ФУНКЦИИ В PHP

Рассмотрим общие концепции функционального программирования – важного принципа в области разработки приложений. При помощи функций можно создавать компоненты многократного использования, которые легко изменяются при необходимости и оказываются особенно удобными при разработке Web-приложений, не содержащих серьезных различий на концептуальном и практическом уровне. Функциональное программирование помогает создавать более короткие и удобочитаемые программы.

Данная глава посвящена функциям PHP, их определению и применению на практике. Хотя основное внимание в ней уделяется определению и вызову пользовательских функций, необходимо помнить и о том, что в PHP существуют сотни стандартных функций. Стандартные функции работают точно так же, как пользовательские, и обеспечивают заметную экономию времени при создании новых приложений. Обновленный список стандартных функций PHP можно найти по адресу http://www.php.net .

Функцией называется фрагмент программного кода, обладающий уникальным именем и предназначенный для решения конкретной задачи.

Функция вызывается по имени в разных точках программы, что позволяет многократно выполнять фрагмент с указанным именем. Преимущество такого решения заключается в том, что блок кода пишется всего один раз, а затем легко модифицируется по мере необходимости.

Определение и вызов функций

Определить новую функцию в PHP несложно. Функции могут создаваться в любой точке программ PHP, однако по соображениям структурной организации кода удобнее разместить все функции, используемые сценарием, в самом начале сценарного файла. Существует и другой способ, заметно повышающий эффективность программирования и способствующий многократному использованию кода, -- выделение функций в отдельный файл (называемый *библиотекой).*Библиотеки удобны тем, что их функции можно использовать в разных приложениях, не создавая лишних копий и не рискуя допустить ошибки в процессе копирования. Эта тема подробно рассматривается в разделе «Построение библиотек функций» ближе к концу главы.

Определение функции обычно состоит из трех частей:

- имени функции;
- круглых скобок, в которых перечисляются необязательные входные параметры, разделенные запятыми;
- тела функции, заключенного в фигурные скобки.

Обобщенный синтаксис функций PHP выглядит так:

```
function имя_функции ([$параметр1. $параметр2, .... $параметрn]) {
тело функции
}
```

Имя функции должно подчиняться определенным условиям. После имени функции следуют обязательные круглые скобки, в которые заключается необязательный список входных параметров ($параметр1, $параметр2, ... $параметрn). Вследствие относительно свободных принципов определения переменных в PHP указывать тип входных параметров не нужно. Хотя такой подход имеет свои преимущества, следует помнить, что механизм PHP не проверяет аргументы на соответствие тем типам, которые должны обрабатываться функцией. Случайные ошибки в использовании входных параметров могут привести к неожиданным последствиям (чтобы убедиться в том, что параметр относится к нужному типу, можно проверить его стандартной функцией gettype( )). После закрывающей круглой скобки следуют фигурные скобки, в которые заключается программный код, ассоциируемый с именем функции.

Рассмотрим простой пример использования функции. Предположим, вы хотите создать функцию для вывода лицензионной информации на Web-странице:

```
function display_copyright() {
print "Copyright &copy; 2021 PHP-Powered Recipes. All Rights Reserved.";
}
```

Если ваш Web-сайт состоит из нескольких страниц, достаточно вызвать эту функцию в конце каждой страницы -- и вам не придется заново переписывать один и тот же текст. А когда наступит 2022 год, одно простое изменение текста, выводимого этой функцией, приведет к автоматическому обновлению всех страниц. Если бы не преимущества функционального программирования, вам пришлось бы вручную редактировать все страницы, на которых выводится лицензионная информация.

Рассмотрим разновидность функции display_copyright(), которой при вызове передается параметр. Предположим, вы отвечаете за администрирование нескольких web-сайтов, каждому из которых присвоено отдельное имя. На каждом сайте имеется собственный административный сценарий с несколькими переменными, относящимися к этому сайту; к их числу принадлежит переменная $site_name с именем сайта.

В этом случае функцию display_copyright() можно записать следующим образом:

```
function display_copyright($site_name) {
print "Copyright &copy; 2001 $site_name. All Rights Reserved.";
}
```

Переменная $site_name, значение которой присваивается за пределами display_copy-right(), передается функции в качестве параметра. Переданное значение можно использовать и модифицировать в любом месте функции, однако любые изменения будут действовать лишь внутри этой функции. Впрочем, специальные ключевые слова позволяют сделать так, чтобы изменения параметров распространялись и за пределы display_copyright(). Эти ключевые слова были представлены в главе 2, в общем обзоре области видимости переменных и ее отношения к функциям.

Вложенные функции

Функции можно вызывать внутри других функций -- по аналогии с тем, как одна управляющая конструкция (if, while, for и т. д.) может находиться внутри другой. Такая возможность удобна в любых программах, и в больших, и в малых, поскольку она увеличивает степень модульности приложения и упрощает сопровождение программы.

В примере, описанном выше, можно полностью избавиться от необходимости модификации даты. Достаточно включить в display_copyright() вызов стандартной функции PHP date( ):

```
function display_copyright($site_name) {
print "Copyright &copy". date("Y"). "$site_name. All Rights Reserved.";
}
```

Параметр Y функции date( ) указывает, что возвращаемое значение представляет собой текущий год, отформатированный в виде четырех цифр. Если системная дата установлена правильно, PHP при каждом выполнении сценария будет выводить год. Функция PHP date( ) отличается исключительной гибкостью и поддерживает 25 разных флагов форматирования даты и времени.

Также допускается объявление функций внутри других функций. Тем не менее, вложенное объявление еще не делает функцию «защищенной», то есть не ограничивает возможность ее вызова той функцией, в которой она была объявлена. Более того, вложенная функция не наследует параметров родительской функции; параметры должны передаваться ей точно так же, как и любой другой функции. Впрочем, вложенные объявления функций все равно могут использоваться из соображений удобства сопровождения и наглядности. Пример вложенного объявления приведен в листинге 1:

Листинг 1. *Эффективное использование вложенных функций*

```
function display_footer($site_name) {
function display_copyright($site_name) {
print "Copyright &copy". date("Y").
$site_name. All Rights Reserved.";
print "<center>
<a href = \"\">home</a> | <a href = \"\">recipes</a> | <a href = \"\">events</a><br>
<a href = \"\">tutorials</a> | <a href = \"\">about</a> I <a href = \"\">contact us</a><br>";
display_copyright($site_name);
print "</center>";
```

```
}
$site_name = "PHP Recipes":
display_footer($site_name);
display_copyhght($site_name);
```
Сценарий выводит следующий результат:
home | recipes | events
tutorials | about | contact us
Copyright © 2001 PHP Recipes. All Rights Reserved

Функцию display_copyright( ) можно вызвать и за пределами display_footer( ) по аналогии стем, как функция display_footer( ) использовалась в предыдущем примере. Концепция защищенных функций в PHP не поддерживается.

Хотя вложенные функции не защищены от вызова из других точек сценария, они не могут вызываться до вызова своей родительской функции. При попытке вызвать вложенную функцию раньше вызова родительской функции выводится сообщение об ошибке.

Возврат значений функцией

По завершении работы функции часто бывает полезно вернуть некоторое значение, для чего результат вызова функции обычно присваивается некоторой переменной. Функции могут возвращать значения любых типов, в том числе массивы и списки. Пример приведен в листинге 2, где функция calculate_cost( ) вычисляет налог с заданной суммы и возвращает общую сумму вместе с налогом. Прежде чем переходить к рассмотрению листинга, просмотрите краткое описание алгоритма на псевдокоде:

Перед вызовом функции задать значения переменных: $price (цена товара) и $tax (налоговая ставка).

Объявить функцию calculate_cost( ). При вызове функция получает два параметра: налоговую ставку и цену товара.

Вычислить цену с учетом налога и вернуть ее командой return.

Вызвать calculate_cost() и присвоить значение, возвращенное функцией, переменной $total_cost.

Вывести соответствующее сообщение.

Листинг 2. *Создание функции для вычисления налога*

```
$price = 24.99; $tax = .06;
function calculate_cost($tax, $price) {
$sales_tax = $tax;
return $price + ($price * $sales_tax);
}
// Обратите внимание на возврат значения функцией calculate_cost(). $total_cost = calculate_cost ($tax. $price);
// Округлить цену до двух десятичных цифр.
$total_cost = round($total_cost. 2);
print "Total cost: $".$total_cost;
// $total cost = 26.49
```

Функции, не возвращающие значений, также называются процедурами.

Существует и другой способ использования возвращаемых значений, при котором вызов функции включается прямо в условную/циклическую команду. В следующей программе (листинг 3) сумма счета пользователя сравнивается с предельным размером кредита. Алгоритм на псевдокоде выглядит так:

Объявить функцию check_limit( ), которая при вызове получает два параметра. Первый параметр, $total_cost, определяет общую сумму счета, накопленную пользователем до настоящего момента. Второй параметр, $credit_limit, определяет максимальную сумму, которую может потратить пользователь.

Если накопленная сумма счета превышает предельный размер кредита, функция возвращает ложное значение (0).

Если условие команды i f оказывается ложным, работа функции еще не завершена. В этом случае общая сумма не превышает предельного размера кредита, поэтому функция должна вернуть логическую истину.

Вызвать функцию check_limit( ) в условии команды if. Проверить, какое значение было возвращено при вызове -- истинное или ложное. В зависимости от результата проверки выполняется то или иное действие.

Если при вызове check_limit( ) было получено значение TRUE, мы предлагаем пользователю продолжить закупку. В противном случае пользователь информируется о превышении кредита.

Листинг 3. *Сравнение текущей суммы счета пользователя с предельным размером кредита*

```
$cost = 1456.22;
$limit = 1000.00;
function check_limit($total_cost. $credit_limit)
if ($total_cost > $credit_limit) :
return 0;
endif;
return 1;
}
if (check_limit($cost. $limit)) :
// Продолжить закупки
print "Keep shopping!";
else :
print "Please lower your total bill to less than $".$limit."!";
endif;
```

При выполнении листинга 3 будет выведено сообщение об ошибке, поскольку значение $cost превышает $limit.

Функция также может возвращать сразу несколько значений при помощи списка. Продолжая кулинарную тему, давайте напишем функцию, которая бы возвращала три лучших года для указанного сорта вина. Функция приведена в листинге 4, но сначала прочитайте алгоритм на псевдокоде:

Объявить функцию best_years( ), вызываемую с одним параметром. Параметр $label определяет сорт вина, для которого пользователь хотел бы узнать три рекомендуемых года.

Объявить два массива, $merlot и $zinfandel. В каждом массиве хранится три рекомендуемых года для соответствующего сорта вина.

Написать команду return, которая бы использовала особые возможности переменных. Выражение $$label сначала интерпретирует переменную $label, а затем интерпретирует полученное значение как имя другой переменной. В настоящем примере массив merlot возвращается в виде списка, и каждый возвращаемый год занимает свою позицию в списке, для которого вызывалась функция.

Вывести сообщение с информацией о рекомендуемых годах.

Листинг 4. *Возвращение функцией нескольких величин*

```
// Сорт вина, для которого выводятся лучшие годы
$label = "merlot";
// Функция использует массивы и "переменную в переменной"
// для возвращения нескольких значений.
function best_years($label) {
$merlot = array("1987", "1983", "1977");
$zinfandel = array("1992", "1990", "1989");
return $$label;
}
// Функция list( ) используется получения возвращаемых значений.
list ($yr_one, $yr_two. $yr_three) = best_years($label);
print "$label had three particularly remarkable years: $yr_one. $yr_two, and $yr_three.";
```

Программа выводит следующий результат:

```
merlot has three particularly remarkable years: 1987, 1983 and 1977.
```

Рекурсивные функции

Ситуация, при которой функция многократно вызывает сама себя, пока не будет выполнено некоторое условие, открывает замечательные возможности.

При правильном использовании *рекурсивные* функции уменьшают объем программы и делают ее более выразительной. Рекурсивные функции особенно часто используются при выполнении повторяющихся действий -- например, при поиске в файлах/массивах и построении графических изображений (например, фракталов). Классическим примером рекурсивных функций, встречающимся во многих курсах программирования, является суммирование чисел от 1 до N. Программа, приведенная в листинге 5, суммирует все целые числа от 1 до 10.

Листинг 5. *Использование рекурсивной функции для суммирования последовательных целых чисел*

```
function summation ($count) {
if ($count != 0) :
return $count + summation($count-1);
endif;
}
$sum = summation(10);
print "Summation = $sum";
```

В результате выполнения листинга 5 будет выведен следующий результат:

```
Summation = 55
```

Если функция вызывается достаточно часто, рекурсия делает программу более эффективной. Тем не менее, при использовании рекурсии необходима осторожность, поскольку ошибки могут привести к зацикливанию программы.

Функции-переменные

Одной из интересных возможностей PHP являются *функции-переменные* (variable functions), то есть динамические вызовы функций, имена которых определяются во время выполнения программы. Хотя в большинстве web-приложений можно обойтись и без функций-переменных, они значительно сокращают объем и сложность программного кода, а также часто снимают необходимость в условных командах if.

Вызов функции-переменной представляет собой имя переменной, за которым следует пара круглых скобок. В круглых скобках могут перечисляться параметры (однако присутствие параметров не обязательно). Обобщенный синтаксис функции-переменной: $имя_функции( );

Следующая программа (листинг 6) демонстрирует эту непривычную, но полезную возможность. Допустим, программа выводит разную информацию в зависимости от языка, выбранного пользователем. В нашем примере для простоты используются приветственные сообщения для англо- и италоязычных пользователей. Алгоритм на псевдокоде:

- создать сообщение для итальянского языка в функции с именем italian.
- создать сообщение для английского языка в функции с именем english.
- передать информацию о выбранном языке в сценарий, присвоив значение переменной $language.

Переменная $language используется для выполнения функции-переменной (в приведенном примере -- italian()).

Листинг 6. *Выбор функции в зависимости от пользовательского ввода*

```
// Приветствие на итальянском языке, function italian( ) {
" print "Benvenuti al PHP Recipes.";
}
// Приветствие на английском языке
function english( ) {
print "Welcome to PHP Recipes.";
}
// Выбрать итальянский язык
$language = "italian":
// Выполнить функцию-переменную
$language( );
```

Листинг 6 демонстрирует интересную концепцию функций-переменных и наглядно показывает, что функции-переменные способствуют уменьшению объема программного кода. Если бы не эта возможность, функцию пришлось бы выбирать командой if или switch; это привело бы к заметному увеличению объема программного кода и риску появления дополнительных ошибок при кодировании.

Построение библиотек функций

Библиотеки функций -- одно из самых эффективных средств экономии времени при построении приложений. Предположим, вы написали серию функций для сортировки массива. Вероятно, эти функции будут неоднократно использоваться в разных приложениях. Вместо того чтобы постоянно переписывать эти функции в новый сценарий или копировать их через текстовый буфер, гораздо удобнее разместить все функции сортировки в отдельном файле и присвоить ему легко узнаваемое имя (например, array_sorting.inc). Пример такого файла приведен в листинге 7.

Листинг 7. *Пример библиотеки функций (array_sorting.inc)*

```
<?
// Файл: array_sorting.inc
// Назначение: библиотека функций для сортировки массивов.
// Дата: 17 июля 2000 г.
function merge_sort($array. $tmparray, $right, $left) {
...
function bubble_sort($array. $n) {
...
}
function quicksort ($array. $right. $left) {
...
}
?>
```

Библиотека array_sorting.inc служит накопителем для всех функций сортировки. Это удобно, поскольку функции фактически группируются по своему назначению и при необходимости можно легко найти нужную функцию. Как видно из листинга 7, в начало библиотеки обычно включается заголовок из нескольких строк комментария, чтобы при открытии файла библиотеки можно было сразу получить краткую сводку его содержимого. После собственной библиотеки функций можно включить ее в сценарий при помощи команд PHP include( ) и require( ), в результате чего все функции библиотеки становятся доступными. В общем виде синтаксис этих команд выглядит так:

```
include(путь/имя_файла);
require( путь/имя_файла );
```

Также существует альтернативный вариант:

```
include " путь/имя_файла";
require " путь/имя_файла";
```

где *путь* определяет относительный или абсолютный путь к файлу. Конструкции include( ) и require( ) подробно описаны в главе 9. А пока достаточно запомнить, что эти конструкции используются для включения файла непосредственно в сценарий.

Предположим, вы хотите воспользоваться функциями библиотеки array_sorting.inc в сценарии. Пример включения библиотеки показан в листинге 8.

Листинг 8. *Включение библиотечного файла (array_sorting.inc) в сценарий*

```
// Предполагается, что библиотека array_sorting.inc
// находится в одном каталоге со сценарием.
include("array_sorting.inc");
// Теперь вы можете использовать любые функции из array_sorting.inc
$some_array = array (50, 42. 35, 46);
// Использовать функцию bubble_sort()
$sorted_array = bubble_sort($some_array, 1);
```