0

PHP Inside

25 02 2004

Русский журнал web разработчика

Стандарты кодирования на PHP

Дополнительные возможности JpGraph

JpGraph – объектно-ориентированная PHP-библиотека, позволяющая достаточно просто создавать высококачественную профессиональную графику

Они делают это так

Интервью с Президентом New York PHP – Хансом Заунером, специально для нашего журнала

PHP5 - не революция, а эволюция

Расмус Лердорф для Linuxworld

Программирование для WEB

Содержание

- JpGraph и русский язык ……..3
- Дополнительные возможности JpGraph ……..…5
- Они делают это так ……..…..20
- Лердорф: Будущий релиз PHP «эволюционен, а не революционен» ……..…..22
- Стандарты кодирования PEAR……...…..23
- Стандарты кодирования на PHP…..……...…….28
- Примеры использования PEAR…40
- Введение в FPDF…..…….…….45
- Работа с базами данных ……54

Редакция

Вот и вышел в свет сигнальный номер журнала о PHP на русском. В лучших традициях программирования ему присвоен номер 0. Его дальнейшая судьба пока неизвестна, но хочется верить, что массив, первым элементом которого он является, будет поистине большим.

В первую очередь хочется сказать огромное спасибо всем тем, кто так или иначе принял участие в создании этого номера – авторам и переводчикам, критикам, вдохновителям и ребятам, подавшим большое количество ценных идей. Спасибо PHPClub Team за поддержку проекта (надеемся на продолжение). Мы хотели создать хороший электронный журнал всем сообществом и распространять его бесплатно, чтобы делиться опытом и учиться, чтобы продвигать технологию в массы и не отставать от заокеанских коллег, в распоряжении которых имеются php magazine и php|architect.

Этот номер был создан энтузиастами, сумевшими найти свободное время и подготовить материалы, поэтому прошу не судить нас строго, если что, в следующий раз мы постараемся сделать лучше. Всех остальных просим присоединяться к проекту и поделиться своим опытом и знаниями – они наверняка пригодятся другим. Журналу могу пожелать – В ДОБРЫЙ ПУТЬ!

Адреса для контактов с журналом: biznw@rambler.ru
project@yugovostok.ru

Присоединяйтесь к проекту!

JpGraph и русский язык

Графика

Материал с сайта http://detail.phpclub.ru
Оригинал статьи доступен по адресу http://blog.redgraphic.ru/as/06-08-03_78/
Автор: Александр Шиляев

Те, кто сталкивался с необходимостью создания различных графиков из PHP, наверняка знают библиотеку JpGraph (http://www.aditus.nu/jpgraph/). Для тех, кто не знает, JpGraph - это ОО библиотека на PHP, применяемая для построения различных графиков. Сверх дизайнерского решения на ней не построишь, но для того, чтобы быстро и с минимумом кода автоматически сформировать график в стиле excel, вполне сгодится. Требует установленной библиотеки GD (http://www.boutell.com/gd/).

Но, как это часто бывает у буржуев, написать что-нибудь они напишут, а вот сделать, чтобы можно было писать на великом и могучем - с этим проблемы. В том же JpGraph даже есть некоторые настройки типа:

```
// Special unicode language support

DEFINE("LANGUAGE_CYRILLIC",true);

// If you are setting this config to
true the conversion

// will assume that the input text is
windows 1251, if

// false it will assume koi8-r

DEFINE("CYRILLIC_FROM_WINDOWS",true);
```

Как эти настройки не изменяй - ничего не происходит. Либо все выводится в виде кракозябр, либо в юникоде типа &#xxxx. Оказывается, не все так просто. Нужен еще шрифт с кириллицей. Библиотека JpGraph (хотя больше не она, а GD) может работать со стандартными TTF-шрифтами. А это очень хорошо, когда отчет с графиками может быть оформлен не в виде стандартных и набивших оскомину пиксельных шрифтов из unix'a, а с помощью стандартных и набивших оскомину шрифтов из windows'a. Так, во всяком случае, привычнее. Но сначала надо сказать JpGraph'у, где эти шрифты взять.

Для того, чтобы настроить русские шрифты надо:

1. Найти их на сервере. Обычно лежат в директории типа /usr/X11R6/lib/X11/fonts/TrueType/ или /usr/X11R6/lib/X11/fonts/TTF/.
2. Проверить, есть ли там стандартные юникодные шрифты и поддерживают ли они кириллицу (как это сделать - личная проблема каждого, у меня их там вообще не было).
3. Если их там нет или они кириллицу не поддерживают, то пишем их туда, например, из директории Winnt/fonts со своего компьютера.
4. Находим в конфигурационном файле JpGraph строчку типа DEFINE("TTF_DIR", ...) и пишем на место директории ту, где нашли и записали шрифты.

Теперь шрифты есть и JpGraph их должен найти. Для этого в своем графике определяем, например, заголовок как шрифт Verdana (он должен быть на сервере):

```
$somegraph->title->SetFont
(FF_VERDANA);
```

Дальше либо покажется русский заголовок графика, либо он будет не в той кодировке, либо (что вероятнее) JpGraph скажет, что на сервере не установлен компонент работы с TTF-шрифтами (FreeType). Если так, то надо либо искать другой сервер, либо перекомпилировать PHP.

У нас проблем с перекомпилированием (слово-то какое!) нет, поэтому пересобираем PHP с ключом --with-freetype-dir=/usr, где /usr это префикс установки проги freetype,

которая, собственно, и занимается поддержкой этих fontov (сисадмин спешит на помощь).

Собственно все. Если есть проблемы с тем, что название вывелось не в той кодировке (например, в кои, а не в windows), то идем в файл настроек и изменяем строку:

```
DEFINE("CYRILLIC_FROM_WINDOWS",true
);
```

Вот теперь, собственно, все. Шрифты должны работать. Размеры, тип шрифта можно установить там же с помощью метода SetFont(). Но для этого уже есть документация.

Дополнительные возможности JpGraph

Автор: Джейсон Свит (Jason E. Sweat)
Перевод: Владимир Жульев
Оригинал статьи: http://phparch.com/issuedata/articles/article_40.pdf

JpGraph (http://www.aditus.nu/jpgraph/) – объектно-ориентированная PHP-библиотека, позволяющая достаточно просто создавать графику профессионального качества, используя минимум кода. Данная статья представляет собой учебный пример, иллюстрирующий некоторые дополнительные возможности библиотеки JpGraph, а именно:

- общая методика разработки скриптов с использованием JpGraph;
- последовательный процесс разработки графиков (в отличие от простой демонстрации конечного результата);
- использование механизмов кэширования JpGraph для увеличения производительности;
- использование карт-изображений на стороне клиента (далее – CSIM, т.е Client Side Image Map. *Прим. перев.*) для реализации быстрой навигации.

Инсталляция и необходимое программное окружение

Обязательные требования:
Версия PHP: минимум - **4.0.4,** рекомендуется **- 4.1**
Операционная система: любая
Дополнительное ПО: **JpGraph,** поддержка библиотеки **GD** в **PHP**

Для начала работы с JpGraph необходимо скачать исходный код, доступный по адресу:
http://www.aditus.nu/jpgraph/jpdownload.php.
Далее распакуйте содержимое архива в одну из директорий, перечисленных в конфигурационной опции PHP *include_path*. Теперь вы можете изменить пути к директориям установленных в вашей системе шрифтов, а также директории кэша изображений. Эти настройки находятся в файле **jpgraph.php.**

Для проверки правильности инсталляции просмотрите страницу **/Examples/testsuit.php**. Эта страница генерирует более 200 демонстрационных графиков, используя библиотеку JpGraph, и позволяет просмотреть исходный код, использованный для построения каждого из них.
Если вы только изучаете JpGraph и исследуете его возможности, вам потребуется справочное руководство. Оно доступно по адресу http://www.aditus.nu/jpgraph/jpdownload.php и содержит как последовательное описание, так и превосходный справочник по классам. Вы также можете посетить форум поддержки JpGraph – http://jpgraph.fan-atics.com/.

Все скрипты в этой статье были разработаны и протестированы на PHP 4.3.0 (с включенной поддержкой библиотеки GD2), установленном как модуль к Apache 1.3.27 под управлением операционной системы RedHat Linux 7.2.

Скрипты используют СУБД MySQL (http://www.mysql.com) и ADOdb (http://php.weblogs.com/adodb) как абстрактный уровень доступа к базе данных. По этой причине все скрипты включают в себя файл **phpa_db.inc.php**, показанный в листинге 1.

Примечание переводчика:
в английском варианте статьи нет информации об используемой версии библиотеки JpGraph. Здесь и далее информация о путях, конфигурационных опциях, URI приведена для версии 1.14, имеющейся в распоряжении переводчика. Код примеров сохранен оригинальный.

Примеры тестировались на Windows 2000 Server SP4/Apache 1.3.27/PHP 4.3.3 (уст. как модуль к Apache)/JpGraph 1.14/MySQL 4.1-alpha.

```
<?php
error_reporting(E_ALL);
require_once 'adodb/adodb.inc.php';
define('MYSQL_DT_FMT', '%Y-%m-%d');
$conn = &ADONewConnection('mysql');
//$conn->debug=true;
$conn->Connect('localhost', 'phpa',
'phpapass', 'phpa');
$ADODB_FETCH_MODE =
ADODB_FETCH_ASSOC;
?>
```

Листинг 1**: phpa_db.inc.php**

Данный включаемый файл создает объект-соединение ADOdb - **$conn**, который используется для доступа к БД в остальных скриптах. Вам необходимо изменить параметры вызова метода **Connect()** в соответствии с вашими установками, в том числе и имя тестовой базы данных.

Примечание переводчика:
Пример, более понятный для русскоязычного читателя:

```
$conn->Connect('localhost',
'username', 'passwd', 'dbname');
```

ADOdb должна быть установлена в одну из директорий, перечисленных в конфигурационной опции PHP *include_path*. Для читателей, не знакомых с ADOdb, я дам очень краткий обзор некоторых базовых возможностей. В ADOdb вы можете получить результат выполнения SQL-запроса используя следующий вызов:

```
$rs = $conn->Execute($sql_statement,
$bind_array);
```

Если запрос был выполнен удачно, метод вернет объект - набор записей ADOdb, в противном случае метод вернет FALSE. Затем используются два метода этого объекта: **GetArray()** и **GetAssoc()**. Метод **GetArray()** возвращает вектор записей, где каждая запись представлена ассоциативным массивом вида ‘COLUMN’ => ‘VALUE’. Метод **GetAssoc()** вместо вектора возвращает ассоциативный массив, индексы которого являются значениями первого столбца для каждой записи.

Приведенные ниже примеры предполагают, что у вас в системе установлен TrueType шрифт Arial, в противном случае JpGraph будет выдавать ошибку. Самый простой способ обойти эту проблему – заменить все ссылки на константу FF_ARIAL на FF_FONT1. FF_FONT1 – встроенный системный шрифт, он выглядит не так приятно как FF_ARIAL, но позволит вам просмотреть приведенные примеры.
Этого должно быть достаточно, чтобы приступить к примерам. Теперь рассмотрим учебную задачу.

Учебная задача

В нашем учебном примере мы рассмотрим данные о продажах компании ABC, вымышленного производителя украшений. ABC производит широкий спектр украшений от дешевого B за $12.99 до украшений E класса «ультра-делюкс» за $1,499.50. В данном примере мы сосредоточимся на продажах в континентальных Соединенных Штатах, где компания разделена на четыре подразделения по регионам. Компания продает товар распространителям по трем каналам: Интернет, телефонный центр, и различные розничные точки.

Вас попросили сделать графики, отображающие данные о продажах в долларах и натуральных показателях. Вам необходимо также разбить графики по позициям каталога для каждого региона и выполнить сравнение со спрогнозированными данными. Компания ABC также требует, чтобы вы сделали график, показывающий данные о продажах за год по каналам для каждого региона. Им также необходима быстрая навигация между двумя графиками.

Разработка базы данных

Модель данных компании ABC составляют шесть таблиц. Центральная таблица – **abc_sales.** Таблица содержит информацию о продажах, включая время, канал, штат, позицию каталога, количество проданных изделий и доход от сделки.

Другие таблицы, составляющие модель данных - **abc_catalog**, **abc_channel**, **abc_forecast**, **abc_region** и **abc_state_region**. Таблица **abc_catalog** содержит суррогатный ключ для позиций каталога, которые клиент может приобрести, описания позиций и цену за единицу. Таблица **abc_channel** содержит суррогатный ключ, наименование и описание канала продажи (web, телефон, розничная точка), через который может быть куплен товар из каталога. Таблица **abc_forecast** хранит информацию о спрогнозированных планах продаж по позициям каталога, каналам продаж и месяцам. Таблица содержит информацию об ожидаемых объемах продаж и доходах по каждому «срезу» данных. Таблица **abc_region** содержит суррогатный ключ и описание для каждого региона продаж компании. Таблица **abc_state_region** ставит в соответствие аббревиатуру штата конкретному региону продаж.

SQL-запросы, используемые для создания таблиц, а также заполнения небольших таблиц, находятся в файле **mysql_ddl.sql** в директории с исходным кодом данной статьи. Что касается других, то таблицы продаж и прогнозов продаж могут быть заполнены с использованием скриптов **abc_gen_sales.php** и **abc_fcst_ins.php** соответственно. Скрипт **abc_gen_sales.php** должен запускаться первым, так как работа **abc_fcst_ins.php** зависит от данных, которые создает первый скрипт.

Примечание переводчика:
Для создания учебной БД потребуются еще и данные о переписи населения в США за 2000 год.
Эта информация (в виде ASCII-файла) доступна по адресу http://www.census.gov/geo/www/gazetteer/tiger/tms/gazetteer/ustracts2k.zip

Поэтапно процесс создания и заполнения учебной БД выглядит так:

1. Скачать файл с данными переписи, распаковать и разместить его в директории с учебными примерами.
2. **jvn_parse_census.php** (соединение с БД - см. комментарии в скрипте) – парсит файл **ustracts2k.txt** (данные переписи в формате ASCII), создает и заполняет таблицу **census_data,** необходимую для работы скрипта **abc_gen_sales.php.**
3. **mysql_ddl.sql.**
4. **abc_gen_sales.php.**
5. **abc_fcst_ins.php.**

Примечание:
Оригиналы данных скриптов были написаны в декабре 2002 года, данные подобраны из расчета, что текущей датой является 15 декабря 2002 года, чтобы графики выводились практически за полный год. Так как сейчас 2003 год, скрипты были модифицированы таким образом, чтобы запрашивать и выводить данные за предыдущий год.

Примечание переводчика: статья переводилась на русский язык в феврале 2004 года, и в скрипты были внесены необходимые изменения.

Сравнение данных о продажах со спрогнозированными показателями по регионам

Нашей первой задачей будет создание графика, сравнивающего объем продаж и доход с прогнозом для каждого региона. Предполагается, что эта информация является собственностью компании и не предназначена для публичного распространения.

При разработке PHP-скриптов, создающих графики с использованием JpGraph, я предпочитаю использовать следующий 4-х шаговый процесс:

1. Получение и обработка данных для построения графика.
2. Создание объекта Graph и задание его основных свойств (таких как цвет и координатные оси графиков).
3. Создание объектов Plot, для размещения на объекте Graph.
4. Завершение объекта Graph и его вывод графика.

Согласно вышеописанному процессу вы, в первую очередь, должны получить данные о продажах и прогнозные данные. Чтобы посмотреть, как используется ADOdb для получения данных этого графика, посмотрите строки 26-122 скрипта **abc_reg_sales_graph.php.** Цель этой статьи – построение графиков, поэтому вопросы извлечения данных из базы и конструирования графиков детально не рассматриваются.

Поскольку графики строятся по регионам, ваш скрипт должен иметь соответствующий параметр и проверять его корректность.
Если параметр корректен, его значение присваивается переменной **$region_id**, в противном случае генерируется ошибка.

Здесь имеет место небольшая проблема. Так как данные, выводимые скриптом – графический объект, страницы сайта должны ссылаться на него используя HTML-тэг **<img>**, т.е так:

```
<img src="abc_reg_sales_graph.php">
```

```
1  $region_id = check_passed_region('region');
2  if (!$region_id) {
3      graph_error('region parameter incorrect');
4  }
5
6  function check_passed_region( $parm )
7  {
8      global $regions;
9
10     if (array_key_exists($parm,$_GET)) {
11            $val = $_GET[$parm];
12            if (array_key_exists($val, $regions)) {
13                  return $val;
14            }
15     }
16     return false;
17 }
18
19 function graph_error($msg)
20 {
21     $graph = new CanvasGraph(WIDTH, HEIGHT);
22
23     $t1 = new Text($msg);
24     $t1->Pos(0.05, 0.5);
25     $t1->SetOrientation('h');
26     $t1->SetFont(FF_ARIAL, FS_BOLD);
27     $t1->SetColor('red');
28     $graph->AddText($t1);
29
30     $graph->Stroke();
31     exit;
32 }
```

Листинг 2: Создание сообщения об ошибки в графическом виде (abc_reg_sales_graph.php)

Если скрипт выведет текстовое сообщение (например, сообщение об ошибке PHP), это будет понято как некорректное изображение, и все, что увидит пользователь – символ отсутствующего изображения на том месте, где должен быть ваш график. Код, показанный в листинге 2, проверяет переданный параметр «регион» и формирует сообщение об ошибке в графическом виде. Данный код предполагает, что вы уже выбрали из базы данных список регионов продаж и сохранили результат запроса в глобальной переменной-массиве **$regions.** Также предполагается, что в скрипт включены библиотечные файлы **jpgraph.php** и **jpgraph_canvas.php.**

После извлечения из БД данных в виде набора записей часто необходимо преобразовать их в формат, более подходящий для построения графика. Это означает создание нескольких массивов с индексами, начинающимися с нуля для каждого числового ряда.
Вместо того, чтобы создавать несколько глобальных массивов для каждого ряда, я предпочитаю иметь единый ассоциативный массив с именем **$graphData**, содержащий массивы для каждого ряда и имеющий индексы, совпадающие с именами этих рядов. Код в листинге 3 собирает эти массивы в единый массив **$graphData,** который будет использоваться в дальнейшем для построения графиков.

Примечание переводчика: Здесь автор статьи приводит довольно труднопереводимое высказывание, смысл которого сводится к следующему.
Вместо того, чтобы создавать для каждого отображаемого на графике числового ряда отдельный массив, автор предлагает использовать один ассоциативный массив для всего графика.
Т.е., вместо:

```
$qty = array(...);
$f_qty = array(...);
```

использовать:

```
$graphData = array
(
        'qty' => array(...),
        'f_qty' => array(…)
);
```

```
1  $graphData['f_qty'] = array();
2  $graphData['labelX'] = array();
3  for ($i=0,$j=count($salesData); $i<$j; $i++) {
4    $row = $salesData[$i];
5    if ('A'==$row['short_desc']) {
6      $graphData['labelX'][] = strftime('%b', mktime(0, 0, 0, $row['m'], 1, $row['y']));
7    }
8    if (!array_key_exists($row['m']-1, $graphData['f_qty'])) {
9      $graphData['f_qty'][$row['m']-1] = $fcstData[$row['f_key']]['qty'];
10     $graphData['f_rev'][$row['m']-1] = $fcstData[$row['f_key']]['rev'];
11     $graphData['qty'][$row['m']-1] = $row['qty'];
12     $graphData['rev'][$row['m']-1] = $row['rev'];
13   } else {
14     $graphData['f_qty'][$row['m']-1] += fcstData[$row['f_key']]['qty'];
15     $graphData['f_rev'][$row['m']-1] += fcstData[$row['f_key']]['rev'];
16     $graphData['qty'][$row['m']-1] += $row['qty'];
17     $graphData['rev'][$row['m']-1] += $row['rev'];
18   }
19   if(!array_key_exists($row['short_desc'], $graphData)) {
20     $graphData[$row['short_desc']]['qty'] = array();
21     $graphData[$row['short_desc']]['rev'] = array();
22   }
23   $graphData[$row['short_desc']]['qty'][] = $row['qty'];
24   $graphData[$row['short_desc']]['rev'][] = $row['rev'];
25 }
```

Листинг 3: Формирование единого массива $graphData (abc_reg_sales_graph.php**)**

Теперь давайте посмотрим на некоторые данные, отображаемые на нашем первом графике. Этот график будет сравнивать количество реализованных единиц товара со спрогнозированным количеством. В листинге 4 приведен код, формирующий этот график.

```
$graph = new graph(WIDTH,
HEIGHT);
$graph->SetScale('textlin');
$b1 = new
BarPlot($graphData['qty']);
$l1 = new
LinePlot($graphData['f_qty']);
$graph->Add($b1);
$graph->Add($l1);
$graph->Stroke();
```

Листинг 4: Ваш первый график

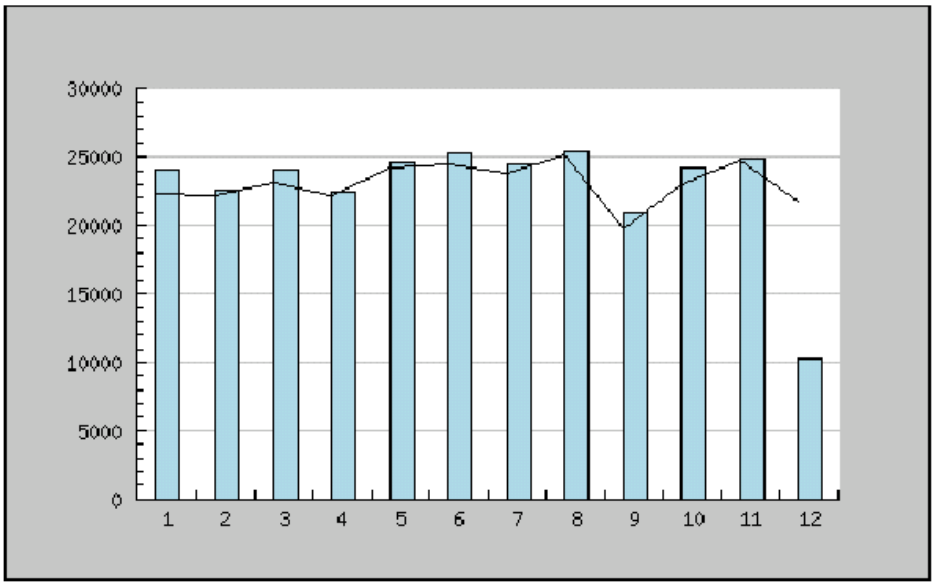

Рисунок 1. Ваш первый график

Этот код иллюстрирует сущность JpGraph API. Улучшение внешнего вида ваших графиков приведет к увеличению объема исходного кода, но код, приведенный в данном примере, является минимальным.

Эти несколько строк кода реализуют 4-х шаговый процесс, описанный выше. На шаге 2 Вы создаете и конфигурируете объект Graph. На шаге 3 создаются объекты BarPlot и LinePlot. Вы завершаете процесс на шаге 4, размещая графики на объекте-изображении и вызывая метод **Graph::Stroke()** для вывода графического объекта. Вызов метода **Graph::Stroke()** предписывает JpGraph напрямую вернуть браузеру изображение. Результат показан на рис. 1.

Непрерывность линии графика (спрогнозированных продаж – *прим.перев.*) на самом деле не соответствует действительности, так как данные в реальности не изменяются плавно от месяца к месяцу (прогнозные данные привязаны к месяцам). Данные прогноза будут лучше представлены «ступенчатой» линией графика.

Давайте внесем эти изменения и, заодно, покажем данные о доходах вместо данных об объемах продаж. Вы можете сделать это, изменив два графика так, как показано в листинге 5. Результат показан на рисунке 2.

```
$b1 = new
BarPlot($graphData['rev']);
$l1 = new
LinePlot($graphData['f_rev'])
$l1->SetStepStyle();
$l1->SetColor('darkgreen');
$l1->SetWeight(3);
```

Листинг 5: Графики для создания изображения, показанного на рис. 2

Во-вторых, было бы неплохо, если бы изображение выводилось таким образом, чтобы линия графика выходила за границу диаграммы, соответствующей декабрю месяцу.

В идеале вам нужна сгруппированная столбцовая диаграмма. Однако, JpGraph не позволяет группировать графики, построенные в разных масштабах. Чтобы добиться этого, мы пойдем на небольшой обман.

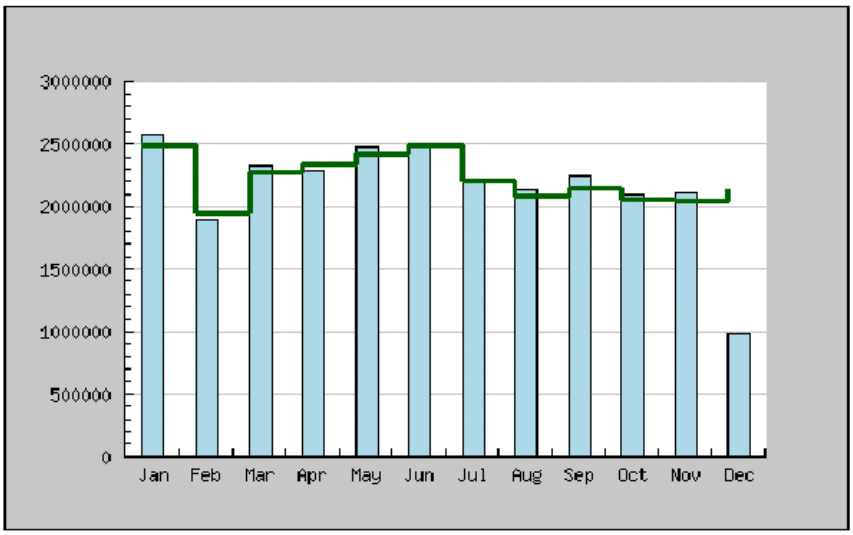

Рисунок 2. Использование «ступенчатой» линии графика.

Для представления объемов продаж и доходов на одном графике вам необходимо воспользоваться возможностью JpGraph создавать на графике вторую ось ординат.

На следующем шаге мы совместим в одном изображении график объемов продаж и график доходов. Здесь есть два принципиальных момента. Во-первых, взглянув на две диаграммы, вы можете увидеть, что графики продаж и доходов представлены в разных масштабах.

Создадим две сгруппированных столбцовых диаграммы (каждую в своем масштабе), в каждой из которых будет присутствовать диаграмма, представленная числовым рядом, состоящим только из нулей. Необходимый нам эффект достигается тем, что в сгруппированной диаграмме для одного масштаба диаграмма, состоящая из нулей, располагается справа, сдвигая основную диаграмму влево, а в сгруппированной диаграмме

для другого масштаба – слева, сдвигая основную диаграмму вправо.

Для форматирования меток на второй оси ординат вы можете создать функцию обратного вызова.

```
for
($i=0,$j=count($graphData['labelX'])
; $i<$j; $i++) {
  $graphData['zero'][$i] = 0;
}
//extend the forecast revenue line
by repeating the last value
$graphData['f_rev'][$j] =
$graphData['f_rev'][$j-1];
```

Листинг 6: Создание графика, состоящего из нулей (abc_reg_sales_graph.php)

JpGraph будет вызывать эту функцию для каждой метки, наносимой на координатную ось, и будет использовать значение, возвращенное функцией вместо номера строки. Это позволит нам выводить числа в формате денежных сумм. Код показан в листинге 7.

```
$graph->y2axis->SetLabelFormatCallback
('y_fmt_dol_thou');
function y_fmt_dol_thou($val)
{
  return '$'.number_format($val/1000);
}
```

Листинг 7: Функция обратного вызова для форматирования меток (format_callback.php)

Для построения графика (см. рис. 3) измените ваш код, включив в него код из листинга 8.

```
1  $graph->SetY2Scale('lin');
2  $graph->SetY2OrderBack(false);
3
4  //generate the individual plots
5  $b1 = new BarPlot($graphData['qty']);
6  $b2 = new BarPlot($graphData['rev']);
7  $b2->SetFillColor('lightgreen');
8  $b1z = new BarPlot($graphData['zero']);
9  $b2z = new BarPlot($graphData['zero']);
10 $l1 = new LinePlot($graphData['f_rev']);
11 $l1->SetStepStyle();
12 $l1->SetColor('darkgreen');
13 $l1->SetWeight(3);
14
15 //create the grouped plots
16 $gb1 = new GroupBarPlot(array($b1, $b1z));
17 $gb2 = new GroupBarPlot(array($b2z, $b2));
18
19 //add the plots to the graph object
20 $graph->Add($gb1);
21 $graph->AddY2($gb2);
22 $graph->AddY2($l1);
```

Листинг 8: Код для построения графика на рис. 3

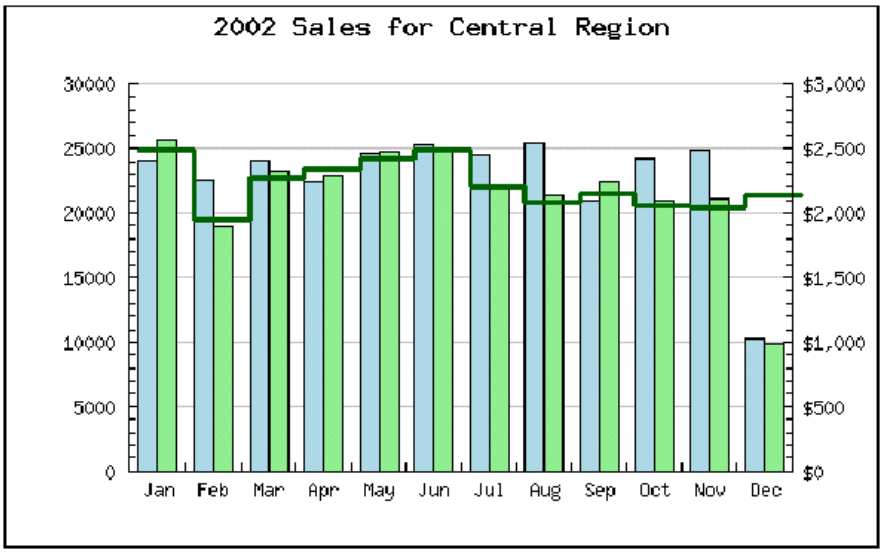

Рис. 3. Сгруппированная столбцовая диаграмма, построенная в разных масштабах.

На данный момент вы имеете график достаточно представительного вида, но к нему можно еще добавить некоторую полезную информацию. Пользователи хотели бы видеть вклад, вносимый каждым товаром в объемы продаж и соответствующий доход. Это может быть достигнуто созданием «слоеной» столбцовой диаграммы, добавленной к каждой сгруппированной столбцовой диаграмме, как показано в листинге 9. Результат показан на рис. 4.

```
1  $colors = array('pink', 'orange', 'yellow', 'lightgreen', 'lightblue');
2
3  $abqAdd = array();
4  $abrAdd = array();
5  for($i=0,$j=count($items); $i<$j; $i++) {
6    $key = $items[$i]['short_desc'];
7    $b1 = new BarPlot($graphData[$key]['qty']);
8    $b1->SetFillColor($colors[$i]);
9    $b1->SetLegend($items[$i]['item_desc']);
10   $abqAdd[] = $b1;
11
12   $b2 = new BarPlot($graphData[$key]['rev']);
13   $b2->SetFillColor($colors[$i]);
14   $abrAdd[] = $b2;
15 }
16 $ab1 = new AccBarPlot($abqAdd);
17 $ab2 = new AccBarPlot($abrAdd);
18 $b1z = new BarPlot($graphData['zero']);
19 $b2z = new BarPlot($graphData['zero']);
20
21 $gb1 = new GroupBarPlot(array($ab1, $b1z));
22 $gb2 = new GroupBarPlot(array($b2z, $ab2));
23
24 $graph->Add($gb1);
25 $graph->AddY2($gb2);
```

Листинг 9: Код для создания «слоеной» столбцовой диаграммы, показанной на рис. 4 (abc_reg_sales_graph.php**)**

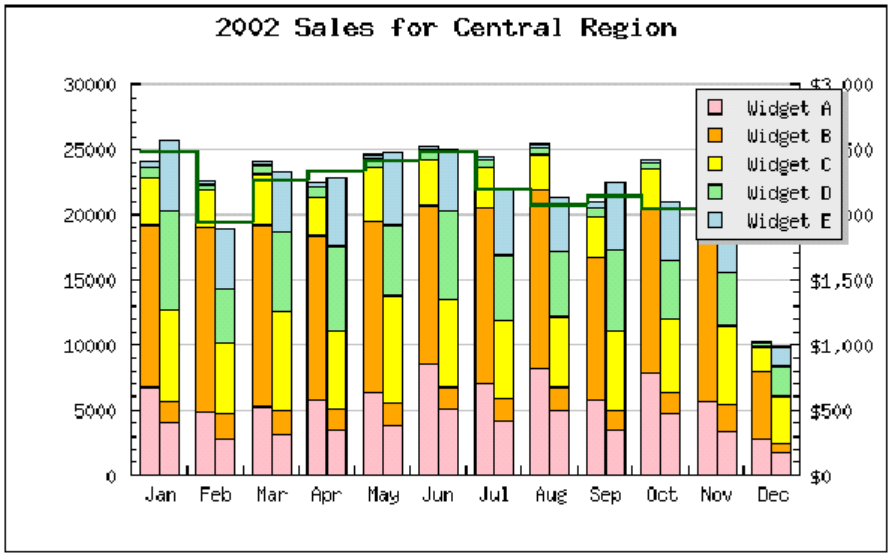

Рис. 4. Использование «слоеной» столбцовой диаграммы.

Мы почти у цели. Необходимо лишь немного отформатировать область, в которой отрисовываются графики, надписи возле координатных осей, а также название региона, для которого строится график. Кроме того, мы должны убедиться в том, что график используется только для внутренних целей фирмы. Один из способов указать на то, что объект является частной собственностью – добавить фоновое изображение. В данном случае, мы используем строку «ABC Co. Proprietary» («Собственность компании ABC» - *Прим. перев.*), написанную по диагонали и многократно повторенную.

Заметьте, что цвет надписи значительно темнее, чем тот, который вы хотели бы видеть на «водяных знаках». Для достижения требуемого эффекта можно использовать метод **Graph::AdjBackgroundImage(),** позволяющий отрегулировать яркость, контрастность и насыщенность изображения перед использованием его в графике.

Это избавит вас от усилий по редактированию изображения во внешнем графическом редакторе.

Посмотрите файл **img/abc-background.png** в директории с исходными кодами данной статьи в качестве примера того, как может выглядеть фоновый рисунок. Вы можете использовать этот рисунок в качестве фонового изображения, используя код из листинга 10.

Примечание: Библиотека GD2, встроенная в PHP 4.3.0, конфликтует с методом **Graph::AdjBackgroundImage().** Если вы используете эту версию PHP, вам придется отказаться от использования этого метода и обрабатывать изображение в графическом редакторе.

```
if (USING_TRUECOLOR) {
  $graph->SetBackgroundImage('img/
background_prefade.png',
BGIMG_FILLFRAME);
} else {
  //AdjBackgroundImage only works
GD, not GD2 true color
  $graph->SetBackgroundImage('img/
background.png', BGIMG_FILLFRAME);
  $graph->AdjBackgroundImage(0.9,
}
```

Листинг 10: Код для использования и настройки фонового изображения (abc_reg_sales_graph.php)

Вы можете использовать код, приведенный в листинге 11, чтобы добавить заключительные штрихи, такие как название графика и координатных осей, и настроить координаты надписей. Результат работы кода показан на рис. 5.

```
1  $graph->title->Set(date('Y')." Sales for{$regions[$region_id]} Region");
2  $graph->title->SetFont(FF_ARIAL, FS_BOLD, 12);
3  $graph->SetMarginColor('white');
4  $graph->yaxis->title->Set('Left Bar Units Sold');
5  $graph->yaxis->title->SetFont(FF_ARIAL, FS_BOLD, 10);
6  $graph->yaxis->SetLabelFormatCallback('y_fmt');
7  $graph->yaxis->SetTitleMargin(48);
8  $graph->y2axis->title->Set('Right Bar Revenue ( $ 000 )');
9  $graph->y2axis->title->SetFont(FF_ARIAL, FS_BOLD, 10);
10 $graph->y2axis->SetTitleMargin(45);
11 $graph->y2axis->SetLabelFormatCallback('y_fmt_dol_thou');
12 $graph->xaxis->SetTickLabels($graphData['labelX']);
13
14 $graph->legend->Pos(0.5, 0.95, 'center', 'center');
15 $graph->legend->SetLayout(LEGEND_HOR);
16 $graph->legend->SetFillColor('white');
17 $graph->legend->SetShadow(false);
18 $graph->legend->SetLineWeight(0);
```

Листинг 11: Код для завершения построения графика на рис. 5 (abc_reg_sales_graph.php**)**

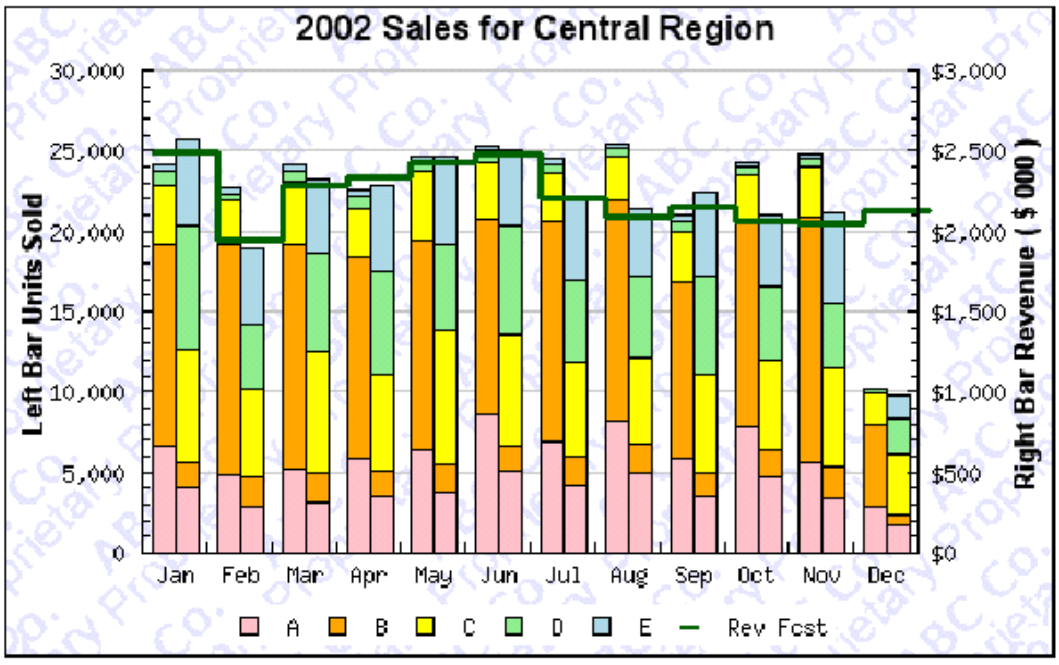

Рис. 5. Законченный вид графика

Из соображений повышения производительности вы решили реализовать механизм кэширования изображений JpGraph для этого графика. Механизм кэширования основан на сохранении копии изображения как файла на сервере. Если копия, сохраненная к кэше, еще действительна, JpGraph вернет ее в ответ на запрос клиента вместо того, чтобы генерировать изображение «на лету». Заметьте, что вы должны иметь на сервере соответствующую привилегию для записи в директорию кэша.

При создании экземпляра класса Graph вам необходимо передать конструктору, кроме ширины и высоты изображения, также имя файла кэшируемого изображения и таймаут в минутах (время, в течение которого действительно изображение, сохраненное в кэше) и, наконец, параметр, сообщающий JpGraph о том, что изображение все равно нужно выводить. Это значит, что вы будете продолжать использовать ссылку на PHP-скрипт как значение атрибута **src** тэга **img.** Код, необходимый для реализации механизма кэширования, приведен в листинге 12.

```
define('GRAPH_NAME',
'abc_reg_sales');
$graphName =
GRAPH_NAME.$region_id.'.png';
$graphTimeout = 60*24;
$graph = new graph(WIDTH, HEIGHT,
$graphName, $graphTimeout, true);
```

Листинг 12: Код, реализующий кэширование изображения. (abc_reg_sales_graph.php**)**

Теперь, если в течение 24 часов будет запрошен тот же график для того же региона, клиенту будет возвращена версия изображения, находящаяся в кэше, и выполнение скрипта прервется после строки **‘new Graph()’.**

Это означает, что для получения максимальной выгоды от кэширования вы должны создавать экземпляр класса Graph раньше, чем будут выполняться запросы к БД.

Сравнение объемов продаж через разные каналы по регионам

Второй график, который вас попросили сделать, показывает объемы продаж для каждого региона по каналам. Кроме того, нужно обеспечить простой способ навигации между первым и вторым графиками. Этот тип отчета предполагает просмотр информации о долях, поэтому наиболее эффективной будет круговая диаграмма. Просмотрите, пожалуйста, еще раз строки 22-82 файла **abc_map_graph.php** в директории с исходными кодами этой статьи, чтобы понять, как выполняются запросы к БД и формирование массивов для последующего построения графиков.

В листинге 13 приведен код, необходимый для создания графика, показанного на рис. 6.

```
$sliceColors = array('lightgreen', '
'lightblue');
$graph = new PieGraph(WIDTH, HEIGHT)
$graph->title-
>Set($regions[$region]['region'].'
Region');
$graph->subtitle->Set('Sales by Char
since '.GRAPH_START);
$p1 = new
PiePlot($graphData[$pickRegion]['rev
$p1-
>SetLegends($graphData[$pickRegion][
]);
$p1->SetSliceColors($sliceColors);
$graph->Add($p1);
$graph->Stroke();
```

Листинг 13: Код для создания к диаграммы на рис. 6

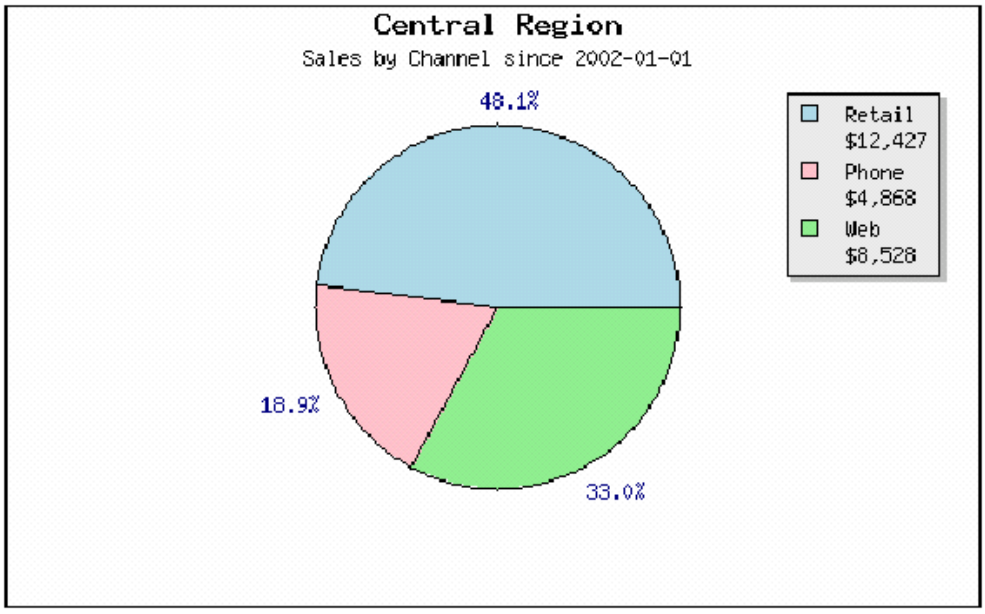

Рис. 6. Простая круговая диаграмма

Вы можете использовать фоновое изображение так, как это показано на примере первого графика, а также разместить дополнительную информацию на вашем графике. Разместим на графике карту США, разделенную по регионам продаж компании ABC. Если вы используете эту карту как фоновое изображение для графика, вы можете расположить круговые диаграммы на фоне соответствующего региона. Посмотрите файл **img/abc-regions.png** как пример использования фонового изображения в этом графике.

Для использования этого примера вам необходимо добавить несколько строк в цикл формирования массива **$graphData,** чтобы сделать возможным динамическое расположение круговых диаграмм для каждого региона:

```
$graphData['r'.$rIndex]['map_x'] =
$regionData[$i]['map_x'];
$graphData['r'.$rIndex]['map_y'] =
$regionData[$i]['map_y'];
```

Теперь взгляните на код листинга 14, создающий график, изображенный на рис. 7.

```
1  $graph = new PieGraph(WIDTH, HEIGHT);
2  $graph->SetBackgroundImage('img/abc-regions.png', BGIMG_FILLFRAME);
3
4  for ($i=0; $i<$rIndex+1; $i++) {
5    $pickRegion = 'r'.$i;
6
7    $p1 = new PiePlot($graphData[$pickRegion]['rev']);
8    $p1->SetCenter($graphData[$pickRegion]['map_x'],
9    $graphData[$pickRegion]['map_y']);
10   $p1->SetSize(PIE_SIZE);
11   $p1->SetLabels($graphData[$pickRegion]['revFmt']);
12   $p1->SetSliceColors($sliceColors);
13   if (!$i) {
14     $p1->SetLegends($graphData['label']);
15   }
16
17   $graph->Add($p1);
18 }
19
20 $graph->legend->Pos(0.9, 0.85, 'center', 'center');
21 $graph->Stroke();
```

Листинг 14: Код для создания графика на рис. 7 (abc_map_graph.php**)**

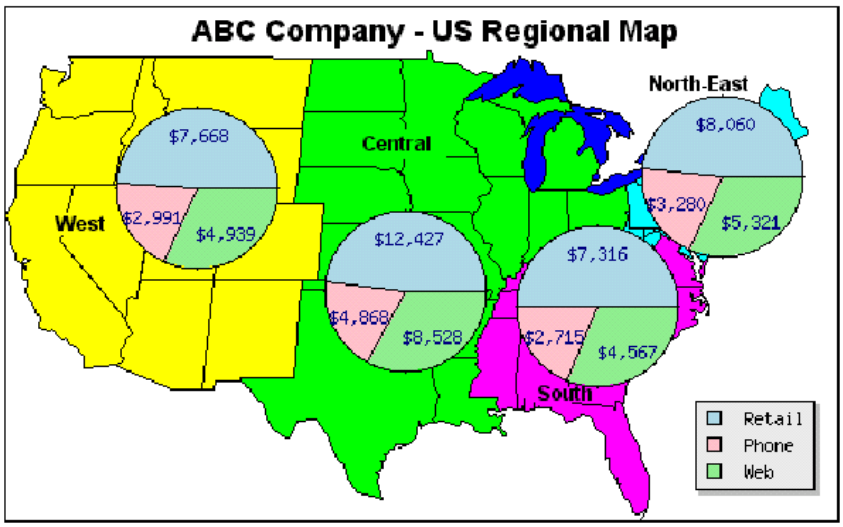

Рис. 7. Использование фонового изображения для показа регионов

В заключение заказчик хочет иметь возможность быстрой навигации между имеющимися круговыми диаграммами и соответствующими графиками продаж, построенными ранее. Вы можете реализовать эту возможность, используя карты-изображения на стороне клиента (Client Side Image Maps – CSIM – *Прим. перев.*). CSIM – HTML-технология, позволяющая определять области на изображении и связывать их с гиперссылками. Для реализации CSIM в этом графике вы должны определить гиперссылки и альтернативный текст для изображений (подсказки для пользователя). Для начала мы определим константу, используемую в большинстве ссылок:

```
define('DRILL_GRAPH',
'abc_reg_sales_graph.php?region=');
```

Теперь в массиве **$graphData** нужно опр ссылки и альтернативный текс изображений:

```
$graphData['r'.$rIndex]['targets']
[] =
DRILL_GRAPH.$regionData[$i]['regio
n_id'];
$graphData['r'.$rIndex]['alts'][]
=
  "Click for more information
regarding
{$regions[$rIndex]['region']}
sales.";
```

Так как в дополнение к двоичным данным необходимо выводить HTML (CSIM), мы не можем вернуть данные клиенту уже привычным нам способом. CSIM и изображение тесно зависят друг от друга. Чтобы сделать это, нам необходимо прибегнуть к кэшированию изображения. Это позволит нам вывести карту-изображение вместе с тэгом **img** и воспользоваться кэшированным изображением. Вместо того, чтобы воспользоваться механизмом кэширования JpGraph, описанным выше, мы применим другую технику и сохраним изображение в директории, из которой клиент сможет запросить его напрямую. Для этого надо создать директорию с именем **'img'** в той же директории, где находится скрипт. Вы также должны иметь на сервере права на запись в эту директорию.

При создании графика обрабатываем его так же, как если бы мы возвращали клиенту двоичный поток – изображение. В цикле формирования круговых диаграмм добавляем информацию о ссылках и альтернативном тексте (см. листинг 14 – *Прим. перев.*):

```
$p1->SetCSIMTargets(
$graphData[$pickRegion]['targets'],
      $graphData[$pickRegion]['alts']
      );
```

Далее выводим график, используя код из листинга 15.

```
define('IMG_DIR', 'img/');
$graphName =
IMG_DIR.'abc_channel_graph.png';
$graph = new PieGraph(WIDTH, HEIGHT);
//the rest of the graph code...
$graph->Stroke($graphName);
$mapName = 'ABC_Region_Drill';
$imgMap = $graph-
>GetHTMLImageMap($mapName);
print <<<EOS
$imgMap
<img src="$graphName" alt="ABC Sales
by Channel"
ismap usemap="#$mapName" border="0">
EOS;
```

Листинг 15: Код для формирования CSIM (abc_map_graph.php**)**

Данный код указывает JpGraph вывести изображение в файл **img/abc_channel_graph.png.** Далее помещаем в переменную **$imgMap** сгенерированную карту-изображение. Тэг **img** определен таким образом, что позволяет использовать сгенерированную карту-изображение.

Ключевые понятия

В данном учебном примере были рассмотрены следующие ключевые понятия, касающиеся построения графиков средствами JpGraph:

- использование в JpGraph линейных графиков, столбцовых и круговых диаграмм;
- использование «слоеных» и сгруппированных столбцовых диаграмм;
- использование второй оси ординат с другим масштабом;
- использование функции обратного вызова для форматирования меток на осях координат;
- генерация сообщений об ошибках в графическом виде;
- создание «водяных знаков» с использованием фоновых изображений;
- использование механизмов кэширования изображений JpGraph для увеличения производительности, а также облегчения использования CSIM;
- использование фонового изображения как части информационного контента диаграммы (расположение круговой диаграммы на фоне карты регионов продаж);
- использование карт-изображений на стороне клиента для быстрой навигации между графиками

Заключение

JpGraph предоставляет простое API, позволяющее быстро создавать профессиональную графику. Соединение возможностей JpGraph и средств PHP для доступа к БД дает вам в руки мощный инструмент для создания динамических графиков в web'е. В данной статье представлены некоторые из дополнительных возможностей JpGraph такие как: кэширование, фоновые изображения и карты-изображения на стороне клиента. Надеюсь, вы достаточно хорошо познакомились с данной технологией и сможете свободно использовать PHP и JpGraph в своих будущих data mining проектах.

Они делают это так.

Интервью с президентом New York PHP Хансом Заунером специально для нашего журнала. Интервьюирует **nw**

Люди

Нью-Йорк PHP (NYPHP) – это группа профессионалов, которая занимается PHP разработками и AMP технологией (**A**pache+**M**ySQL+**P**HP). Имея в своем составе свыше тысячи разработчиков, группа является самой большой на территории Соединенных Штатов. Она активно поддерживает PHP/AMP сообщество, участвуя в Open Source проектах, организовывая сетевые мероприятия, встречи и листы рассылок. NYPHP тесно сотрудничает с деловыми кругами Нью-Йорка и всем США, продвигая PHP и укрепляя влияние AMP технологий.

Современный мегаполис является хорошим рынком и энергичные, квалифицированные разработчики могут занять здесь свою нишу. NYPHP объединяет знания и навыки сообщества PHP и связанных с ним технологий. Это некоммерческая организация, основной целью которой является обмен знаниями и обучение друг у друга. В будущем организация планирует разрабатывать Open Source проекты и базы знаний, подготавливать статьи и обучающие материалы.

Помимо AMP технологий, некоторые сообщества ориентируются на LAMP. Это сокращение от **L**inux+**A**pache+**M**ySQL+**P**HP/Perl/Python.
Но NYPHP не сосредотачивает внимание на конкретной операционной системе и стремится к ориентации, прежде всего на язык PHP.

Президент NYPHP Ханс Заунер (Hans Zauner) дал небольшое интервью для нашего журнала. Вопросы были разбиты на три темы.

О NYPHP

nw: Как долго существует ваше сообщество?
HansZ: Почти два года.
nw: И сколько разработчиков в него входят? Хотя бы примерно.
HansZ: 1000+
nw: А есть ли у вашего сообщества какой либо официальный (или частный) журнал или сайт, помимо nyphp.com?
HansZ: На самом деле наш главный сайт – nyphp.org, так же у нас размещено некоторое количество статей на `phundamentals.nyphp.org`. Еще мы продолжаем работать над CMS сообщества на `community.nyphp.org`.

> ***Ханс Заунер:*** *PHP еще только входит в большие корпорации Нью Йорка. Этот процесс начался недавно, и по мере его продвижения, зарплаты разработчиков будут расти.*

О PHP разработчиках в Нью-Йорке

nw: Как вы оцениваете уровень жизни PHP разработчика в Нью-Йорке в среднем (средний заработок, социальный статус и прочее)?
HansZ: Трудно сказать, но я считаю, что у некоторых этот уровень высок. PHP еще только входит в большие корпорации Нью-Йорка. Этот процесс начался недавно, и по мере его продвижения, зарплаты разработчиков будут расти. На данный момент PHP в основном используется в проектах малого и среднего бизнеса.
nw: Скажите, как PHP разработчики в Нью-Йорке проводят свое свободное время?
HansZ: Что такое свободное время? ☺.
Если серьезно, то об этом сложно говорить. У нас много разных интересов.
nw: Какие сорта и марки пива пьют американские разработчики? ☺
HansZ: Еще один сложный вопрос. Я думаю, что нам нравится практически все. Лично я уважаю Becks, другие склонны к Guiness и Weisse.
nw: А как много разработчиков работает не в офисе, а дома в качестве фрилансеров (в процентах)?

HansZ: Я бы сказал большинство – свыше пятидесяти процентов. Много разработчиков работает полный день в офисе и имеет дополнительный заработок на стороне.

О PHP в Нью Йорке

nw: Какое программное обеспечение вы используете для разработки PHP скриптов (я имею ввиду IDE, web сервера, СУБД, инструменты тестирования и прочее)?

HansZ: Многие разработчики используют Zend Studio, но, конечно, не забывают и любимые текстовые редакторы (UltraEdit, vi и другие). Также популярны Dreamviewer, Homesite и GoLive. Основная разработка идет на базе Apache и MySQL, правда, IIS, Oracle и MSSQL тоже в ходу.

nw: И, наконец, последний вопрос. Недавно наше сообщество организовало выпуск электронного журнала на русском языке. Что вы могли бы пожелать нашему журналу и сообществу PHP разработчиков в России (СНГ)?

HansZ: Это впечатляющее зрелище - видеть, как PHP распространяется по всему миру. Я хотел бы пожелать Русскому сообществу всего самого наилучшего, и, надеюсь, мы сможем продолжить сотрудничество в будущем.

nw: Спасибо за ответы!

HansZ: Спасибо вам. Если вы хотите, чтобы я ответил на некоторые вопросы более подробно, или у вас будут другие вопросы, не стесняйтесь их задавать.

Всего самого наилучшего, Ханс Заунер, президент New York PHP. http://nyphp.com

Лердорф: Будущий релиз PHP «эволюционен, а не революционен»

Надя Камерон
Оригинал: http://www.linuxworld.com.au/index.php?id=1241831333&fp=2&fpid=1
Перевод: **nw**

«Ожидаемый релиз PHP 5 будет больше «эволюционным», чем «революционным» инструментом, обеспечивающим беспрепятственное взаимодействие с другими языками программирования», - сказал его создатель.

Обсуждая различия между текущей четвертой версией PHP и ее наследником в интервью с LinuxWorld, PHP гуру Расмус Лердорф сказал, что в пятой версии улучшена поддержка новых web протоколов и совместимость с другими языками программирования, что открывает большое количество новых возможностей. К примеру, в версии 5 будет улучшена поддержка Java и XML.

Другая ключевая деталь грядущего релиза версии 5 – обновление оригинальной базы кода PHP, большая часть которого была написана в 1994 году, добавил Расмус.

PHP (гипертекстовый процессор) - это инструмент объектно-ориентированного программирования с открытым кодом (open source), предназначенный для скриптования содержимого web приложений и их построения. Лердорф сказал, что первоначальной идеей создания PHP было «создать простой скрипт».

«PHP был основан на средстве, работающем лично для меня и никого больше. Я никогда не намеревался изменить мир программирования. Это простой язык с простым синтаксисом».

Тем не менее, PHP стал широко распространенным инструментом по всему интернету. Согласно исследованию 2002 года, проведенному NetCraft и охватившему свыше шести миллионов web доменов, каждый пятый из них использует PHP. Исследование показало, что здесь также существует тенденция ежемесячного роста в среднем на 15 процентов.

На сегодняшний день написанием кода и патчей для PHP занимается 800-900 человек. Комментируя успех PHP, Лердорф сказал, что он ожидал разработки какой-либо другой технологии, «которая делала бы то же самое, что и PHP, только лучше».

«Я всегда ожидал появления новой технологии, которая могла бы делать то же самое, что и PHP, но пока такой технологии нет. Прошло уже 10 лет».

«Программное обеспечение с открытым исходным кодом (open source) имеет одно большое преимущество перед коммерческим софтом – возможность «обрезать и подправить» код, чтобы настроить его для своих нужд. Это остается приоритетом в разработке новых версий PHP», - сказал Лердорф.

Но, не смотря на все будущие улучшения, необходимые для поддержки будущих web протоколов и языков программирования, PHP останется web-ориентированным скриптовым языком.

«Это клей, соединяющий web», - сказал Расмус. «Пока будет web, нам будет нужен клей, соединяющий все это вместе». Выход пятой версии PHP ожидается в этом году. Финальную бета-версию (бета 3) можно скачать для тестирования с сайта группы разработки PHP http://www.php.net.

Расмус Лердорф будет выступать на ежегодной конференции linux.conf.au, которая состоится в Аделаиде (Австралия) 14 – 17 января 2004 года (на момент публикации конференция состоялась – *прим. переводчика*). Там он представит некоторую документацию и руководства.

Стандарты кодирования PEAR

Материал с сайта http://www.pear.php.net

Замечание: Стандарты кодирования PEAR используются в коде, который в итоге станет частью PEAR, который в свою очередь поставляется с дистрибутивом PHP или доступен для скачивания через утилиты инсталляции PEAR.

Отступы

Используйте для отступа 4 пробела, а не табуляцию. Если вы используете Emacs для редактирования кода, вы должны установить indent-tabs-mode в ноль. В листинге 1 приведен пример настройки Emacs для этой цели:

```
(defun php-mode-hook ()
  (setq tab-width 4
        c-basic-offset 4
        c-hanging-comment-ender-p nil
        indent-tabs-mode
        (not
         (and (string-match "/\\(PEAR\\|pear\\)/" (buffer-file-name))
              (string-match "\.php$" (buffer-file-name))))))
```

Листинг 1

А это пример конфигурации для редактора VIM:

```
set expandtab
  set shiftwidth=4
  set tabstop=4
```

Управляющие структуры

Управляющие структуры включают в себя операторы if, for, while, switch, и др. Ниже приведен пример оформления оператора if, который в этом отношении является самым сложным:

```
if ((condition1) || (condition2)) {
    action1;
} elseif ((condition3) &&
(condition4)) {
    action2;
} else {
    defaultaction;
}
```

В управляющих структурах между ключевым словом и открывающей круглой скобкой должен находиться пробел, чтобы отличать их от вызова функций.

Настойчиво рекомендуется использовать фигурные скобки, даже в том случае, когда их использование не является необходимостью. Использование фигурных скобок увеличивает читабельность кода и уменьшает вероятность логических ошибок при изменении кода.

Синтаксис оператора switch:

```
switch (condition) {
case 1:
    action1;
    break;

case 2:
    action2;
    break;

default:
    defaultaction;
    break;

}
```

Вызовы функций

Вызовы функций должны быть написаны без отступов между именем функции, открывающей скобкой и первым параметром. Отступы в виде пробела должны присутствовать после каждой запятой в перечислении

параметров. Пробелов также не должно быть между последним параметром, закрывающей скобкой и точкой с запятой. Пример:

```
$var = foo($bar, $baz, $quux);
```

Как можно заметить, в примере используются пробелы с двух сторон от знака "=". Если подобные присвоения результатов функций переменным объединяются в блоки, то для повышения читабельности рекомендуется следующий синтаксис:

```
$short         = foo($bar);
$long_variable = foo($baz);
```

Определения функций

Определения функций следуют такому соглашению:

```
function fooFunction($arg1, $arg2 =
'')
{
if (condition) {
statement;
}
return $val;
}
```

Аргументы функций со значениями по умолчанию должны находиться в конце списка аргументов. Функции всегда должны возвращать значение, если это возможно в принципе. Чуть более подробный пример (листинг 2):

Комментарии

Комментарии внутри кода классов должны соответствовать синтаксису комментариев PHPDoc, который напоминает Javadoc. За дополнительной информацией о PHPDoc обращайтесь сюда: http://www.phpdoc.de/

Дополнительные комментарии, кроме тех, что предусмотрены PHPDoc, только приветствуются. Основное правило в данном случае: каждая часть кода повышенной сложности должна быть прокомментирована до того, как вы забыли, как она работает.

Подходят комментарии в стилях C (/* */) и C++ (//). Использование комментариев в стиле Perl/shell (#) не рекомендуется.

Подключение кода (including)

В тех местах, где вы используете подключение файлов других классов вне зависимости от условий, используйте конструкцию **require_once()**. Если же подключение файлов зависит от каких-либо условий, то следует использовать **include_once()**. В этом случае вы всегда будете уверены в том, что файлы подключаются только единожды.

```
function connect(&$dsn, $persistent = false)
{
    if (is_array($dsn)) {
        $dsninfo = &$dsn;
    } else {
        $dsninfo = DB::parseDSN($dsn);
    }

    if (!$dsninfo || !$dsninfo['phptype']) {
        return $this->raiseError();
    }

    return true;
}
```

Листинг 2

Замечание: include_once() и **require_once()** являются конструкциями, а не функциями. Вам *не обязательно* использовать скобки вокруг имени файла, который подключается.

Тэги PHP-кода

Всегда используйте `<?php ?>` вместо `<? ?>` для выделения PHP-кода. Это необходимо для обеспечения работы PEAR на разных операционных системах и с различными настройками.

Блок комментариев в заголовке

Все базовые файлы исходного кода в PEAR должны содержать следующий ниже блок комментариев в заголовке (листинг 3):

```
/* vim: set expandtab tabstop=4 softtabstop=4 shiftwidth=4: */
// +----------------------------------------------------------------------+
// | PHP version 4                                                        |
// +----------------------------------------------------------------------+
// | Copyright (c) 1997-2002 The PHP Group                                |
// +----------------------------------------------------------------------+
// | This source file is subject to version 2.0 of the PHP license,       |
// | that is bundled with this package in the file LICENSE, and is        |
// | available at through the world-wide-web at                           |
// | http://www.php.net/license/2_02.txt.                                 |
// | If you did not receive a copy of the PHP license and are unable to   |
// | obtain it through the world-wide-web, please send a note to          |
// | license@php.net so we can mail you a copy immediately.               |
// +----------------------------------------------------------------------+
// | Authors: Original Author <author@example.com>                        |
// |          Your Name <you@example.com>                                 |
// +----------------------------------------------------------------------+
//
// $Id$
```

Листинг 3

Нет четкого правила, которое определяет момент, когда новый разработчик должен быть добавлен в список авторов данного файла. В общем случае, его внос в изменения этого файла должен относиться к категории "значительных" (т.е. около 10%-20% процентов кода).

Исключения могут быть в случае переписывания функций или добавления новой логики.

Простая реорганизация кода и исправление ошибок не приводит к добавлению нового участника в список авторов.

Файлы, которые не входят в базовую часть PEAR, должны включать такой же блок комментариев в заголовке, включая авторские права, лицензию и список авторов. Комментарии должны быть отформатированы для того, чтобы сохранять свою целостность.

Использование CVS

Эта часть касается только пакетов, использующих CVS на cvs.php.net.

Включайте ключевое слово CVS - Id в каждый файл. Добавьте эту метку в каждый файл, если там ее еще нет, или исправьте уже существующую запись "Last Modified:" и т.п.

В оставшейся части этой главы предполагается, что вы имеете

представление о тэгах CVS и ветках (branches).

Тэги CVS предназначены для того, чтобы пометить файлы, которые принадлежат к конкретному релизу. Ниже приводится список необходимых и рекомендуемых тэгов:

RELEASE_*n_n*

(обязательный). Используется для пометки релиза. Если вы не используете его, то вы не сможете вернуться назад и затребовать пакет в том виде, в котором он находился во время прошлого релиза.

QA_*n_n*

(необязательный). Если вы чувствуете, что перед выпуском релиза необходимо выпустить предварительную версию (release candidat), то вы можете сделать ветку (branch) кода для того, чтобы изолировать релиз и вносить только критически важные изменения до релиза. При этом обычный процесс разработки может продолжаться в основном дереве кода.

MAINT_*n_n*

(необязательный). Если вам нужно сделать "микро-релиз" (например, версию 1.2.1 и т.п. после 1.2), вы также можете использовать ветку в том случае, если основной код меняется достаточно активно, и вы хотите вносить только небольшие изменения в микро-релизы.

Обязательным является только тэг RELEASE, остальные рекомендуются для вашего же удобства.

Пример того, как пометить тэгом релиза 1.2 пакет "Money_Fast":

```
$ cd pear/Money_Fast
$ cvs tag RELEASE_1_2
T Fast.php
T README
T package.xml
```

Сделав так, вы получаете возможность использовать веб-сайт PEAR для дальнейшего процесса выпуска релизов.

Пример создания ветки для QA:

```
$ cvs tag QA_2_0_BP
...
$ cvs rtag -b -r QA_2_0_BP QA_2_0
$ cvs update -r QA_2_0
$ cvs tag RELEASE_2_0RC1
...далее, создаем настоящий релиз
из то же ветки:
$ cvs tag RELEASE_2_0
```

Тэг "QA_2_0_BP" - это тэг ветки. Рекомендуется всегда выделять ветки этим тэгом. Служебные ветви (MAINT branches) могут быть отмечены как релиз и без использования этого тэга.

Примеры URLов

Используйте "example.com" для примеров URLов, как описано в RFC 2606.

Соглашения об именах

В общем случае имена классов, функций и переменных всегда должны быть "говорящими" для того, чтобы читатель мог сразу понять, для чего они используются.

Классы

Имена классов должны быть удобочитаемыми и понятными. Избегайте использования аббревиатур там, где это возможно. Имена классов должны начинаться с буквы в верхнем регистре. Иерархия классов PEAR также

отражается на именах классов, каждый уровень отделяется знаком подчеркивания. Примеры хороших имен классов:

Log Net_Finger HTML_Upload_Error

Функции и методы

Функции и методы должны использовать "венгерскую нотацию" (в другом варианте - "верблюжачью" =)). Функции также должны иметь префикс в виде имени пакета для того, чтобы избежать проблем с аналогичными функциями из других пакетов. Первая буква в имени функции должна быть в нижнем регистре, каждая первая буква "слова" в имени функции - в верхнем. Несколько примеров:

connect() getData() buildSomeWidget()

Приватные методы и свойства (те методы и атрибуты, которые используются только внутри самого класса; PHP пока(?) не поддерживает их настоящее выделение) должны быть префиксированы знаком подчеркивания. Например:

_sort() _initTree() $this->_status

Константы

Имена констант всегда должны быть в верхнем регистре с подчеркиваниями для разделения слов. В качестве префикса в именах констант должно использоваться имя пакета/класса, в котором они используются. Например, все константы, которые используются в пакете `DB::`, начинаются с "`DB_`".

Глобальные переменные
Если в вашем пакете необходимо объявить глобальные переменные, то их имя должно начинаться с подчеркивания, имени пакета и еще одного пакета. Например, пакет PEAR использует глобальную переменную, которая называется $_PEAR_destructor_object_list.

Встроенные переменные TRUE, FALSE, NULL

Встроенные переменные PHP **`TRUE`**, **`FALSE`** и **`NULL`** должны быть написаны в нижнем регистре.

Стандарт кодирования на PHP

Материал с сайта http://tony2001.phpclub.net
Перевод: **Антон Довгаль [tony2001]**
Оригинал статьи: http://alltasks.net/code/php_coding_standard.html
Автор оригинала: Фредерик Кристиансен (Fredrik Kristiansen)

Почему стандартизация так важна

Стандарты нужны уже потому, что одинаково достают всех и таким образом, эти все чувствуют себя командой. Все, предложенное ниже, использовалось многими компаниями во множестве проектов, и споры по этим правилам длились буквально недели. Предлагаемый стиль не является чьим-либо персональным стилем и, конечно, в стандартах допускаются частные дополнения.

Положительные моменты

Когда проект пытается принять тот или иной общепринятый стандарт, случаются следующие хорошие вещи:

- программисты могут прочитать код и легко разобраться, что в нём происходит;
- новые программисты быстрее вписываются в проект;
- новые люди в PHP избавлены от необходимости разрабатывать свой персональный стиль и стоять насмерть, защищая его;
- новые люди в PHP избавлены от "необходимости" допускать те же самые ошибки, которые всегда допускают новички;
- в устойчивых системах люди делают меньше ошибок;
- у программистов появляется общий враг :-)

Отрицательные моменты

Плохие вещи тоже случаются:

- как правило, стандарты - это абсолютный хлам, поскольку разрабатывались людьми, ничего не соображающими в PHP;
- как правило, стандарты - это абсолютный хлам, поскольку это не то, что я хочу;
- стандарты снижают креативность;
- для состоявшихся программистов необходимость в стандартах исчезает;
- стандарты насаждают слишком много структуры;
- всё равно люди не следуют стандартам.

Обсуждение

Опыт многих проектов приводит нас к следующему заключению: с введением стандартов проект продвигается быстрее. Но тогда получается, что стандарты - залог успеха? Конечно, нет. Но они способствуют успеху, а нам нужно использовать все возможности! Будем честны с собой: большинство аргументов против того или иного стандарта исходит от нашего самолюбия. В хорошем стандарте редко случается найти ограничение, которое отрицательно сказалось бы на качестве проекта, в большинстве своём всё это - лишь дело вкуса. Итак, проявите больше гибкости, контролируйте своё самолюбие и помните, что проект продвигается единой командой, а не отдельными программистами.

Принятие стандарта

Трактовки

Наличие слова "обязательно" означает, что все проекты, использующие этот документ, должны придерживаться этого правила.
Слова "нужно", "должен" и подобные им означают, что решение о применении, изменении правила или отказе от него находятся в вашей компетенции.
Слово "рекомендуется" схоже по смыслу со вторым пунктом в том, что правило применяется по возможности.

Принудительное принятие

Прежде всего, любые, хоть сколько-нибудь важные решения по

стандартизации желательно принимать коллективно. Может быть, для вашей конкретной ситуации такой стандарт не подходит: возможно, сам стандарт не учитывает какие-то важные моменты; возможно, те или иные проблемы упорно игнорируются кем-то главным :-) В любом случае, как только стандарт будет-таки утверждён, все поведут себя как взрослые люди и поймут, что в навязанных им правилах есть здравый смысл; что если эти правила подходят для многих программистов, то стоит их придерживаться, пусть и с некоторыми оговорками.
Если вариант коллективного принятия не проходит, можно объявить соблюдение стандартов необходимым условием успешного прохождения анализа исходников.
Если и это не проходит, то остаётся потворствовать всем предложениям и идеям противника.

Этапы принятия идеи

1. Это невозможно.
2. Может быть, это как-то и получится, но всё это слабовато и неинтересно.
3. Именно так надо делать, я вам говорю.
4. Да, сначала я подумал именно об этом.
5. Иначе и быть не может.

Если вы изначально воспринимаете что-либо предвзято, оставайтесь восприимчивым к альтернативам. Вполне возможно, что вы убедитесь, что предложенное вам действительно абсолютный хлам, но только таким путём вы можете найти другое решение. Так что позвольте себе пройти немного в этом направлении.

Выбирайте правильные имена

Имена - сердце программирования. В прошлом люди верили, что если узнать настоящее имя человека, можно получить власть над ним. Если вы подберёте правильные имена, вы наделите себя и других (всех, кто придёт после вас) властью над кодом. Просьба не смеяться, всё серьёзно.
Имя является результатом продолжительного осмысления мира, в котором "живёт" тот или иной объект. Только тот программист, который понимает систему в целом, в состоянии дать объектам имена, вписывающиеся в концепцию создаваемой системы. Если имя подобрано правильно, то всё стоит на своих местах, отношения между объектами ясны, значения легко угадываются, и все человеческие мотивировки и ожидания срабатывают, как хотелось бы.
И если вы вдруг обнаружили, что все объекты вашего кода называются *Фигня* и *Штуковина*, то такой код вам стоит серьёзно пересмотреть.

Имена классов

- Давайте классу имя тогда, когда вы знаете, что этот класс будет делать, как и для чего. Если вы этого не знаете, то вполне возможно, вы не продумали до конца концепцию модуля;
- Наличие имён, составленных более чем из трёх слов, может привести к тому, что система будет путать различные объекты программы. Пересмотрите код программы. Прогоните код через контроль циклически избыточного кода CRC и посмотрите, не берут ли ваши объекты на себя больше задач, чем вы планировали;

- Не поддавайтесь искушению присвоить производному классу имя, производное от имени родительского класса. Лучше будет, если класс будет жить своей жизнью, кто бы ни был его родительским классом;
- Иногда помогают суффиксы. Например, если в вашей системе используются различные агенты, то имя типа DownloadAgent несёт достаточную смысловую нагрузку;
- В качестве разделителей слов используйте заглавные буквы, строчные - для остальной части слов;
- Первая буква в имени - заглавная;
- Никаких underscore-ов ('_').

Обоснование
Из всех других вариантов многие выбрали этот как лучшее компромиссное решение.

Пример

class NameOneTwo
class Name

Имена методов

Как правило, функция или метод совершают какое-либо действие, поэтому желательно, чтобы из имени было понятно, какое именно действие будет совершаться: **CheckForErrors()** [ИщиОшибки()] вместо **ErrorCheck()** [ПоискОшибок()]; **DumpDataToFile()** [СваливайДанныеВФайл()] вместо **DataFile()**[ФайлДанных()]. Кроме того, так легче будет отличить метод от класса. Иногда помогают суффиксы:

- Max - чтобы показать максимальное значение;
- Cnt [count: количество, подсчёт] - чтобы показать текущее значение какого-либо счётчика;
- Key - чтобы показать ключевое значение. Например: RetryMax содержит максимальное количество возможных попыток, а RetryCnt - номер текущей попытки;

Префиксы тоже иногда нелишни:

- Is - для обозначения вопроса. Где ни встретится вам Is, вы всегда будете знать, что это вопрос;
- Get - получить значение;
- Set - установить значение.

Например: **IsHitRetryLimit()** [примерно: это ли последняя попытка?].

Никаких аббревиатур заглавными буквами

Если в имени переменной содержится аббревиатура, лучше вместо всех заглавных оставить только первую букву заглавной, а остальные написать строчными.

Неправильно: **GetHTMLStatistic()**.
Правильно: **GetHtmlStatistic()**.

Обоснование
При формировании имён, содержащих сокращения, люди используют свои интуитивные системы по-разному, поэтому лучше придерживаться единой стратегии формирования имён для всех случаев, и добиться тем самым предсказуемости именования. Возьмём, к примеру, NetworkABCKey. Заметьте, что C от ABC и K от Key воспринимаются больше как буквенное сочетание. В принципе, некоторые не возражают против аббревиатур целиком из заглавных, другие же их просто ненавидят; так что в разным проектах люди придерживаются разных стратегий.

Пример

```
function FluidOz()     //a не FluidOZ
function GetHtmlStatistic()                //a не
GetHTMLStatistic

class NameOneTwo
{
   function DoIt() {};

   function HandleError() {};
}
```

Имена аргументов в методах

- Первая буква - всегда строчная;
- Все остальные слова в имени начинаются с большой буквы, как при именовании классов.

Обоснование
Всегда можно легко определить, какие переменные поступили в метод в качестве аргумента.

Пример

```
class NameOneTwo
{
   function          StartYourEngines(&$someEngine,
&$anotherEngine) {
      $this->mSomeEngine = $someEngine;
      $this->mAnotherEngine = $anotherEngine;
   }

   var $mSomeEngine;
   var $mAnotherEngine;
}
```

Имена переменных

- Используйте только строчные буквы;
- В качестве разделителя слов используйте underscore ('_').

Обоснование

- При таком подходе область действия переменных более уловима;
- Все переменные кода выглядят по-разному и легко распознаются при чтении.

Пример

```
function HandleError($errorNumber)
{
   $error = new OsError;
   $time_of_error = $error->GetTimeOfError();
   $error_processor = $error->GetErrorProcessor();
}
```

Имена элементов в массивах

- Именование элементов массивов происходит по правилам именования переменных;
- В качестве разделителя слов используйте underscore ('_');
- И не используйте в качестве разделителя дефис ('-').

Обоснование

Если в качестве разделителя используется дефис, то при включении опции Magic Quotes вы получите сообщения об ошибках.

Пример

```
$myarr['foo_bar'] = 'Hello';
print "$myarr[foo_bar] world"; // выведет: Hello
world

$myarr['foo-bar'] = 'Hello';
print "$myarr[foo-bar] world"; // получим warning
```

Одиночные и двойные кавычки

Осуществляя доступ к элементам массива, можете использовать как одинарные, так и двойные кавычки. Не используйте кавычки при включённой опции Magic Quotes.

Обоснование
За исключением случаев использования Magic Quotes, некоторые конфигурации PHP выдают сообщение об ошибке, если при доступе к элементам массива кавычки опускаются.

Пример

```
$myarr['foo_bar'] = 'Hello';
$element_name = 'foo_bar';
print "$myarr[foo_bar] world"; // выведет: Hello world
print "$myarr[$element_name] world"; // выведет: Hello world
print "$myarr['$element_name'] world"; // parse error
print "$myarr["$element_name"] world"; // parse error
```

Глобальные переменные

- В именах глобальных переменных рекомендуется использовать префикс 'g'.

Обоснование
Такой префикс необходим для определения области действия переменных

Пример

```
global $gLog;
global &$grLog;
```

Имена функций

- Для функций PHP используйте правила C GNU: имена из строчных букв с underscore-ами ('_') в качестве разделителей.

Обоснование
Такой стиль позволяет делать различие между функциями и любыми именами, связанными с классами.

Пример

```
function some_bloody_function()
{
}
```

Правила расстановки фигурных скобок

Из трёх существующих стилей расстановки фигурных скобок допустимы два, причём первому рекомендуется отдавать предпочтение:

1. Открывающая скобка ставится под соответствующим оператором и на одном отступе с ним:
2. Unix-стиль расстановки фигурных скобок, когда открывающая скобка ставится на одной строке с соответствующим оператором, а закрывающая - на одном отступе с оператором:

```
if ($condition) {     while ($condition) {
    ...                   ...
}                     }
```

Обоснование
Ещё одна религиозная война, где мир установился с принятием компромиссного решения. Допускаются оба стиля, однако многие находят первый стиль более эргономичным и эстетичным. Почему - это целая тема для отдельного психологического

исследования.
Преимущество первого стиля заключается не только в психологии. Если вы используете текстовый редактор (например, vi), поддерживающий проверку на парность скобок, первый стиль будет более удобен. Почему? - спросите вы. Допустим, вы наткнулись на большой блок кода и желаете узнать, где же он заканчивается. Наводите курсор на открывающую скобку, жмёте нужную кнопку, и редактор находит парную скобку.

Пример

```
if ($very_long_condition && $second_very_long_condition) //два очень длинных условия
{
   ...
}
else if (...)
{
   ...
}
```

Итак, для перемещения от блока к блоку вам понадобится стрелка вниз и кнопка поиска парной скобки. И не надо ёрзать и гнать до конца строки, чтобы там найти ту самую открывающую скобку.

Правила расстановки скобок () рядом с операторами и функциями

- Не ставьте скобки сразу за операторами. Разделите оператор и скобки пробелом;
- Ставьте скобки непосредственно сразу за именем функции;
- Не используйте скобки в блоке return, если нет в этом необходимости.

Обоснование
Операторы - это не функции. Если поставить скобки сразу за оператором, функции и операторы будут выглядеть почти одинаково.

Пример

```
if (condition)
{
}

while (condition)
{

}

strcmp($s, $s1);

return 1;
```

Правила по отступам/табуляциям/ пробелам

- Отступ для каждого нового уровня - 3-4 пробела;
- Используйте не табуляцию, а пробелы. Большинство редакторов в состоянии пробелы заменить табуляцией;
- Делайте столько отступов, сколько вам нужно, но не более того. Если вы делаете отступ более, чем четвёртого-пятого уровня, стоит подумать о вынесении кода в отдельный блок.

Обоснование

- Когда люди используют разные значения табуляции, код бывает невозможно прочитать или распечатать, поэтому пробелы предпочтительнее;
- Никто никогда не договорится об оптимальном количестве пробелов. Просто будьте последовательны. 3-4 пробела - наиболее распространённое явление;

- С того момента, когда люди вознамерились ограничить количество уровней отступа, кажется, что ни разу они так ничего и не добились. Мы надеемся, что программисты сами примут мудрое решение о достаточной глубине отступов.

```
function func()
{
   if (something bad)    //не подходит
   {
      if (another thing bad)  //тоже не подходит
      {
         while (more input)  //ещё код
         {
         }
      }
   }
}
```

Форматирование блоков if then else

Внешний вид

На вкус программиста. Разный стиль расстановки фигурных скобок обусловит немного разный внешний вид условных блоков. Вот один из распространённых стилей:

```
if (condition)              // Комментарий
{
}
else if (condition)         // Комментарий
{
}
else                        // Комментарий
{
}
```

Если у вас в условном блоке есть else if, то стоит поставить else для всех необработанных значений. Даже если не предпринимаются никакие действия, это может быть простая запись в лог.

Формат условия

При сравнении всегда ставьте константы слева. Например:

```
if ( 6 == $errorNum ) ...
```

Первая причина такому поведению - это то, что парсер найдёт ошибку, если вы поставите только один знак равенства ('=') вместо двух. Вторая причина - при чтении кода вы находите нужное вам значение сразу в начале условия, а не ищете где-то в конце. К такому формату привыкаешь не сразу, но этот стиль действительно полезен.

Формат switch

- При наличии соответствующего комментария допускаются блоки, передающие управление вниз;
- Рекомендуется всегда ставить блок default, который бы сообщал об ошибке в случаях, когда попадание на него должно быть исключено, но, тем не менее, имело место;
- Если вам нужно создать какие-либо переменные, то весь соответствующий код ставьте внутри блоков case.

Пример

```
switch (...)
{
   case 1:
      ...

    // УПРАВЛЕНИЕ ПЕРЕДАЁТСЯ ВНИЗ

   case 2:
   {
      $v = get_week_number();
       ...
   }
   break;

   default:
}
```

Использование continue, break и ?:

Continue и break

`Continue` и `break` - это тот же самый `goto`, только названый по-другому. Именно поэтому они рассмотрены в этой части документа.

Как и `goto`, `continue` и `break` творят всякие разные чудеса в коде, поэтому их использование рекомендуется свести до минимума. Одним мановением руки читатель кода переносится бог знает куда по какой-то незадокументированной причине. При использовании `continue` возникают две проблемы:

- continue может обойти условный блок;
- continue может обойти наращивание/уменьшение.

Пример

Представим себе ситуацию, где имеют место обе проблемы:

```
while (TRUE)
{
  ...
  // Много кода
  ...

  if (/* какое-то условие */) {
    continue;
  }
  ...

  // Много кода
  ...
  if ( $i++ > STOP_VALUE) break;
}
```

Note: "много кода" нужно для того, чтобы программист не смог легко отследить проблему.

Из приведённого выше примера мы можем составить себе следующее правило: использование continue и break в одном блоке - прямая дорога к багам.

?:

Проблема обычно заключается в том, что люди пытаются запихать слишком много кода между ? и :. Вот несколько правил:

- Условие заключайте в скобки, тем самым отделяя его от остального кода;
- По возможности действия, производимые по условию, должны быть простыми функциями;
- Если весь блок ветвления плохо читается, будучи расположен на одной строке, то блоки else и then размещайте каждый на отдельной строке.

Пример

```
(условие) ? funct1() : func2();

or

(условие)
   ? длинный блок
   : ещё один длинный блок;
```

Выравнивание блоков объявления переменных

Блок объявления переменных должен выравниваться

Обоснование

- Прозрачность стиля;
- Блоки инициализации переменных также рекомендуется выравнивать табуляцией;
- Знак & должен ставиться при типе переменной, а не при её имени.

Пример

```
var      $mDate
var&     $mrDate
var&     $mrName
var      $mName

$mDate    = 0;
$mrDate   = NULL;
$mrName   = 0;
$mName    = NULL;
```

Несколько комментариев по комментариям

Комментарии должны быть содержательными

Воспринимайте комментарии как описание вашей системы. Будьте готовы к тому, что ваши комментарии будут извлечены роботом из текста исходника и оформлены в виде технического руководства. Комментарии к классам станут одной частью описания, сигнатуры методов - другой частью, аргументы методов - третьей, реализации методов - четвёртой. Все эти части должны слиться в единое целое и информировать удалённого в пространстве и времени читателя о том, что именно вы сделали и почему.

Документируйте принятые решения

Комментарии должны документировать принятые решения. Каждый раз, когда вы выбрали какой-либо способ реализации, поставьте комментарий, повествующий о том, что вы выбрали и почему. Для археологов это станет самой полезной информацией.

Используйте заголовки

Используйте систему автоматической генерации документации, такую как PHPDoc. В других разделах этого документа будут описаны приёмы использования PHPDoc для документирования классов и методов.

Структура заголовков обеспечивает их анализ и извлечение из исходника - структурированные заголовки уже приносят пользу, в отличие от обычных. Поэтому рекомендуется потратить немного времени на их заполнение - и документирование вам больше не понадобится.

Стиль комментариев

Каждая часть проекта имеет свой особый стиль комментариев.
Явно указывайте на gotchas ["ловушки" в коде]

Комментируйте как любые изменения переменных, не совсем вписывающиеся в нормальный ход программы, так и все конструкции, которые могут натворить дел при изменении кода. Для явного указания на проблемные или потенциально проблемные куски кода используйте встроенные зарезервированные слова.

Формат описания gotchas

Ключевое слово gotcha должно ставиться в самом начале комментария.
Комментарии могут содержаться на нескольких строках, но первая строка должна быть ясным и законченным по смыслу конспектом.

Комментарий должен включать в себя имя автора и дату замечания. Эти сведения содержатся и в документации исходников, но иногда бывает трудно их отыскать. Иногда случается, что потенциально опасный участок кода слишком долго не исправляется. Дата gotcha позволяет определить такие места. Указание на автора gotcha показывает, с кого нужно спросить.

Зарезервированные слова для описания gotchas

- **:TODO:**

  Означает, что здесь нужна доработка, простое напоминание.

- **:BUG: [id]**

  Означает, что здесь находится Выявленная ошибка, дайте описание и (не обязательно) номер ошибки.

- **:KLUDGE:**

  Если то, что вы сделали, уродливо выглядит и работает, отметьте этот факт и объясните, как вы поступите в следующий раз, если у вас будет время.

- **:TRICKY:**

  Сообщение о том, что данный отрезок кода очень сложен в исполнении, и потому не стоит ничего менять, не разобравшись предварительно во всех "коварствах" конструкции.

- **:WARNING:**

  Предупреждение о чём-либо.

- **:PARSER:**

  Иногда вам приходится что-то делать для успешного парсинга. Задокументируйте этот факт. Возможно, со временем проблема исчезнет.

- **:ATTRIBUTE:**

  Общий вид представления атрибута в комментарии.

  Вы можете создать свои атрибуты, и они тоже будут извлечены роботом.

Документация интерфейсов и реализаций

Существует два вида читателей для вашей документации:

- пользователи класса
- разработчики класса

Немного предусмотрительности, и мы сможем извлечь оба вида документации из исходного кода программы.

Пользователи класса

Пользователям класса необходимы сведения об интерфейсе класса; такие сведения легко добываются из заголовков, если, конечно, они правильно структурированы. При заполнении заголовочного блока комментариев, давайте только сведения, необходимые для использования класса. Не вдавайтесь в подробности реализации алгоритма - не надрывайтесь. Исключение составляют случаи, когда знание алгоритма необходимо для корректного использования класса. Воспринимайте заголовочный блок комментариев как без пяти минут главу из технического руководства.

Разработчики класса

Разработчикам класса требуется глубокое знание реализации класса. Комментарии этого типа находятся в файлах с исходным кодом класса; забудьте на время про особенности интерфейса. Заголовочный блок комментариев в исходнике должен описать все особенности алгоритма и решения по проектированию класса. Блоки комментариев внутри методов реализации должны предоставить ещё более подробную информацию.

Документация по директориям

В каждой директории проекта должен находиться файл `README` с описанием следующих моментов:

- Зачем была создана данная директория и что она содержит;
- По строчке описания каждого файла этой директории. Как правило, описание берётся из значения атрибута `NAME` в заголовке файла;
- Инструкции по сборке и установке;
- Ссылки на связанные источники: директории исходников, online документация, документация на бумажных носителях, документация по разработке;
- И всё, что как-либо может помочь.

Представьте себе, что через полгода после ухода последнего из зачинателей проекта в команду приходит новый человек. Этот одинокий и напуганный странник должен по кусочкам собрать и воссоздать полную картину проекта, пройдясь по директориям исходников и прочтя все `README`, make-файлы и заголовки в файлах исходников.

Повторное использование кода

Повторное использование кода за пределами одного проекта практически невозможно, если у вас нет разработанного проектного каркаса [framework]. Объекты строятся в соответствии с предоставляемыми сервисами. В разных проектах разные наборы сервисов, что и усложняет повторное использование объекта.

Разработка проектного каркаса отнимает много сил и времени. Но даже если по каким-то причинам вы не создали себе подобной системы, существует несколько приёмов поощрения повторного использования кода.

Не бойтесь маленьких библиотек

Один из врагов повторного использования кода - тот факт, что люди не составляют из своего кода библиотеки. Класс многократного использования может быть похоронен в директории одной из программ и может никогда не испытать волнующего чувства реинкарнации в новом проекте. И только потому, что программист не соизволил вынести этот класс (или классы) в библиотеку.

Одна из причин трагедии: люди не любят маленькие библиотеки. Есть в маленьких библиотеках нечто такое, что люди считают неправильным. Подавите в себе это чувство. Компьютеру абсолютно всё равно, сколько у вас библиотек.

Если вы написали код, который можно использовать повторно, но он не вписывается в вашу библиотеку, создайте новую. Если человек действительно стремится к многократному использованию, библиотеки недолго остаются маленькими.

Держите свою базу библиотек [репозиторий]

Большинство компаний не имеет никакого понятия, какой код у них есть. И большинство программистов до сих пор не сообщают о том, что они сделали, и не интересуются тем, что уже написано. Репозитории призваны изменить ситуацию к лучшему.

В идеальном мире программист мог бы зайти на сайт, посмотреть по каталогу или поиском найти нужный пакет библиотек и закачать себе. Если вы можете наладить такую систему, в которой программисты на добровольной основе будут поддерживать базу исходников - это прекрасно. Если вы заведёте библиотекаря, который бы отслеживал коэффициент повторного

использования, то это просто роскошно. Другой способ - автоматическая генерация репозитория из исходников. Достигается подобный эффект через использование стандартных заголовков для классов, методов, библиотек и различных подсистем. Такие заголовки служат одновременно техническим руководством и пунктами в списке репозитория.

Временное комментирование больших блоков

Иногда при тестировании возникает необходимость закомментировать большой блок кода. Самый простой способ - это заключение блока в конструкцию if(0):

```
function example()
{
   роскошный код

   if (0) {
      много кода
   }

   ещё больше кода
}
```

Комментарии /**/ вы использовать не можете, потому что комментарии не могут содержать комментарии, а большой блок вашего кода непременно будет содержать комментарии, ведь так?

Дополнение редакции

Предисловие к оригинальной статье

PHP Coding Standard

Last Modified: 2003-02-17

The PHP Coding Standard is with permission **based on** Todd Hoff's C++ Coding Standard.

Rewritten for PHP by Fredrik Kristiansen / DB Medialab, Oslo 2000-2003.

Using this Standard. If you want to make a local copy of this standard and use it as your own you are perfectly free to do so. That's why we made it! If you find any errors or make any improvements please e-mail me the changes so I can merge them in. Recent Changes.

Примеры использования PEAR

Maxim Matyukhin

Классы

Данная статья была написана с целью популяризации PEAR-классов среди php-программистов. Я постарался собрать здесь простые примеры того, как pear может
упростить и ускорить написание скриптов на PHP.

Пару слов о PEAR.
PEAR - это набор классов, написанных на PHP. Сейчас он содержит около 100 классов. Это число могло быть больше, но у pear-сообщества жесткие требования к чистоте и стилю оформления программного кода.
Официальный сайт pear: http://pear.php.net

Далее я просто покажу несколько примеров использования pear-классов.

1. HTML::Select
(http://pear.php.net/HTML_Select)
Данный класс позволяет генерировать SELECT-поля для форм. Вот наиболее типичный пример его использования:

```
<?
// хеш-массив значений 'текст' => 'значение'
$values = array(
    'value1'=>'1',
    'value 2' => '2',
    'value 3'=>'3');
$selected = '2'; // поле со значением '2' сделаем
//выделенным
// подключаем класс
require_once('HTML/Select.php');
// создаем экземпляр класса
$select = & new HTML_Select('select_field');
// загружаем данные, из которых будет
//генерироваться SELECT-поле
$select->loadArray($values, array($selected));
echo $select->toHtml(); // выводи HTML-код
?>
```

Это наиболее общий пример использования класса HTML::Select.
Конструктор в качестве параметра принимает имя поля (атрибут name тега SELECT)

Метод loadArray() принимает массив данных, из которых строится тег SELECT и массив значений, которые нужно выделить.
Результатом выполнения данного кода будет такой HTML:

```
<select name="select_field" size="1">
  <option value="1">value1</option>
  <option value="2"
selected="selected">value 2</option>
  <option value="3">value 3</option>
</select>
```

Класс также позволяет строить SELECT-поля на основе данных из базы. Для этого можно использовать метод loadQuery:

```
<?
// подключаем класс
require_once('HTML/Select.php');
// создаем экземпляр класса
$select = & new
HTML_Select('select_field');
$sql = 'SELECT id, title FROM tab_name';
$select->loadQuery($db, $sql, 'title', 'id');
echo $select->toHtml();
?>
```

Метод loadQuery() в качестве первого параметра принимает ссылку на экземпляр класса pear::DB или dsn-строку для создания такого объекта (подробности ищите в документации к pear::DB).

С недавнего времени появился еще набор классов с названием HTML_Select_Common (http://pear.php.net/HTML_Select_Common).
Этот набор классов позволяет просто создавать SELECT-поля со списком стран, штатов США и т.п.

Вот как выглядит создание тега SELECT со списком стран:

```
<?
require_once('HTML/Select/Common/Country.php');
  $country = new HTML_Select_Common_Country();
  echo $country->toHtml('country', 'ru');
?>
```

Данный пример создаст SELECT-тег со списком стран и выделит в нем Россию. Вот пример создания списка штатов США:

```
<?
require_once('HTML/Select/Common/USState.php');
  $state = new HTML_Select_Common_USState();
  echo $state->toHtml('state');
?>
```

2. HTTP::Download
(http://pear.php.net/HTTP_Download)
Данный класс позволяет создавать скрипты для скачивания файлов с поддержкой докачки. Можно организовать как скачивание файла из файловой системы,
так и скачивание файла, записанного в базу данных. Вот самый простой пример использования (взят из документации):

```
<?
        require_once('HTTP/Download.php');
        $params = array(
          'file' => '../hidden/download.tgz',
          'contenttype' => 'application/x-gzip',
 'contentdisposition' => array(HTTP_DOWNLOAD_ATTACHMENT, 'latest.tgz'));
$error = HTTP_Download::staticSend($params, false);
?>
```

Докачка будет организована автоматически.

3. HTML::BBCodeParser
(http://pear.php.net/HTML_BBCodeParser)
Те, кто сидят на форумах, наверное уже привыкли к BB-кодам, которые используются для оформления сообщений в форуме (например в phpBB, VBulletin и т.п).
Данный класс позволяет преобразовать текст с BB-кодом в HTML-код:

```
<?
require_once('HTML/BBCodeParser.php');
$text = '[b] Всем Привет [/b]
[i]Люди, как работать с pear-классами ? [/i]';
echo $text =
HTML_BBCodeParser::staticQparse($text);
?>
```

Если выполнить этот пример, то получите такой результат:

```
<b> Всем Привет </b>
<i>Люди, как работать с pear-классами ?
</i>
```

По умолчанию, этот класс обрабатывает следующие BB-коды:
- [b], [i], [u], [s], [sup], [sub]

Но класс также позволяет подключить подсветку e-mail, создание ссылок, изображений, списков и т.д. В архиве с классом идет пример, который показывает, как использовать все эти возможности.

4. HTML_Crypt
(http://pear.php.net/HTML_Crypt).
Мне этот класс показался оригинальным, поэтому я включил его в обзор.
Этот класс позволяет зашифровать строку на PHP, которую в последствии можно будет расшифровать Яваскриптом на стороне клиента. В основном этот класс применяется для шифрования e-mail от

спамерских роботов:

```
<?
require_once('HTML/Crypt.php');
$c = new HTML_Crypt();
$c->obStart();
echo " Это пример использования класса
HTML_Crypt, автор которого
<a href=\"mailto:mike@blueroot.net\">Michael
Dransfield</a><br>";
echo "Комментарии к статье шлите на
<a
href=\"mailto:max@webscript.ru\">max@webscri
pt.ru</a>";
$c->obEnd();
?>
```

Результатом будет такой HTML-код (листинг 1):

```
Это пример использования класса HTML_Crypt, автор которого
<script language="JavaScript" type="text/JavaScript">
  var a,s,n;
  function ja4fc707c63aa221b5326b8c0cb348e87(s){
    r='';
    for(i=0;i<s.length;i++){
      n=s.charCodeAt(i);
      if(n>=8364){n=128;}
      r+=String.fromCharCode(n-3);
    }
    return r;
  }
  a='?d#kuhi@%pdlowr=plnhCeoxhurrw1qhw%APlfkdho#Gudqvilhog?2dA';
  document.write (ja4fc707c63aa221b5326b8c0cb348e87(a));
</script>
<br>Комментарии к статье шлите на
<script language="JavaScript" type="text/JavaScript">
  var a,s,n;
  function da4fc707c63aa221b5326b8c0cb348e87(s) {
    r='';
    for(i=0;i<s.length;i++){
      n=s.charCodeAt(i);
      if(n>=8364){n=128;}
      r+=String.fromCharCode(n-3);
    }
    return r;
  }
  a='?d#kuhi@%pdlowr=pd{Czhevfulsw1ux%Apd{Czhevfulsw1ux?2dA';
  document.write (da4fc707c63aa221b5326b8c0cb348e87(a));
</script>
```

Листинг 1

Отступы я расставил сам (на самом деле весь js-код выводится в одну строку). Если вы откроете этот HTML-код в броузере, то там увидите нормальный e-mail.

5. HTML::treeMenu

(http://pear.php.net/HTML_treeMenu)

Один из моих любимых классов :-) Позволяет строить древовидные DHTML-меню. Если броузер не поддерживает работу со слоями, то в нем просто будет выведено раскрытое дерево. Пример создания меню приведен в листинге 2.

```
<?php
  require_once('HTML/TreeMenu.php');

  $icon = 'folder.gif';
  $expandedIcon = 'folder-expanded.gif';

  $menu = new HTML_TreeMenu();

  $node1 = new HTML_TreeNode(array(
    'text' => "First level", 'link' => "test.php", 'icon'=>$icon));
  $node1_1 = &$node1->addItem(new HTML_TreeNode(array(
    'text' => "Second level", 'link' => "test.php", 'icon'=>$icon)));
  $node1_1_1 = &$node1_1->addItem(new HTML_TreeNode(array(
    'text' => "Third level", 'link' => "test.php", 'icon'=>$icon)));
  $node1_1_1_1 = &$node1_1_1->addItem(new HTML_TreeNode(array(
    'text' => "Fourth level", 'link' => "test.php", 'icon'=>$icon)));
  $node1_1_1_1->addItem(new HTML_TreeNode(array(
    'text' => "Fifth level", 'link' => "test.php", 'icon'=>$icon)));

  $node1->addItem(new HTML_TreeNode(array(
    'text' => "Second level, item 2", 'link' => "test.php", 'icon'=>$icon)));
  $node1->addItem(new HTML_TreeNode(array(
    'text' => "Second level, item 3", 'link' => "test.php", 'icon'=>$icon)));

  $menu->addItem($node1);
  $menu->addItem($node1);

  // Create the presentation class
  $options = array('images' => './images');
  $treeMenu = &new HTML_TreeMenu_DHTML($menu, $options);
?>
<html>
<head>
  <script src="TreeMenu.js" language="JavaScript"></script>
</head>
<body>
<?$treeMenu->printMenu()?>
</body>
</html>
```

Листинг 2

Пример взят из документации, правда, я его немножко упростил, чтобы не испугать новичков.

6. Archive::Tar
(http://pear.php.net/Archive_Tar)
Данный класс позволяет создавать архивы типа .tar.gz. Многие, наверное, сталкивались с проблемой, что с помощью расширения ZLib

нельзя заархивировать несколько файлов в один архив. Данный класс решает данную проблему. Вот типичный пример использования класса:

```
<?
include_once("Archive/Tar.php");
$tar = & new Archive_Tar("archive.tar.gz", true);
$tar->create("/dir/to/archive/"); // добавляем каталог со
//всеми его файлами в архив
$tar->add('file.txt'); // добавляем файл в архив
?>
```

Данный код создает архив archive.tar.gz, добавляет в него содержимое каталога /dir/to/archive/ и файл file.txt .
Класса для работы с zip-архивом пока нет. Но, насколько мне известно, он пишется.

7. MP3::ID
(http://pear.php.net/MP3_ID)
Так как часто на форумах спрашивают, как определить параметры MP3-файла, решил включить этот класс в обзор. MP3::ID позволяет читать и писать ID3-теги MP3-файла.

Вот пример использования:

```
<?
require_once("MP3/Id.php");
$file = "Triplex_Brigada.mp3";

$id3 = &new MP3_Id();
$id3->read($file);
echo 'Artist : '.$id3->getTag('artists');
echo 'Name: '.$id3->getTag('name');
?>
```

Ну вот и все. На первый раз, думаю, достаточно.

Ссылки по теме:
http://pear.php.net - Официальный сайт PEAR.
http://news.php.net/group.php?group=php.pear.dev - очень полезная конференция по pear

Введение в FPDF.

Автор: **nw**

На базе официальной документации fpdf и Adobe Sys. Inc

1. Что такое PDF и как его можно прикрутить к web-ориентированным приложениям средствами PHP

В наше время формат документов PDF приобретает большую популярность. Он был разработан компанией Adobe Systems Incorporated. Как указано в документации, THE ADOBE PORTABLE DOCUMENT FORMAT (PDF) – переносимый формат документов, является «родным» для программных продуктов семейства Adobe Acrobat. Их цель – дать пользователю возможность легко обмениваться электронными документами и просматривать их независимо от той среды, в которой эти документы были созданы. PDF опирается на графическую модель, позволяющую отображать картинки и текст вне зависимости от установленных на компьютере устройств и разрешения. В документах этого формата присутствуют такие объекты, как гиперссылки и аннотации, что делает их интерактивными.

С другой стороны, web-приложения зачастую нуждаются в отображении динамически составленных документов, таких как отчеты, прайслисты, счета и многое другое. Это позволяет персонализировать приложение и сделать его более мощным по своим функциональным возможностям. Помимо PDF, существуют и другие решения, но этот формат можно назвать одним из самых удачных, так как PDF документ без потери форматирования можно вывести на принтер или конвертировать в HTML или текст.

PHP, как один из самых мощных и популярных современных средств разработки web-приложений, справляется с задачей генерации PDF документов «на лету». Для этого разработано несколько дополнительных инструментов. Не возьмусь перечислить их все, но назову одни из самых известных – библиотеку PDFLib, ClibPDF и PHP класс FPDF.

2. FPDF – только PHP

Названные в предыдущей главе PDFLib и ClibPDF требуют дополнительной настройки PHP, в то время как класс FPDF является чистым PHP кодом и легко подключается к скриптам командой include() и другими подобными. Скачать класс и ознакомиться с подробной документацией можно на сайте www.fpdf.org. Дополнительным (порой решающим) аргументом в пользу этого решения можно рассматривать его бесплатность для использования как в личных, так и коммерческих целях. Цитата из лицензионного соглашения:

«FPDF is Freeware (it is stated at the beginning of the source file). There is no usage restriction. You may embed it freely in your application (commercial or not), with or without modification».

Разрешается также видоизменять исходный код класса. Никаких ограничений.

3. Решение проблемы с кириллицей

При создании русскоязычных документов средствами иностранных программных продуктов (библиотек, приложений и прочего) часто возникает проблема правильного отображения кириллических шрифтов. Не всякий зарубежный продукт корректно работает (а то и вовсе не работает) с кириллицей. К счастью, класс FPDF не принадлежит к их числу и легко настраивается на работу с русским языком.
Если быть точным, то сам класс настраивать практически не придется. Проблема может возникнуть с файлами кириллических шрифтов. Оговорюсь, что тестирование класса я проводил на wintel платформе (впрочем, весь приведенный

код работал и на коммерческом *nix хостинге). В windows одним из самых основных форматов шрифтов (наряду с PostScript) является TTF (True Type Font). Но для правильной работы наших скриптов необходим и еще один формат файлов - AFM (файл метрики шрифта). Как считается, AFM файлы поставляются вместе с TTF. В своей ОС я AFM файлов не обнаружил.

Здесь нам на помощь приходят полезные софтины, в частности – ttf2pt1. Одна из задач данной утилиты – сгенерировать метрический файл для True Type или PFB. Другими словами, появляется возможность взять из директории /fonts (ОС Windows) любой .TTF файл шрифта с поддержкой кириллицы и получить для него метрику при помощи нашей волшебной утилиты. Скачать утилиту можно по следующим линкам: http://ttf2pt1.sourceforge.net и http://fpdf.org/fr/dl.php?id=22 (для Windows).

После того, как утилита скачана, ее необходимо запустить из командной строки (в windows Пуск->Выполнить команду cmd). Формат вызова утилиты для нужной нам цели выглядит следующим образом:

```
ttf2pt1 -A font.ttf font
```

К примеру, если вы положили скачанный экземпляр ttf2pt1 прямо на диск C:\ , а файл шрифта times.ttf в C:\CyrFonts, то вам будет необходимо запустить следующую команду:

```
c:\ttf2pt1 -A c:\CyrFonts\times.ttf times
```

где c:\ttf2pt1 – вызов программы, -A – ключ, указывающий на необходимость сформировать файл AFM, c:\CyrFonts\times.ttf – это адрес файла True Type шрифта и, наконец, times – это имя будущего метрического файла. Итак, AFM файл готов.

Следующим шагом является генерация файла описания шрифта. Этот файл будет иметь знакомое нам расширение - PHP. Вместе с классом FPDF поставляется полезный скрипт для решения этой задачи. Его можно найти в директории font/makefont/ класса. Использовать его просто. Для этого создадим PHP файл (скажем, mf.php) и в нем укажем:

```
<?php
require('font/makefont/makefont.php');
MakeFont('times.ttf',times.afm','cp125
1');
?>
```

Используя require, мы подключаем нужный скрипт. Понятно, что для этого рядом с нашим файлом должна быть папка font, содержащая в себе makefont/makefont.php. А вот функция MakeFont() уже является специфической и по определению имеет следующий формат:

MakeFont(**string** fontfile, **string** afmfile [, **string** enc [, **array** patch [, **string** type]]])

где fontfile – путь к TTF или PFB файлу, afmfile – путь к AFM файлу, enc – имя используемой кодировки (по умолчанию это cp1252), patch – опциональное изменение кодировки и type – тип шрифта (по умолчанию True Type). Для выбора кодировки можно воспользоваться следующим списком:

- cp1250 (Central Europe)
- cp1251 (Cyrillic)
- cp1252 (Western Europe)
- cp1253 (Greek)
- cp1257 (Baltic)
- ISO-8859-1 (Western Europe)
- ISO-8859-2 (Central Europe)
- ISO-8859-4 (Baltic)

ISO-8859-5 (Cyrillic)

- ISO-8859-7 (Greek)
- ISO-8859-15 (Western Europe)
- ISO-8859-16 (Central Europe)
- KOI8-R (Cyrillic)

Кодировка определяет связь между кодом (от 0 до 255) и символом. Для выбора кириллической кодировки в Windows используйте cp1251. Обычно кодировки с префиксом cp используются в Windows, в то время как Linux системы используют ISO.

Составленный нами скрипт mf.php необходимо открыть в браузере. Он подготовит для наших нужд необходимый файл с расширением php. Итак, что мы имеем? У нас теперь есть комплект из трех файлов шрифта – times.ttf, times.afm и times.php. Важно два из них (times.ttf и times.php) положить в нужное место. Этим местом является директория font, находящаяся в папке класса. Впрочем, вы вольны сами указать место директории, которая будет хранить шрифты. Для этого нужно определить константу FPDF_FONTPATH обыкновенным для PHP способом:

```
define('FPDF_FONTPATH','font/');
```

Теперь наша система готова к разработке web приложения с динамической генерацией русскоязычного PDF документа.

4. Начинаем работу

Для наглядного иллюстрирования возможностей FPDF класса попробуем создать реальный документ – прайс-лист фирмы, занимающейся оптово-розничной продажей алкогольных напитков. Краткое техническое задание следующее:

- в шапке документа должны быть данные: логотип, наименование фирмы, заголовок документа;
- в теле документа должны быть приведены данные по товарным позициям, включающие наименование товара, розничную цену и оптовую цену;

Сразу разобьем нашу работу на три этапа. Первый этап – вывод статической информации. Для простоты примера к статической информации мы отнесем все, кроме данных по товарным позициям. На втором этапе уделим внимание табличному выводу товарных позиций в теле документа. Уточним, что в этой статье мы рассмотрим загрузку данных из файла CSV, где разделителем является точка с запятой. Я остановился на этом решении по одной причине – такой файл легко получить из формата xls и, одновременно, с ним легко работать из PHP приложения в других целях (например, организовать вывод в HTML). На третьем этапе мы рассмотрим доставку PDF файла конечному пользователю.

Приступим к практическому знакомству с классом FPDF. Для начала создадим файл price.php, который будет осуществлять вывод PDF документа прямо в браузер (остальные способы мы рассмотрим в главе 6). Рядом с этим рабочим файлом положим скачанный ранее fpdf.php (файл класса) и папку font с вложенными в нее файлами кириллических шрифтов (см. предыдущую главу). Теперь в файле price.php подключим класс FPDF и установим путь к папке шрифтов.

```
<?php
   define('FPDF_FONTPATH','font/');
   require('fpdf.php');
?>
```

Необходимо уточнить, что тэг <?php должен стоять на самой первой строке файла, так как в последствии мы будем самостоятельно отправлять

заголовки (headers), и пустая строка, текст или разметка HTML в начале файла могут нам помешать.

Для вывода статической информации (логотип, название фирмы, название документа) мы будем использовать следующие методы класса FPDF: cell() и image(). Метод cell() выводит ячейку (прямоугольную фигуру) с опциональной установкой границы, цветовой заливкой и строкой текста. Формат записи метода следующий:

Cell(**float** w [, **float** h [, **string** txt [, **mixed** border [, **int** ln [, **string** align [, **int** fill [, **mixed** link]]]]]]])

Метод image() отвечает за вывод графического изображения JPG или PNG. Формат вызова метода следующий:

Image(**string** file, **float** x, **float** y, **float** w [, **float** h [, **string** type [, **mixed** link]]])

Конечно, перед тем, как воспользоваться методами класса, сначала необходимо создать экземпляр этого класса. Приведем наш файл к следующему виду:

```
<?php
	define('FPDF_FONTPATH','font
/');
	require('fpdf.php');

	//Создадим экземпляр класса
	$price = new FPDF();
	$price->Open();
?>
```

После создания экземпляра класса нам будет необходимо указать используемые шрифты. Так как кириллический Times New Roman (рассмотренный в предыдущей главе) не является в классе FPDF шрифтом, установленным по умолчанию, сначала придется его «показать» скрипту. Это мы сделаем при помощи метода AddFont().

```
<?php
define('FPDF_FONTPATH','font/');
require('fpdf.php');

//Создадим экземпляр класса
	$price = new FPDF();
	$price->Open();
//Подключаем кириллический шрифт
$price->
AddFont('TimesNewRomanPSMT','','times.
php');
$price->
SetFont('TimesNewRomanPSMT','',12);
?>
```

Первым аргументом функции мы указываем наименование шрифта. Его можно посмотреть в сгенерированном PHP файле (значение переменной $name). Второй аргумент – форматирование текста (B – Bold, I – Italic и смешанный BI или IB). Если аргумент пустой, то шрифт обычный. Для использования B и I необходимо также подключить соответствующие типы шрифтов (для Times New Roman это могут быть файлы timesI.ttf и timesBd.ttf). Третий аргумент – PHP файл описания (его мы сгенерировали в предыдущей главе). Теперь шрифт можно применять в данном документе. Для использования на странице установим его размер методом SetFont(). Этот метод можно вызывать несколько раз в одном скрипте, в то время как добавление AddFont() делается один раз для каждого шрифта. Формат записи SetFont() следующий:

SetFont(**string** family [, **string** style [, **float** size]])

Наконец-то можно сформировать содержимое страницы. На этом этапе мы столкнемся с увеличением количества кода, и поэтому, во избежание путаницы, расширим класс FPDF своими методами. Конечно, нет необходимости вносить изменения в сам FPDF класс. Мы напишем свой price.class.php, наследуемый от FPDF.

```
<?php
class Price extends FPDF {
function PrintTitle($title,$image,$company) {
                //Выводим логотип
        $this->Image($image,6,6,40,20);
                //Устанавливаем шрифт для наименования компании
        $this->SetFont('TimesNewRomanPSMT','',20);
                //Выводим наименование компании
        $this->Cell(210,4,$company,0,0,'C');
                //Переходим на следующую строку
        $this->Ln();
                //Переходим на следующую строку
        $this->Ln();
                //Делаем отступ от левого края (рисуя прозрачную
ячейку)
        $this->Cell(37);
                //Устанавливаем цвет заливки следующих ячеек
(R,G,B)
        $this->SetFillColor(209,204,244);
                //Устанавливаем шрифт для наименования документа
        $this->SetFont('TimesNewRomanPSMT','',12);
                //Выводим наименование документа
        $this->Cell(150,8,$title,0,0,'C',1);
                //Переходим на следующую строку
        $this->Ln();
    }
}
?>
```

Листинг 1

Итак, создадим новый файл, в который запишем код с листинга 1.

При этом наш основной файл price.php примет вид (листинг 2):

```
<?php
    define('FPDF_FONTPATH','font/');
    require('fpdf.php');
    require('price.class.php');

$price = new Price();
$price->Open();
$price->AddFont('TimesNewRomanPSMT','','times.php');
    //Добавляем страничку в документ
$price->AddPage();
    //Выводим заголовок, используя написанный нами метод в файле класса
    //price.class.php
$price->PrintTitle('Прайс-лист','logo.jpg','Компания "ALKO SELL"');
    //Выводим документ в браузер
$price->Output();
?>
```

Листинг 2

В этом файле мы столкнулись еще с несколькими методами класса FPDF. Необходимо заметить, что документ сначала создается в буфере и лишь потом, при вызове метода Output(), выводится в браузер. Поэтому общая схема работы с документом следующая: создаем в буфере документ методом Open(), затем для работы добавляем в этот документ страничку методом AddPage(), формируем содержимое документа различными методами типа Cell() и, наконец, выводим его из буфера в браузер. Если у вас установлен adobe acrobat версии не ниже 5.0, то при запуске price.php в браузере должен открыться документ price.pdf (в Internet Explorer) или сформируется документ doc.pdf для скачивания (в Opera).

Этот документ пока не представляет собой практической ценности, однако на его примере мы разобрали основы вывода статических данных в документ .PDF.

5. Подключаем источник данных

Перейдем к самому интересному – динамическому формированию содержимого документа. Именно этой возможностью и ценен для PHP разработчика класс FPDF. Я уже говорил выше, что здесь мы рассмотрим вариант получения данных из CSV файла.

Как указано в нашем коротком техническом задании, тело прайс-листа должно состоять из трех граф: наименование товара, розничная цена и оптовая цена. Соответственно, подготовим и файл данных следующего вида:

Пиво «Балтика»;15.00;13.45
Пиво «Оболонь»;14.00;12.30
Водка «Nemiroff», 1 л.;100.00;89.45

Сохраним его как price.csv. Конечно, в реальном прайс-листе товарных позиций и граф будет больше.

Возможны так же деления по группам товаров (например «пиво», «водка», «вино»), но в данном конкретном примере попробуем рассмотреть упрощенную версию (количество записей можно увеличить).

Перед нами встает необходимость получить данные динамически и представить их в виде обыкновенной таблицы в три колонки, с заголовками колонок (столбцов). Перед созданием экземпляра объекта Price добавим инициализацию массива заголовков (соберем все заголовки в массив):

```
$header =
array("Наименование","Розн.","Опт.");
```

Далее нам понадобится еще больше расширить класс Price (унаследованный от класса FPDF) методами LoadData() и ImprovedTable(). Первый метод должен будет извлекать данные из CSV файла в массив $data, а второй метод пригодится для отрисовки таблицы с данными из этого массива. Рассмотрим метод LoadData() для класса Price.

```
function LoadData($file)
{
   //Получаем данные в массив строк
   $lines=file($file);
   $data=array();
//Разделяем столбцы каждой строки по точке с запятой.
   foreach($lines as $line)

$data[]=explode(';',chop($line));
//Возвращаем массив с данными
   return $data;
}
```

Вставим этот код в файл класса price.class.php. Теперь перейдем к рассмотрению метода ImprovedTable().

Вот его код:

```
function ImprovedTable($header,$data)
{
   //Указываем ширину столбцов
   $w=array(100,40,40);

   //Выводим заголовки столбцов
   for($i=0;$i<count($header);$i++)
      $this-
>Cell($w[$i],7,$header[$i],1,0,'C');
   $this->Ln();

//Выводим данные
//Сначала установим шрифт для данных
$this->
SetFont('TimesNewRomanPSMT','',8);
   foreach($data as $row)
   {
/*Первый параметр Cell() – ширина столбца, указанная ранее в массиве $w, второй параметр – высота столбца, третий параметр – строка для вывода, LRBT – означает прорисовку границ со всех сторон ячейки (Left, Right, Bottom, Top). Можно также указать выравнивание в ячейке по правому краю (‘R’) */

//Рисуем ячейку наименования товара
$this->Cell($w[0],6,$row[0],'LRBT');

//Рисуем ячейку розничной цены
 $this->
Cell($w[1],6,number_format($row[1]),'L
RBT',0,'R');

//Рисуем ячейку оптовой цены
 $this->
Cell($w[2],6,number_format($row[2]),'L
RBT',0,'R');

//Переходим на следующую строку
      $this->Ln();
   }
   //Closure line
   $this->Cell(array_sum($w),0,'','T');
}
```

Вставим и этот метод в код класса Price. Теперь, когда мы обросли двумя новыми методами, попробуем их применить. Изменим код нашего главного файла (листинг 3).

Можно смотреть на промежуточный результат. Это практически все, чего мы хотели достичь.

Добавим только небольшую деталь – номера страниц с указанием их общего числа в нижней части каждой страницы. Ведь наш источник данных может состоять не из трех записей, а, например, из ста.

Для этого необходимо организовать вывод «подвала» (footer) на каждой странице. С этой целью напишем метод Footer() нашего класса Price. Следует отметить, что методы Footer() и Header() заранее прописаны в классе FPDF и вызываются автоматически при выполнении метода AddPage() и Close().

По умолчанию они пустые, но их можно определить в своих наследуемых классах, при этом ни Footer() ни Header() не потребуется явно вызывать из приложения. Добавим в класс Price() следующий код метода:

```
function Footer()
{
   //Позиционирование в  1.5 см
от нижней границы
   $this->SetY(-15);
         //TimesNewRomanPSMT
italic 8
   $this-
>SetFont('TimesNewRomanPSMT','
',8);
   //Номер страницы
   $this->Cell(0,10,'Страница
'.$this->PageNo().'/{nb}',0,0,'C');
}
```

```
<?php
	define('FPDF_FONTPATH','font/');
	require('fpdf.php');
	require('price.class.php');

$header = array("Наименование товара","Розн. цена","Опт. цена");
$price=new Price();
$price->Open();

	//Получим массив данных из файла в $data
$data = $price->LoadData("price.csv");
$price->AddFont('TimesNewRomanPSMT','','times.php');
$price->AddPage();
$price->SetFont('TimesNewRomanPSMT','',12);
$price->PrintTitle('Прайс-лист','logo.jpg','Компания "ALKO SELL"');

//Нарисуем таблицу. Аргументами метода являются массив наименований
// столбцов и массив данных из файла price.csv
$price->ImprovedTable($header,$data);
$price->Output();
?>
```

Листинг 3

Теперь этот метод будет автоматически исполняться при вызове метода AddPage(). Для корректной работы необходим последний штрих – вызов из файла приложения (price.php) метода $price->AliasNbPages() перед $price->AddPage().

AliasNbPages([**string** alias])

При написании метода Footer() мы использовали также PageNo(), метод, возвращающий номер текущей страницы и параметр {nb}, который по умолчанию будет заменен цифрой общего количества страниц в текущем документе. Документ готов, и перед нами встает необходимость его вывода в браузер.

6. Вывод в браузер

По идее, существует всего два варианта вывода документа в браузер – открытие (если установлен adobe acrobat) и скачивание без непосредственного открытия. В классе FPDF выводом документа в браузер управляет метод Output(). В нашем примере мы использовали этот метод без дополнительных аргументов. Однако документация к FPDF приводит следующий формат его записи:

Output([**string** file [, **boolean** download]])

Метод предназначен для сохранения PDF документа в локальный файл или для непосредственного вывода в браузер (если установлена программа просмотра PDF файлов).
Аргумент file означает имя файла. Если таковое отсутствует, то производится попытка открыть документ в окне браузера. Если аргумент file определен, то аргумент download указывает, что файл должен быть сохранен на сервере (значение false) или у пользователя (при установке true выводится диалог «Сохранить как»).

Соответственно, можно выделить три варианта написания метода Output() для нашего документа. Первый – Output() – пытается открыть документ в окне браузера. Второй вариант написания выведет у пользователя диалоговое окно «Сохранить как» и предложит скачать документ на его диск – Output("AlkoPrice.pdf", true). И, наконец, третий вариант просто сохранит документ на локальном для скрипта сервере – Output("AlkoPrice.pdf",false) или просто Output("AlkoPrice.pdf"), так как по умолчанию атрибут download всегда имеет значение false.

Стоит отдельно рассмотреть компрессию полученного файла. По умолчанию будет произведена попытка его «ужать» средствами Zlib. Это расширение (extension) должно быть установлено в системе. Если установка не была произведена, то документ получится не сжатым, и будет весить немного больше.

Для начала, пожалуй, все!

Авторы: **Михаил Еремеев** и **Сергей [msa78@mail.ru]**

1. Обзор технологий баз данных

1.1. Функции СУБД

Любая современная СУБД, независимо от производителя, выполняет все или некоторые нижеприведенные функции:

- **Непосредственное управление данными во внешней памяти** - эта функция включает обеспечение необходимых структур внешней памяти как для хранения данных, непосредственно входящих в БД, так и для служебных целей, например, для убыстрения доступа к данным в некоторых случаях (обычно для этого используются индексы). В некоторых реализациях СУБД активно используются возможности существующих файловых систем, в других работа производится вплоть до уровня устройств внешней памяти. В развитых СУБД пользователи в любом случае не обязаны знать, использует ли СУБД файловую систему, и если использует, то как организованы файлы. В частности, некоторые СУБД поддерживает собственную систему именования объектов БД.
- **Управление буферами оперативной памяти** - СУБД обычно работают с БД значительного размера; по крайней мере, этот размер обычно существенно больше доступного объема оперативной памяти. Понятно, что если при обращении к любому элементу данных будет производиться обмен с внешней памятью, то вся система будет работать со скоростью устройства внешней памяти. Практически единственным способом реального увеличения этой скорости является буферизация данных в оперативной памяти. При этом, даже если операционная система производит общесистемную буферизацию (как в случае ОС UNIX), этого недостаточно для целей СУБД, которая располагает гораздо большей информацией о полезности буферизации той или иной части БД. Поэтому в развитых СУБД поддерживается собственный набор буферов оперативной памяти с собственной дисциплиной замены буферов. Существует отдельное направление СУБД, которое ориентировано на постоянное присутствие в оперативной памяти всей БД. Это направление основывается на предположении, что в будущем объем оперативной памяти компьютеров будет настолько велик, что позволит не беспокоиться о буферизации.
- **Управление транзакциями.** Транзакция - это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется, и СУБД фиксирует (COMMIT) изменения БД, произведенные этой транзакцией, во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД. Понятие транзакции необходимо для поддержания логической целостности БД. Поддержание механизма транзакций является обязательным условием даже однопользовательских СУБД (если, конечно, такая система заслуживает названия СУБД). Понятие транзакции еще более важно в многопользовательских СУБД. То свойство, что каждая транзакция начинается при целостном состоянии БД и оставляет это состояние целостным после своего завершения, делает очень удобным использование понятия транзакции как единицы активности пользователя по отношению к БД. При соответствующем управлении параллельно выполняющимися

транзакциями со стороны СУБД каждый из пользователей может в принципе ощущать себя единственным пользователем СУБД (на самом деле, это несколько идеализированное представление, поскольку в некоторых случаях пользователи многопользовательских СУБД могут ощутить присутствие своих коллег).

- **Журнализация.** Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя. Обычно рассматриваются два возможных вида аппаратных сбоев: так называемые мягкие сбои, которые можно трактовать как внезапную остановку работы компьютера (например, аварийное выключение питания), и жесткие сбои, характеризуемые потерей информации на носителях внешней памяти. Примерами программных сбоев могут быть: аварийное завершение работы СУБД (по причине ошибки в программе или в результате некоторого аппаратного сбоя) или аварийное завершение пользовательской программы, в результате чего некоторая транзакция остается незавершенной. Первую ситуацию можно рассматривать как особый вид мягкого аппаратного сбоя; при возникновении последней требуется ликвидировать последствия только одной транзакции.

Понятно, что в любом случае для восстановления БД нужно располагать некоторой дополнительной информацией. Другими словами, поддержание надежности хранения данных в БД требует избыточности хранения данных, причем та часть данных, которая используется для восстановления, должна храниться особо надежно.

Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД. Журнал (лог) - это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью (иногда поддерживаются две копии журнала, располагаемые на разных физических дисках), в которую поступают записи обо всех изменениях основной части БД. В разных СУБД изменения БД журналируются на разных уровнях: иногда запись в журнале соответствует некоторой логической операции изменения БД (например, операции удаления строки из таблицы реляционной БД), иногда - минимальной внутренней операции модификации страницы внешней памяти; в некоторых системах одновременно используются оба подхода. Во всех случаях придерживаются стратегии "упреждающей" записи в журнал (так называемого протокола Write Ahead Log - WAL). Грубо говоря, эта стратегия заключается в том, что запись об изменении любого объекта БД должна попасть во внешнюю память журнала раньше, чем измененный объект попадет во внешнюю память основной части БД. Известно, что если в СУБД корректно соблюдается протокол WAL, то с помощью журнала можно решить все проблемы восстановления БД после любого сбоя.

Для восстановления БД после сбоя используют журнал и архивную копию БД. Грубо говоря, архивная копия - это полная копия БД к моменту начала заполнения журнала (имеется много вариантов более гибкой трактовки смысла архивной копии). Конечно, для нормального восстановления БД после жесткого сбоя необходимо, чтобы журнал не пропал. Как уже отмечалось, к сохранности журнала во внешней памяти в СУБД предъявляются особо повышенные требования.

Тогда восстановление БД состоит в том, что исходя из архивной копии по журналу воспроизводится работа всех транзакций, которые закончились к моменту сбоя. В принципе, можно даже воспроизвести работу незавершенных транзакций и продолжить их работу после завершения восстановления. Однако в реальных системах это обычно не делается, поскольку процесс восстановления после сбоя является достаточно длительным.

- **Поддержка языков БД** - для работы с базами данных используются специальные языки, в целом называемые языками баз данных. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков. Чаще всего выделялись два языка: язык определения схемы БД (SDL - Schema Definition Language) и язык манипулирования данными (DML - Data Manipulation Language). SDL служил, главным образом, для определения логической структуры БД, т.е. той структуры БД, какой она представляется пользователям. DML содержал набор операторов манипулирования данными, т.е. операторов, позволяющих заносить данные в БД, удалять, модифицировать или выбирать существующие данные.

В современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс с базами данных. Стандартным языком наиболее распространенных в настоящее время реляционных СУБД является язык SQL (Structured Query Language). В нескольких лекциях этого курса язык SQL будет рассматриваться достаточно подробно, а пока мы перечислим основные функции реляционной СУБД, поддерживаемые на "языковом" уровне (т.е. функции, поддерживаемые при реализации интерфейса SQL).

Прежде всего, язык SQL сочетает средства SDL и DML, т.е. позволяет определять схему реляционной БД и манипулировать данными. При этом именование объектов БД (для реляционной БД - именование таблиц и их столбцов) поддерживается на языковом уровне в том смысле, что компилятор языка SQL производит преобразование имен объектов в их внутренние идентификаторы на основании специально поддерживаемых служебных таблиц-каталогов. Внутренняя часть СУБД (ядро) вообще не работает с именами таблиц и их столбцов. Язык SQL содержит специальные средства определения ограничений целостности БД. Опять же, ограничения целостности хранятся в специальных таблицах-каталогах, и обеспечение контроля целостности БД производится на языковом уровне, т.е. при компиляции операторов модификации БД компилятор SQL на основании имеющихся в БД ограничений целостности генерирует соответствующий программный код. Специальные операторы языка SQL позволяют определять так называемые представления БД, фактически являющиеся хранимыми в БД запросами (результатом любого запроса к реляционной БД является таблица) с именованными столбцами. Для пользователя представление является такой же таблицей, как любая базовая таблица, хранимая в БД, но с помощью представлений можно ограничить или, наоборот, расширить видимость БД для конкретного пользователя. Поддержание представлений производится также на языковом уровне.

Наконец, авторизация доступа к объектам БД производится также на основе специального набора операторов SQL. Идея состоит в том, что для выполнения операторов SQL разного вида пользователь должен обладать различными полномочиями. Пользователь, создавший таблицу БД, обладает полным набором полномочий

для работы с этой таблицей. В число этих полномочий входит полномочие на передачу всех или части полномочий другим пользователям, включая полномочие на передачу полномочий. Полномочия пользователей описываются в специальных таблицах-каталогах, контроль полномочий поддерживается на языковом уровне.

1.2. Типы баз данных

Базы данных поддерживаемые СУБД могут быть следующих типов: реляционные, нереляционные (и иерархические) и объектно-ориентированные.

Наибольшее распространение получили реляционные базы данных. В таких базах данные хранятся в двухмерных таблицах, связанных между собой логическими связями (relations).

В нереляционных базах данные хранятся не в таблицах, а в виде пар **ключ—значение.** Причем, эти значения никак логически не связаны между собой.

Иерархические базы данных являются подвидом нереляционных баз данных. Данные также хранятся в виде пар **ключ-значение.** Но группы таких пар дополнительно упорядочиваются с помощью путем иерархического вкладывания групп в логические контейнеры, которые, в свою очередь, тоже могут быть вложены друг в друга. Таким образом, возникает возможность по логической организации хранимых данных.

Объектно-ориентированные базы данных представляют хранилища объектов, с помощью которых осуществляется описание предметной области информационного проекта.

1.3. Структура хранения данных в реляционной базе данных

В любой реляционной базе данных используются одинаковые понятия, разъяснение которых приводится ниже.

Таблица - основная логическая единица реляционной базы данных. С точки зрения программиста или пользователя таблица представляет собой двухмерную структуру, строки которой служат для хранения данных, а столбцы - для определения (описания) структуры данных.

Поле (столбец, колонка) - служат для логического описания данных хранимых в записях, а также определяют их тип - в столбце могут храниться только данные одного и того же типа.

Запись (строка, кортеж) - связанный набор полей различного типа. Является единицей хранения данных в таблице, так как все поля данной записи рассматриваются как логически неразрывное целое.

Ключ (ключевое поле) - какой-либо столбец таблицы, предназначенный для обеспечения уникальности (неповторимости) записей в таблице. В таблице не могут иметься записи, поля которых содержат идентичные данные - как раз для обеспечения данного правила и вводится (определяется) ключ таблицы. Понятно, что значения в данном поле должны быть уникальны для обеспечения уникальности связанных с ними записей.

Связи. Таблицы в базе данных можно логически связывать, причем связь осуществляется по ключевым полям. Ключ таблицы, который используется в другой таблице для обеспечения уникальности данных, называется **внешним ключом.**

1.4. СУБД, поддерживаемые в PHP

В настоящее время PHP поддерживает следующие базы данных:

- Oracle
- Adabas-D
- Sybase
- FilePro
- MSQL
- Velocis
- MySQL
- Informix
- Solid

- DBase
- ODBC
- Unix dbm
- PostgreSQL
- MS SQL

Кроме того, с помощью интерфейса ODBC можно подключаться к другим неподдерживаемым СУБД.

1.5. Архитектура сетевого приложения с поддержкой базы данных

Основными частями или уровнями сетевого приложения с поддержкой базы данных являются:

- **Клиент:** веб-броузер пользователя, апплет Java, приложение Java или, возможно, программа-клиент, зависимая от платформы.
- **Логика приложения:** кодируется в алгоритмах, используемых в сценариях CGI, специальных модулях веб-сервера или даже зависящем от приложения сервере.
- **Соединение с базой данных:** API базы данных или общие протоколы для соединения, такие как ODBC или JDBC.
- **Сервер базы данных:** СУБД, ООСУБД и т. д.

Реализация таких приложений может осуществляться на основе многозвенной (multi-tiered) модели, поскольку один или несколько уровней могут быть объединены вместе. Обычно реализуется трехзвенная система:

Первое звено: веб-клиент (например, броузер пользователя).

Второе звено: веб-сервер, сценарии CGI и API для соединения с базой данных (например, Apache, поддерживающий базы данных SQL, и сценарии PHP).

Третье звено: сервер баз данных (например, сервер MySQL).

2. Сервер MySQL

2.1. Возможности сервера MySQL

Хотя PHP обеспечивает подключение и работу с более чем 10 различными СУБД, наиболее популярной СУБД для работы в связке с PHP остается MySQL. В 90% процентов инсталляций PHP установлен и MySQL, поэтому предлагаю вкратце ознакомиться с данной СУБД.

MySQL является небольшим, компактным и простым в использовании сервером баз данных, идеальным для приложений малого и среднего размера. MySQL доступна на ряде платформ: UNIX. Windows NT и Windows 95/98. Основные преимущества MySQL это:

- Поддержка различных языков - MySQL поддерживает кодировки почти всех европейских языков, а также русского языка. MySQL также может выдавать сообщения об ошибках на этих языках
- Простые API доступа клиентов к базе данных - приложения баз данных MySQL могут быть написаны на таких языках, как C, PERL, PHP
- Хранение больших таблиц - каждая таблица хранится в виде отдельного файла. Таким образом, максимальный размер таблицы равен максимальному размеру файла, допускаемого операционной системой. Каждая же база данных (состоящая из одной и более таблиц) хранится в отдельном каталоге

- Скорость, устойчивость и простота использования.

Следует сказать, что некоторые функции СУБД в MySQL отсутствуют:

- Вложенные команды SELECT - нельзя выполнять SELECT-запрос, имеющий вложенные SELECT-подзапросы. Это некритично, так как большинство таких запросов могут быть переписаны

без применения подзапросов

- Отсутствуют транзакции - транзакция является логической единицей работы, включающей в себя одну или несколько команд SQL, выполняемых одним пользователем. Транзакция завершается, когда она явным образом фиксируется или откатывается этим пользователем. Основное назначение механизма транзакций - цельность операций записи в базу данных, а также предотвращение одновременного изменения данных несколькими пользователями. Так вот, такой механизм в MySQL не предусмотрен — единственная имеющаяся альтернатива - это блокировка таблиц с помощью специальных команд
- Хранимые процедуры и триггеры - процедура представляет собой набор команд SQL, которые скомпилированы и хранятся в самой СУБД, что дает существенный выигрыш в производительности при их вызове пользователем. Триггер - это хранимая процедура, которая вызывается при возникновении некоторого события
- Внешние ключи - это способ логического связывания нескольких таблиц, путем введения одинаковых колонок (полей) данных. В MySQL же, как Вы помните, каждая таблица хранится в отдельном файле
- Представления (виды) - это специальные представления данных, содержащихся в одной или нескольких таблицах (или представлениях). Представление принимает данные, которые возвращаются запросом, и обращается с ними, как с таблицей. Представление можно рассматривать как «сохраненный» запрос или «виртуальную таблицу», причем память для представления не выделяется

Отсутствие этих функций может показаться недостатком, но дело в том, что при проектировании MySQL основной упор делался на скорость работы и простоту проектирования баз данных, поэтому некоторые не очень необходимые функции были просто отброшены разработчиками.

2.2. Установка сервера MySQL

Дистрибутив MySQL поставляется в различном виде, в зависимости от платформы. Дистрибутив для UNIX представляет собой сжатый архив, а для Windows - стандартную установочную программу. Соответственно, для установки MySQL под UNIX необходимо скомпилировать исходные файлы и запустить установку, а в случае Windows - просто осуществить запуск инсталляционной программы.

После установки в указанном Вами каталоге располагаются выполнимые файлы MySQL, базы данных, а также документация по MySQL. Кроме того, в MySQL сразу же создаются две базы данных: **mysql** и **test.** База данных **test** является тестовой и не содержит данных и таблиц, а база данных **mysql** является служебной и содержит таблицы (таб. 1).

После установки Mysql сервера его работоспособность можно проверить с помощью консольной интерактивной утилиты mysql. С помощью последней можно выполнить: подключение к MySQL серверу, получение информации о сервере, выполнение SQL-запросов. Перечень основных команд данной утилиты приводится в таб. 2.

Таблица	
columns_priv	Содержит разрешения на доступ и выполнение операций над колонкам таблиц баз данных MySQL
db	Содержит разрешения на доступ и выполнение операций над
func	Содержит функции MySQL, определенные пользователями
host	Содержит перечень узлов, а также их разрешений на доступ к
tables priv	Содержит разрешения на доступ и выполнение операций над таблицами баз данных MySQL
user	Содержит перечень пользователей и их глобальные разрешения

Таблица 1

Команда	Сокр. запись	Назначение
help	**\h**	Вызов краткой справки по командам утилиты
?	**\?**	Синоним **help**
clear	**\c**	Команда очистки
connect	**\r**	Установка (повторная) соединения с сервером. В качестве произвольных аргументов можно указать узел и базу данных
ego	**\G**	Отправка команды (запроса) на MySQL сервер. Отображение результатов по вертикали
exit	**\q**	Выход из утилиты. Можно также использовать команду **quit**
go	**\R**	Отправка команды (запроса) на MySQL сервер
notee	**\t**	Отключение вывода во внешний файл.
print	**\P**	Вывод текущей (выполняющейся) команды.
r.. hash	**\#**	Перестройка (внутреннего) хэша выполнения
source	**\.**	Выполнение SQL скрипта. Принимает имя файла в качестве аргумента
status	**\s**	Получение статусной информации от сервера
tee	**\T**	Установка внешнего файла. Все полученные результаты будут выводиться в указанный файл.
use	**\u**	Использование другой базы данных. Принимает имя базы данных в качестве аргумента

Таблица 2

имеется также консольная утилита mysqladmin, которая не является интерактивной и ориентирована на автоматизированное выполнение батчей (групп команд) и SQL-скриптов.

3. Обзор синтаксиса языка SQL

В последующих разделах рассматриваются наиболее употребительные команды SQL для модификации структуры базы данных и для доступа и обновления данных в реляционной базе данных. Производители баз данных расширяют SQL в своих продуктах, поэтому приложение SQL, написанное для одной базы данных, может не сразу начать работать с другой базой данных. За полным описанием всех команд SQL следует обратиться к документации по SQL той реляционной базы данных, которую вы используете. Ниже рассматривается вариант SQL. специфический для MySQL.

3.1. Команды определения данных

Команды определения данных, или запросы, являются предложениями языка SQL, модифицирующими схему базы данных путем создания или изменения объектов в текущей базе данных.

CREATE

Используется для создания новой базы данных или таблицы в существующей базе данных. Синтаксис создания новой таблицы сложнее, поскольку требует включения описаний полей. Синтаксис создания базы данных очень прост:

CREATE DATABASE имя_базы_данных

Ниже описаны синтаксические переменные для команды CREATE TABLE:

CREATE TABLE имя__таблицы

(список_определения)

В списке определения указывается (через запятую) перечень полей создаваемой таблицы в виде:

имя колонки тип

Где тип может иметь одно из следующих значений:

TYNYINT	DATE
SMALLINT	TIME
MEDIUMINT	TIME STAMP
INT INTEGE	DATETIME TINYBLOB
BIGINT	BLOB
REAL	MEDIUMBLOB
DOUBLE	LONGBLOB
FLOAT	TINYTEXT
DECIMAL	TEXT
NUMERIC	LONGTEXT
CHAR	ENUM
VARCHAR	SET

DROP

Как и в случае ключевого слова CREATE, существует несколько команд DROP DATABASE. **DROP** TABLE, **DROP** INDEX и **DROP** FUNCTION. Ниже подробно рассматриваются команды **DROP DATABASE** и **DROP** TABLE.

Для уничтожения базы данных (будьте осторожны, эта команда полностью удаляет все данные во всех таблицах базы данных и саму базу данных) используется команда:

DROP DATABASE [IF EXISTS] имя базы данных

Приложение SQL, написанное для одной базы данных, может не сразу начать работать с другой базой данных.

Для удаления таблицы используется аналогичный синтаксис: **DROP TABLE [IP EXISTS] имя_таблицы**

Одной командой можно удалить несколько таблиц. Например, для удаления созданных ранее таблиц **document** и **author** можно выполнить:

DROP TABLE document, author

3.2. Команды обработки данных

Эти команды используются для изменения данных, хранящихся в базе данных.

INSERT

Команда **INSERT** используется для ввода строк в таблицу базы данных. Общий синтаксис команды следующий:

```
INSERT [INTO] имя_таблицы
(список_колонок) VALUES
(список_Значений)
```

В списке колонок перечисляются изменяемые колонки, а в списке значений - присваиваемые им значения. Ключевое слово **INTO** указывать необязательно. Пример команды **INSERT:**

```
INSERT INTO author (id,
fullname, email)
   VALUES ('jOOl', 'John
   Writer',
   'jw@somewhere.nice.com')
```

REPLACE

Синтаксис команды **REPLACE,** которая является специфической для MySQL, аналогичен синтаксису команды **INSERT,** и выполняется она сходным образом. Разница заключается в том, что если в таблице существует старая запись с таким же значением в уникальном индексе, как и в добавляемой, он заменяется новой записью.

```
REPLACE [INTO] имя_таблицы
(список_колонок) VALUES
(список_значений)
```

UPDATE

При изменении одной или более колонок в существующей записи используется команда **UPDATE.** Ее синтаксис следующий:

```
UPDATE имя_таблицы SET
имя_колонки_1=значение_1,
имя_колонки_2=значение_2 ,...
[WHERE условие_поиска]
```

Если условие поиска **WHERE** отсутствует, указанные значения записываются во все названные поля таблицы.

Ниже приведен пример использования команды **UPDATE:**

```
UPDATE documents SET
title='Table of Contents',
Comment='Fixed typo in the
title' WHERE id=231
```

Эта команда изменяет название документа с id, равным **231,** и добавляет комментарий, объясняющий причину изменения.

DELETE

Эта команда противоположна командам **INSERT** и **REPLACE.** Команда **DELETE** удаляет из таблицы одну или более записей, удовлетворяющих некоторому условию. Если условие **WHERE** отсутствует, удаляются все записи в таблице, но не сама таблица, поэтому данную команду следует использовать с осторожностью. Синтаксис команды следующий:

```
DELETE FROM имя_таблицы
[WHERE условие_поиска]
```

ниже приведены примеры использования команды **DELETE.** Удаление всех статей, опубликованных до 1 января 1999 года:

DELETE FROM documents WHERE published < 4990-01-01'

3.3. Команды, используемые для поиска в базе данных

Команду **SELECT** вы будете использовать в своих Web-приложениях при осуществлении поиска в базах данных MySQL. Поэтому данная команда имеет сложный синтаксис:

```
SELECT [DISTINCT]
   выражение_выборки FROM
   имя_таблицы [WHERE
   условие_поиска] [GROUP BY
   список_колонок] [HAVING
   where_definition]
    [ORDER BY имя_колонки |
формула [ASC | DESC]]
[LIMIT [смещение]
количество_столбцов] [PROCEDURE
имя_процедуры]
```

Параметры **обязаны** присутствовать в том порядке, в котором они перечислены выше, поэтому параметр **HAVING** указывается раньше, чем **ORDER BY** или **LIMIT,** или **PROCEDURE.**

SELECT используется не только для извлечения строк из базы данных, но для получения значений математических выражений:

```
SELECT SQRT((144 % 5) - 1)
```

Ниже приводится краткое описание некоторых параметров:

DISTINCT

Используется для исключения повторяющихся строк в возвращаемых результатах.

- **FROM**
  Используется для указания таблиц, участвующих в процессе выборки. Таблицам MOI\I присваиваются псевдонимы, что удобно, например, когда при извлечении данных нужно сравнивать поля из разных таблиц с одинаковыми именами.

- **WHERE**
  Условие отбора, которое может содержать операторы сравнения, математические функции и логические выражения. Например, следующая команда извлекает имя автора и название созданного им документа для всех авторов с фамилией «Smith»:

```
SELECT DISTINCT
  document.title,
  author.fullname FROM
  document, author WHERE
  author.fullname LIKE "%
  Smith"
```

LIKE

Функция поиска по шаблону, которая возвращает **true,** если выражение соответствует шаблону, и **false** в противном случае. В шаблоне можно использовать символы-заполнители: '%', обозначающий соответствие нулю или более символам, и '_', соответствующий ровно одному символу.

GROUP BY

Этот параметр позволяет группировать результаты по некоторому полю. Обычно он используется только с агрегатными функциями в параметрах выборки. Например, для получения среднего возраста **CLERKS** по различным отделам организации используется команда:

```
SELECT AVG(age), deptno FROM staff
WHERE job='CLERK' GROUP BY deptno
```

- **HAVING**
  Условия, используемые в этом предложении, аналогичны условиям в предложении **WHERE.** Различие заключается в том, что **HAVING** определяет сгруппированную таблицу, образуемую исключением групп, не отвечающих условию, из результата ранее заданного предложения GROUP BY. Например, для того, чтобы в предыдущем примере исключить получение результатов для тех отделов, в которых средний возраст служащих менее 30 лет, используем команду:

```
SELECT AVG(age) FROM staff
WHERE job='CLERK' GROUP BY
depno HAVING AVG(age)>=30
```

- **ORDER BY**
  Это предложение упорядочивает результаты по некоторому полю. Можно задать возрастающий порядок **ASC** (по умолчанию) или убывающий **DESC.** Например, для получения документов с «PHP» в заголовке и в порядке убывания даты публикации нужно выполнить команду:

```
SELECT id, title,
published FROM document
WHERE title LIKE л%PHP%'
ORDER BY published
```

- **LIMIT**
  Используется для ограничения количества строк, возвращаемых командой **SELECT.** В том предложении могут быть один или два числовых аргумента.

Если используются два аргумента, то первый из них указывает смещение от первой строки, а второй - количество возвращаемых строк (от смещения). Смещение отсчитывается от нуля, т. е. первая строка результата имеет смещение, равное нулю. Например, для возврата первых десяти названий документов:

```
SELECT titles, published
FROM document LIMIT 10
```

Для возврата следующих 10 строк (строки 11-20):

```
SELECT titles, published FROM
document LIMIT 10, 10
```

- **PROCEDURE**
  В MySQL можно определять процедуры на C++ для доступа и модификации данных запроса перед возвращением их клиенту. Дополнительные сведения о создании процедур можно найти в руководстве по MySQL.

3.4. Использование утилит выполнения запросов

Для выполнения запросов к MySQL серверу можно использовать различные утилиты.

С помощью консольной интерактивной утилиты **mysql** можно выполнить: подключение к MySQL серверу, получение информации о сервере, выполнение SQL-запросов. Для того, чтобы выполнить из этой утилиты запрос, достаточно набрать в консоли SQL-запрос и завершить его символом (;) или командой go (\g). Например:

```
mysql> CREATE DATABASE
accounts \g
```

Имеется также консольная утилита **mysqladmin,** которая не является интерактивной и ориентирована на автоматизированное выполнение батчей (групп команд) и SQL-скриптов.

Но выполнение запросов из консоли не всегда удобно. Для облегчения работы и удобства существуют визуальные и Web утилиты, которые позволяют получать визуальное представление структуры базы данных, а также автоматически генерировать SQL-запросы на основе введенных (выбранных) пользователем параметров.

Наиболее распространенной Windows-утилитой для работы с MySQL-сервера является DB Tools, которая позволяет осуществлять визуальное администрирование сервера MySQL, а также осуществлять DB-программирование последнего.

Но наиболее удобным средством администрирования MySQL-сервера является свободно распространяемое скриптовое приложение phpMyAdmin. Оно написано на PHP и выполняется под Apache-сервером. С его помощью через браузер можно:

- Создавать и удалять базы данных
- Создавать, копировать, удалять и изменять таблицы
- Удалять, редактировать и добавлять поля
- Выполнять любой SQL-запрос, включая групповые запросы (батчи)
- Управлять ключами полей
- Загружать текстовые файлы в таблицы
- Создавать и считывать дампы (dumps) таблиц
- Экспортировать и импортировать данные в CSV-формате
- Администрировать несколько серверов и выделенные базы данных
- Проверять ссылочную целостность
- Взаимодействовать с пользователем на более, чем 38 языках

Для установки достаточно скопировать скрипты phpMyAdmin в папку, расположенную в корневом каталоге Apache. После чего необходимо осуществить настройку путем редактирования файла confis.inc.Dhn. Наиболее полезные ключи данного файла приводятся в таблице 3:

Ключ	Тип	Назначение
ScfgPmaAbsoluteUri	**string**	Должен содержать абсолютный URL, указывающий на **phpMyAdmin**
ScfgServers	**array**	Массив настроек узлов MySQL. **phpMyAdmin** позволяет осуществлять подключение к нескольким серверам MySQL. Соответственно, настройки каждого подключения содержатся в ассоциативном массиве, и ссылка на каждый элемент массива осуществляется следующим образом: cfgServers[l]['host'], cfgServers[I]['host'] …
$cfgServers[n] ['host']	**string**	Имя n-го MySQL сервера. Например, "localhost"
ScfgServers [n] ['port']	**string**	Порт для подключения к службе MySQL на n-ом сервере
$cfgServers[n] ['connect type']	**string**	Тип соединения. Возможны значения "top" и "socket"
$cfgServers[n]['auth_type']	**string**	Тип аутентификации при подключении. Возможны значения "http", "cookie" или "config"
$cfgServers[n]['user'|	**string**	Имя пользователя, которое используется для подключения к серверу MySQL. В случае типа аутентификации "http" или "cookie" должно оставаться пустым
ScfgServers [n] ['password']	**string**	Пароль, который используется для подключения к серверу MySQL. В случае типа аутентификации "http" или "cookie" должен оставаться пустым
ScfgServerDefault	**integer**	Номер сервера (подключения), с которым нужно автоматически соединяться при запуске **phpMyAdmin**

Таблица 3

Более подробную информацию о настройке **phpMyAdmin** и назначении отдельных ключей можно получить из файла Documentation.html, поставляющегося вместе со скриптами **phpMyAdmin.**

3. Создание Web-интерфейсов БД на PHP

3.1. Общие принципы создания интерфейсов БД

Простота подключения к базам данных является одной из наиболее сильных сторон PHP. Благодаря этому, в PHP можно быстро строить интерфейсы для баз данных или приложения, работающие с базами данных.

Работа с базой данных состоит из трех последовательно выполняемых этапов. Рекомендации по реализации каждого из этих этапов в PHP приводятся ниже:

- Подключение к базе данных - рекомендуется выносить блок функций, отвечающих за подключение к базе данных в отдельный файл, который затем включать в нужных местах с помощью директив включения **include()** и **require().**
- Выполнение запросов и обработка полученных результатов - в большинстве приложений SQL-запросы динамически формируются в PHP-коде, для чего чаще всего используется операция конкатенации строк (.).

- Хотя бывают случаи, когда используются неизменяемые SQL-запросы.

Отключение от базы данных в PHP обычно не требуется, так как интерпретатор PHP может выполнить это действие автоматически при прекращении выполнения скрипта.

3.2. Функции PHP для работы с MySQL

- **mysql_connect**

Создает соединение с сервером MySQL.

```
int mysql_connect (string [hostname [-.port} [ : /path_to_socket]] , string [username], string [password]);
```

Аргументы этой функции представлены в таблице 4 ниже; все они являются необязательными:

Соединение (между клиентом - программой PHP и сервером MySQL) закрывается при вызове mysql_close или выходе из сценария PHP.

- **Mysql_close**

Завершает соединение с сервером MySQL и является необязательной.

```
int mysql_close(int [link-identifier] ) ;
```

- *link identifier*

Ссылка на закрываемое соединение.
Значение по умолчанию: Идентификатор ссылки для соединения

Функция mysql_close возвращает true в случае успеха и false при ошибке.

Параметр	Описание	Значение по
hostname	Имя узла, на котором выполняется сервер базы данных. Его не нужно указывать, если база данных и вебсервер работают на одной машине	"localhost"
.port	Порт, на котором сервер баз данных слушает запросы. Этот параметр необходимо задавать только в случае, если в настройках используется порт, отличный от установленного в MySQL по умолчанию	":3306"
./path to socket	Сокет Unix, на котором сервер слушает запросы	":/tmp/mysql.sock"
username	Имя пользователя, которому разрешено соединение с базой данных	Пользователь, владеющий процессом веб-сервера
password	Пароль пользователя; если не задан, считается пустой строкой	

Таблица 4

Функция возвращает идентификатор связи - положительное число, указывающее на соединение - в случае успеха и false при неудаче. Этот идентификатор связи будет использоваться при вызове всех функций, посылающих запросы серверу MySQL.

- **mysql_select_db**

Выбирает базу данных в качестве активной.

```
int mysql_select_db(string database name, int [link identifier]);
```

- *database name*

Имя базы данных, которая должна стать активной

- *link-identifier*

Ссылка на соединение, по которому запрос будет послан серверу базы данных. *Значение по умолчанию:* Идентификатор ссылки для соединения, открытого последним. Если открытых соединений нет, функция пытается открыть новое соединение, используя ***mysq_connect*** *с* параметрами, установленными по умолчанию.

параметр database ___ name является необходимым, а параметр linkidentfier - необязательным.
Функция возвращает true в случае успеха и false при неудаче.
Все команды SQL, передаваемые серверу MySQL, будут выполняться с активной базой данных.

- **mysql_query**

Посылает команду SQL серверу MySQL для исполнения:

```
int mysql_query(string query, int [link identifier]);
```

- *query*

Команда SQL, которую нужно послать серверу MySQL

- *link-identifier*

Ссылка на соединение, по которому запрос будет послан серверу базы данных. *Значение по умолчанию:* Идентификатор ссылки для соединения, открытого последним. Если открытых соединений нет, функция пытается открыть новое соединение, используя ***mysql_connect*** *с* параметрами, установленными по умолчанию.

Параметр **query** является необходимым, а параметр **linkidentifier** - необязательным.

Функция возвращает идентификатор результата (положительное целое) в случае успеха и false при неудаче. Идентификатор результата содержит результат выполнения команды SQL на сервере MySQL.

- **mysql_list_dbs**

Перечисляет базы данных, имеющиеся на сервере MySQL.

```
int mysql_list_dbs(int [link identifier]) ;
```

- *link-identifier*

Ссылка на соединение, по которому запрос будет послан серверу базы данных. *Значение по умолчанию:* Идентификатор ссылки для соединения, открытого последним. Если открытых соединений нет, функция пытается открыть новое соединение, используя ***mysql_connect*** *с* параметрами, установленными по умолчанию.

Параметр linkidentifier является необязательным. В случае успеха функция возвращает идентификатор результата, при неудаче возвращается false. Для прохода по идентификатору результата с целью получения списка баз данных нужно использовать функцию mysql_tablename().

- **mysql_list_tables**

Перечисляет все таблицы в базе данных MySQL.

```
int mysql_list_tables(string database, int [link_identifier]);
```

- *database name*

Имя базы данных, которая должна стать активной.

- *link-identifier*

Ссылка на соединение, по которому запрос будет послан серверу базы данных.

Значение по умолчанию: Идентификатор ссылки для соединения, открытого последним. Если открытых соединений нет, функция пытается открыть новое соединение, используя ***mysql_connect*** *с* параметрами, установленными по умолчанию.

Параметр database является необходимым, а параметр linkidentifier - необязательным. В случае успеха функция возвращает идентификатор результата, при ошибке возвращается false. Для прохода по идентификатору результата с целью получения списка таблиц нужно использовать функцию mysql_tablename().

- **mysql_num_rows**
  Возвращает количество строк в идентификаторе результата (который содержит результат выполнения команды SQL).

  ```
  int mysql_num_rows(int result_identifier);
  ```

  - *result_identifier*
    Идентификатор результата, возвращаемый mysql_ db_query, mysql_query, mysql_list_tables, mysql_list_dbs

Параметр result identifier является обязательным. Эта функция используется, если выполненный запрос соответствовал команде SELECT.

- **mysql_table_name**
  Получить имя таблицы/базы данных по идентификатору результата:

  - *result_identifier*
    Идентификатор результата, возвращаемый mysql_db_query, mysql_query, mysql_list_ tables, mysql_list_dbs.
  - *i*
    Индекс в *result_ identifier.*

  Оба параметра: resultidentifier и i - являются обязательными.

  Функция возвращает имя таблицы/базы данных с индексом i в идентификаторе результата.

- **mysql_num_fields**
  Получает количество полей в результирующем наборе. int mysql_num_fields(int resul_identifier);

  - *result_identifier*
    Идентификатор результата, mysql_db_query, mysql_query, mysql_list_ tables, mysql_list_dbs возвращаемый mysql_query, mysql_list_dbs

Параметр result_identifier является обязательным. Функция возвращает количество полей в resultidentifier.

- **mysql_field_name**
  Извлекает имя поля из базы данных.

  ```
  string mysql_field_name (int result_±dent±f±er, int field_lndex);
  ```

  - *result_identifier*
    Идентификатор результата, возвращаемый mysql_db_query, mysql_query, mysql_list_ tables, mysql_list_dbs

  - *field_index*
    Индекс поля в идентификаторе результата

Параметры result_identifier и fieldindex являются обязательными. Функция возвращает имя поля с индексом fieldindex в resultidentifier.

- **mysql_field_type**
  Возвращает **тип** данных указанного **поля**

  ```
  string mysql_field_type (int result_ident±fier, int field_index);
  ```

  - *result_identifier*
    Идентификатор результата, возвращаемый Mysql_db_query,

mysql_query, mysql_list_ tables, mysql_list_dbs

- *field_index*
  Индекс поля в идентификаторе результата

Параметры result_identifier и field_index являются обязательными. Функция возвращает тип поля с индексом fieldindex в resultidentifier.

- **mysql_fetch_row**
  Извлекает очередную строку из идентификатора результата в виде пронумерованного массива.

```
array mysql_fetch_row(int result_identlfier);
```

  - *result_identifier*
    Идентификатор результата, возвращаемый mysql_db_query, mysql_query, mysql_list_tables, mysql_list_dbs

Параметр result_identifier является обязательным. Функция возвращает массив (соответствующий текущей строке) или false, если строк больше нет. mysql_fetch_row() инкрементирует внутреннее поле указателя строки в result identifier, поэтому каждый следующий вызов mysql_fetch_row() возвращает очередную строку результата. mysqldataseek

- **mysql_fetch_array**
  Осуществляет выборку строки в виде ассоциативного массива.

```
array mysql_fetch_array(int result_identifier, INT [result_type]);
```

  - *result_identifier*
    Идентификатор результата, возвращаемый mysql_db_query, mysql_query, mysql_list_tables, mysql_list_dbs

  - *result_type*
    Константа, обозначающая тип (или типы) возвращаемого массива

Параметр **result_identifier** является обязательным. Необязательный второй аргумент **result_typc** может иметь следующие значения:

- MYSQLJMUM: в возвращаемом массиве будут содержаться только числовые индексы (аналогично **mysql_fetch_row** ());
- MYSQL ASSOC: в возвращаемом массиве будут содержаться только ассоциативные индексы;
- MYSQL BOTH: в возвращаемом массиве будут содержаться как числовые, так и ассоциативные индексы.

Если второй аргумент **result_type** не задан, то в качестве его значения предполагается **MYSQL BOTH.**

Каждый последующий вызов **mysql_fetch_array()** возвращает массив, соответствующий очередной строке, или **false,** если строк больше нет. Это расширенный вариант **mysql_fetch_row(),** которая возвращает массив только с числовыми индексами.

Напротив, **mysql_fetch_array()** возвращает результаты и как ассоциативный массив с именами полей в качестве ключей; производительность при этом не страдает. Ограничением ассоциативных массивов является то, что если имена некоторых полей дублируются, то приоритет имеет последнее из них, и для извлечения остальных полей с тем же именем приходится использовать числовые индексы или псевдонимы полей результата.

- **mysql_result**
  Получить данные из идентификатора результата.

```
mixed mysql_result(int result_identifier, int row, mixed [field]);
```

- *result_identifier*

Идентификатор результата, возвращаемый mysql_db_query, mysql_query, mysql_list_tables, mysql_list_dbs

- *row*

Строка, из которой должны быть получены данные

- *field*

Поле в строке, из которого должны быть получены данные

Параметры result_identifier и row являются обязательными, а field - необязательным. Функция возвращает содержимое строки row и колонки field из result_identifier. Необязательный аргумент field может быть смещением колонки, именем колонки или table.column_name. Если аргумент field не указан, возвращается очередное поле строки.

Редакция

Спасибо участникам проекта!

Материалы подготовили:

- **Еремеев Михаил,** хобби – PHP, программирование, радиоэлектроника. Статус - PHP-Specialist - www.specialist.ru/?public=154398. Его проект: http://phpru.com, это молодой проект, посвящённый программированию на php. Ищем друзей для ведения форума, администрирования и.т.д.
- **Сергей [msa78@mail.ru]**.
- **Maxim Matyukhin.**
- **Владимир Жульев.**
- **Олищук Андрей [nw; project@yugovostok.ru]** также координатор этого проекта.

Авторы, чьи статьи мы взяли с сайтов

- **Александр Шиляев,** его статья взята с сайта «PHP в деталях»
- **Антон Довгаль [Tony2001],** его перевод взят с сайта tony2001.phpclub.net

Помощь и ценные идеи

- **Войцеховский Александр [young]**
- **Елена Тесля [Lenka]**
- **Maxim Matyukhin**
- **http://www.phpclub.ru [администрацию и сообщество сайта PHP разработчиков]**

Дизайн обложки

- студия **grafit44** - web-дизайн, компьютерная графика http://grafit44.by.ru , grafit44@bk.ru