag „{*platzhalter*}, welches wir mit Datenbankinhalt oder PHP Variablen ersetzen wollen.

```
$smarty->assign("platzhalter", "Berliner");
```

Das „Assign" wird im Code vor „Display" und "Fetch" aufgerufen.

Weitere Templates einbinden

In Templates lassen sich weitere Templates einbinden. Diese kann man z.B. nutzen um einen einheitlichen Header und Footer auf die Page zu bekommen.

```
{include file="header.html"}
```

Die obere Zeile bindet das Template „header.html" ein. Das einzubindene Template kann aber auch durch eine Variable bestimmt werden.

{include file=$headerdatei}

Diese muss dann aber vor der Ausgabe

```
$smarty->assign("headerdatei", "header.html");
```

zugewiesen werden.

Smarty Plugins nutzen

Es gibt verschiedene Arten von Plugins für Smarty. Die meist verwendeten sind wohl die Variablen-Modifikatoren. Mit Hilfe von Plugins kann man die Ausgabe der Variable auf der Page anpassen, Variableninhalte in Grossbuchstaben umwandeln, die Ausgabe verkürzen oder ein Datum in einem bestimmten Format anzeigen.

Beispiel:

```
{*$tagesdatum|date_format*}
{*$tagesdatum|date_format:"%A, %B %e, %Y"*}
```

AUSGABE:

```
Feb 6, 2001
Tuesday, February 6, 2001
```

Einfache Entscheidung

In Smarty ist es möglich einfache IF THEN ELSE Konstruktionen zu nutzen. Dies ist sehr nützlich, wenn Adminfunktionen versteckt werden sollen oder bestimmte Webseitenelemente ausgeblendet werden sollen.

```
{*if $elementanzeigen == "1"*}
  und hier ist der wichtige text !!
{*else*}
  hier wird nix angezeigt.
{*/if*}
```

Smarty Sektionen

Da es sehr häufig vor kommt, dass eineTabelle angezeigt oder einfach ein Array ausgeben werden muss, besitzt Smarty die Sektionen. Wenn in einer Variable ein Array gespeichert ist, wird mit Hilfe der Sektion eine Schleife gebildet. Wenn das Array leer oder nicht zugewiesen ist, kann man einen Erstatztext ausgeben.

```
wir haben diese news hier :
{*section name=aktnews loop=$newsarrayvariable*}
  {*$newsarrayvariable[aktnews]*}<br>
{*sectionelse*}
  es sind leider keine news vorhanden<br>
{*/section*}
```

Charmed ! Die Macht der 2

Um Worte zu sparen, liefere ich hier Code, welcher die Anzeige einer „News"-Seite bewirkt. Zunächst das Template:

(termine.html)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
            <head>
                        <meta http-equiv="content-type" content="text/html;charset=iso-8859-1">
            </head>

            <body>
                {*include file="header.html"*}
                        <table border="0" cellspacing="2" cellpadding="0">
                        {*section name=termin loop=$termine*}
                                    <tr>
                                                <td>{*$termine[termin].datum*}</td>
                                                <td>{*$termine[termin].text*}</td>
                                    </tr>
                        {*sectionelse*}
                                    <tr><td colspan="2">leider keine news vorhanden</td></tr>
                        {*/section*}
                        </table>
                        {*include file="footer.html"*}
            </body>
</html>
```

Und nun der PHP-Code:

(index.php)

```
<?php
require_once 'lib/pear/DB.php';

$db = DB::connect('mssql://user:passwort@dbserver/datenbankname');
if (DB::isError($db)) {
    die ($db->getMessage());
}
$recordset=&$datenbank->getAll("select datum, text from termine order by 1", array(), DB_FETCHMODE_ASSOC);

require 'lib/smarty/Smarty.class.php';

$smarty = new Smarty;
$smarty->compile_check = true;
$smarty->debugging = false;
$smarty->caching = false;
$smarty->right_delimiter = "*}";
$smarty->left_delimiter = "{*";

$smarty->assign('termine', $recordset);
$smarty->display('termine.html');
?>
```

Caching – der letzte Schliff

Ich möchte hier noch das Caching bemerken. In dem vorherigen Beispiel wird bei jedem Zugriff auf das Script die Seite neu aus der Datenbank generiert. Um die Geschwindigkeit zu erhöhen, besitzt Smarty mehrere Funktionen, die das Caching unterstützen. Als Erstes müssen wir das Caching aktivieren.

```
$smarty->caching = true;
```

Dann stellen wir den Datenbankcode so um, dass diese nur abgefragt wird, wenn die Seite nicht mehr gecached ist.

```
<?php
require_once 'lib/pear/DB.php';

require 'lib/smarty/Smarty.class.php';

$smarty = new Smarty;
$smarty->compile_check = true;
$smarty->debugging = false;
$smarty->caching = true;
$smarty->right_delimiter = "*}";
$smarty->left_delimiter = "{*";

if (!$smarty->is_cached("termine.html")) {
      $db = DB::connect('mssql://user:passwort@dbserver/datenbankname');
      if (DB::isError($db)) {
         die ($db->getMessage());
      }
      $recordset=&$datenbank->getAll("select datum, text from termine order by 1", array(),
DB_FETCHMODE_ASSOC);
      $smarty->assign('termine', $recordset);
}

$smarty->display('termine.html');
?>
```

So einfach ist das.

Als letzte Anregung möchte ich euch noch mitgeben, dass Smarty nicht auf HTML-Seiten beschränkt ist. Ebenso gut lässt sich auch XML erzeugen.

Linkliste

Smarty	http://smarty.php.net/
Smarty Crashcourse	http://smarty.php.net/crashcourse.php
Smarty deutsche Anleitung	http://smarty.php.net/manual/de/
Smarty FAQ	http://smarty.incutio.com/?page=SmartyFrequentlyAskedQuestions
PEAR	http://pear.php.net/
PEARDB	http://pear.php.net/package/DB
Einführung in PEAR	http://www.sebastian-r.de/phpug/pear.pdf
Quick Start Guide to Pear DB	http://vulcanonet.com/soft/?pack=pear_tut

Über den Autor

Bernd Ott ist Jahrgang 1975 und seit Mitte der achtziger Jahre ist er im Bereich der Softwareentwicklung tätig. Er arbeitet seit 1995 als festangestellter Softwareentwickler mit den Programmiersprachen Delphi, PHP, Java und C. Seit 1999 betreibt er nebenberuflich ein Einzelunternehmen. Dieses beschäftigt sich mit Unternehmensberatung, Prozesstechnik, Lichttechnik und DTP. In seiner verbleibenden Freizeit ist er Diskjockey (house, techno musik).