Глава 4

# Операции, операторы и массивы

***В этой главе...***

- Символы-разделители
- Операции
- Управляющие операторы
- Массивы

Синтаксис PHP во многом позаимствован из C-подобных языков. Для людей, знакомых с любым таким языком, изучение синтаксиса PHP не составит никакого труда.

## Символы-разделители

Так же, как в C, Java, Pascal и еще множестве других языков, в PHP команды отделяются друг от друга точкой с запятой. Единственное исключение составляет тот случай, когда в блоке PHP-инструкций содержится только одна команда. В этом случае точка с запятой могут опускаться. (С практической точки зрения ее лучше все-таки ставить. Это позволит избежать ошибок при добавлении в такой блок других команд.)

```
<?php
      echo 'First instruction';
      echo 'Second instruction';
?>
```

## Операции

Операции используются для манипулирования значениями одной или нескольких переменных. Простейшим примером может служить операция сложения в арифметике. Операции языка PHP в основном аналогичны операциям в языке C.

Существует несколько групп операций.

- Арифметические, которые определяют простейшие математические действия над переменными.
- Операции присваивания, предназначенные для изменения значения некоторой переменной.
- Логические, позволяющие вычислять булевы выражения.
- Поразрядные, предназначенные для работы с двоичным представлением числа.

Это те операции, которые можно встретить практически в каждом языке программирования. В то же время в языке PHP определено и несколько достаточно специфических операций.

- Конкатенация (обычно эта операция реализуется при помощи библиотечной функции).
- Подавление ошибки.
- Условная операция.

Ниже в табл. 4.1 приведен перечень основных операций языка PHP в порядке их приоритета.

**Таблица 4.1. Основные операции языка PHP**

| Операция | Описание |
|---|---|
| `()` | Определение приоритета |
| `[]` | Доступ к элементу массива по индексу |
| `!` | Логическое отрицание |
| `~` | Поразрядное отрицание |
| `++,--` | Инкремент, декремент |
| `@` | Подавление ошибки |
| `* / %` | Умножение, деление, деление с остатком |
| `+ - .` | Сложение, вычитание, конкатенация |
| `<< >>` | Поразрядный левый сдвиг, правый сдвиг |
| `< <= > >=` | Меньше, меньше или равно, больше, больше или равно |
| `== != === !==` | Равно, не равно |
| `&` | Поразрядное И |
| `^` | Поразрядное исключающее ИЛИ (XOR) |
| `\|` | Поразрядное ИЛИ |
| `&&` | Логическое И |
| `\|\|` | Логическое ИЛИ |
| `?:` | Условная операция |
| `= += -= *= /= .= %= &= != ~= <<= >>=` | Присваивание |
| `and` | Логическое И |
| `xor` | Логическое исключающее ИЛИ |
| `or` | Логическое ИЛИ |

## Арифметические операции

Язык PHP содержит набор арифметических операций, который присутствует в большинстве C-подобных языков программирования. Если операнды `+`, `-`, и `*` являются целыми числами, то результат операции также будет целым. Если какой-либо из операндов окажется числом с плавающей точкой, то результат также будет действительным числом. Операция деления интерпретируется аналогичным образом. Если при выполнении операции над целыми числами возникнет переполнение, то результат будет преобразован в число с двойной точностью (`double`). Ниже в табл. 4.2 приведены арифметические операции языка PHP.

При делении на нуль (0) возникает ошибка.

**Таблица 4.2. Арифметические операции**

| Выражение | Результат |
|---|---|
| `$a+$b` | Сумма `$a` и `$b` |
| `$a-$b` | Разница `$a` и `$b` |
| `$a*$b` | Произведение `$a` и `$b` |
| `$a/$b` | Частное `$a` и `$b` |
| `$a%$b` | Остаток от деления `$a` и `$b` |

## Операции присваивания

В общем случае выражение `$a=`*выражение* следует понимать как "переменной `a` присвоить результат вычисления *выражения*". То есть команду типа `$a=$a+1` следует трактовать следующим образом. Сначала берется значение переменной `a`, потом к нему добавляется `1`, а результат сохраняется где-то в оперативной памяти. После этого результат извлекается из памяти и присваивается переменной `a`, которая в результате увеличивается на 1.

Для удобства в PHP определено несколько составных операций присваивания, при которых кроме присваивания выполняется еще какая-нибудь операция. По возможности лучше использовать именно их, поскольку это приводит к более компактной записи, и позволяет интерпретатору более эффективно обработать такую команду. Составные операции и эквивалентные им обычные операции приведены в табл. 4.3.

**Таблица 4.3. Составные операции языка PHP**

| Выражение | Эквивалент |
|---|---|
| `$a+=$b` | `$a=$a+$b` |
| `$a-=$b` | `$a=$a-$b` |
| `$a*=$b` | `$a=$a*$b` |
| `$a/=$b` | `$a=$a/$b` |
| `$a .=$b` | `$a=$a.$b` |
| `$a%=$b` | `$a=$a%$b` |
| `$a&=$b` | `$a=$a&$b` |
| `$a!=$b` | `$a=!$b` |
| `$a~=$b` | `$a=~$b` |
| `$a<<=$b` | `$a=$a<<$b` |
| `$a>>=$b` | `$a=$a>>$b` |

## Логические операции

Логические операции описаны в табл. 4.4.

Операторы `&&` и `and`, `||` и `or` отличаются только приоритетом.

## Поразрядные операции

Поразрядные операции предназначены для выполнения логических, а также некоторых других операций над отдельными битами операндов. Значение 1 бит соответствует `true`, а значение 0 — `false` (табл. 4.5).

**Таблица 4.4. Логические операции языка PHP**

| Выражение | Имя операции | Результат |
|---|---|---|
| `$a && $b` | Логическое И | `true`, если `a` и `b` равны `true` |
| `$a \|\| $b` | Логическое ИЛИ | `false`, если `a` и `b` равны `false` |
| `!$a` | Логическое НЕ | `true`, если `a` равно `false`; `false`, если `a` равно `true` |
| `$a and $b` | Логическое И | Аналогично `&&` |
| `$a or $b` | Логическое ИЛИ | Аналогично `\|\|` |
| `$a xor $b` | Логическое исключающее ИЛИ | `true`, если `a` и `b` имеют различные значения |

**Таблица 4.5. Поразрядные операции языка PHP**

| Выражение | Результат |
|---|---|
| `$a & $b` | Побитовое И |
| `$a \| $b` | Побитовое ИЛИ |
| `$a ^ $b` | Побитовое исключающее ИЛИ |
| `~$a` | Побитовое НЕ (инверсия) |
| `$a<<$b` | Сдвиг влево на `$b` бит (с заполнением нулями) |
| `$a>>$b` | Сдвиг вправо на `$b` бит (с заполнением нулями) |

## Операции сравнения

Операции сравнения проверяют некоторое условие и возвращают `true` при его выполнении и `false` — в противном случае (табл. 4.6).

**Таблица 4.6. Операции сравнения в PHP**

| Выражение | Результат |
|---|---|
| `$a == $b` | `true`, если `$a` равно `$b` |
| `$a != $b` | `true`, если `$a` не равно `$b` |
| `$a === $b` | `true`, если `$a` равно `$b`, а `$a` и `$b` имеют одинаковый тип |
| `$a !== $b` | `true`, если `$a` не равно `$b`, или `$a` и `$b` имеют разные типы |
| `$a <= $b` | `true`, если `$a` меньше или равно `$b` |
| `$a >= $b` | `true`, если `$a` больше или равно `$b` |
| `$a < $b` | `true`, если `$a` меньше `$b` |
| `$a > $b` | `true`, если `$a` больше `$b` |

Отдельно следует упомянуть стандартную ошибку начинающих программистов, вместо операции == использующих операцию присваивания =. Результат присваивания обычно равен `true` (если только не присваивается нулевое значение). В результате программа начинает вести себя совсем не так, как планировалось. Подобные ошибки обнаружить достаточно сложно. Поэтому в процессе написания программы надо тщательно следить за программным кодом.

## Инкрементирование и декрементирование

Инкрементирование (декрементирование) — это операция увеличения (или уменьшения) переменной на 1.

Различают префиксную и постфиксную формы инкрементирования/декремен-тирования — в зависимости от того, до или после имени переменной расположен символ операции (табл. 4.7).

**Таблица 4.7. Декрементирование и инкрементирование в языке PHP**

| Выражение | Результат |
|---|---|
| `$a++` | Возвращается `$a`, после этого значение `$a` увеличивается на 1 |
| `$a--` | Аналогично, но с уменьшением на 1 |
| `++$a` | Увеличивается значение `$a`, после этого возвращается значение `$a` |
| `--$a` | Аналогично, но с уменьшением на 1 |

Операции инкрементирования и декрементирования (как и многие другие операции) в языке PHP полностью аналогичны соответствующим операциям языка C/C++. Так что описание их использования можно без проблем найти в соответствующей литературе.

## Другие операции

`( )`

Операция изменения приоритета выполнения операций. Операции, заключенные в круглых скобках, выполняются в первую очередь.

`[]`

Используя эту операцию, можно получить значение элемента массива по его индексу. Массивы рассматриваются в этой главе ниже.

`@`

Операция подавления ошибки. Если при выполнении строки с этим символом произойдет ошибка, то диагностическое сообщение будет "подавлено". Например, интерпретация строки `@echo(5/0);` не приведет ни к какому результату. Та же строка, но без символа `@`, приведет к генерации сообщения об ошибке деления на нуль.

`.`

Операция конкатенации строк, т.е. "дописывание" одной строки в конец другой. Например, результат роботы команды `echo("My name is "."Michael");` будет `My Name is Michael`.

# Управляющие операторы

Любой сценарий PHP состоит из выражений. В состав выражения может входить операция присваивания, вызов функции, а также управляющие операторы (условные операторы, циклы, блоки ветвления и другие). Как правило, каждое выражение заканчивается точкой с запятой (`;`). Выражения могут быть сгруппированы и образовывать сложное выражение.

Любой сценарий или программа выполняются (или интерпретируются) линейно, т.е. сверху вниз. Управляющие операторы позволяют это изменить, многократно выполнив требуемый фрагмент кода или пропустив его, в зависимости от результата проверки какого-либо условия.

## Оператор `if-else-elseif`

Данная конструкция является одной из самых важных практически в любом языке программирования. И PHP не является исключением. Этот управляющий оператор позволяет выполнять или пропускать определенные фрагменты кода при заданных условиях. Общий вид оператора `if` приведен ниже.

```
if (условие 1) {
   блок кода 1
} elseif (условие 2) {
...
} elseif (условие N) {
   блок кода N
} else {
   альтернативый блок кода
}
```

Когда *условие 1* истинно (т.е. его результатом является значение `true`), выполняется лишь *блок кода 1*. В противном случае проверяется *условие 2* и т.д. Если все условные выражения оказались ложными, выполняется *альтернативный блок кода*, определенный оператором `else`. Например, так.

```
if ($a > $b) {
   print "a is bigger than b";
} elseif ($a == $b) {
   print "a is equal b";
} else {
   print "b is bigger than a";
}
```

В языке PHP имеется также тернарная операция `?:` со следующим синтаксисом.

```
((выражение1) ? (выражение2) : (выражение3))
```

Эта конструкция интерпретируется следующим образом. Значение всего выражения равно *выражению2*, если *выражение1* истинно. В противном случае (т.е. когда *выражение1* ложно) значением всего выражения считается *выражение3*.

## Цикл `while`

Цикл проще всего реализовать с помощью следующей конструкции.

```
while (условие) {

   блок кода;

}
```

Блок кода многократно выполняется до тех пор, пока *условие* остается истинным. Условие проверяется каждый раз в начале цикла. Если *условие* принимает значение `false` сразу на первой итерации, то операторы тела цикла не выполнятся ни одного раза.
Ниже приведен пример использования оператора `while` для вывода чисел от 1 до 10.

```
<?php
$i = 1;
while ($i <= 10) {
   echo $i++;
}
?>
```

## Цикл do-while

Этот цикл очень похож на предыдущий, за исключением того, что *условие* проверяется в конце каждой итерации после выполнения операторов тела цикла. Другими словами, гарантируется, что операторы тела цикла будут выполнены хотя бы один раз.

```
do {
  блок кода;
} while (условие);
```

Вот пример использования цикла do — while.

```
$i = 10;
do {
   print $i;
}
while ($i > 20);
```

В приведенном примере переменная $i будет выведена лишь один раз, поскольку условие цикла не выполняется.

## Цикл for

В языке PHP наиболее сложные циклы можно реализовать с использованием оператора for. Для этого используется следующий синтаксис.

```
for (выражение 1; выражение2; выражение 3) {
   блок кода;
}
```

*Выражение 1* выполняется один раз в начале первой итерации без проверки каких бы то ни было условий. Как правило, это выражение используется для инициализации переменной-счетчика цикла. В начале каждой итерации проверяется *выражение 2*. Если оно истинно, то выполняется *блок кода*, в противном случае выполнение цикла прекращается. В конце каждой итерации выполняется *выражение 3*. Каждое из выражений может быть пустым. Рассмотрим популярный пример (его можно найти в документации по языку PHP), в котором различными способами с помощью оператора цикла for выводятся числа от 1 до 10.

**Пример 1** ("классический" способ).

```
for ($i=1; $i <= 10; $i++) {
     echo $i;
}
```

**Пример 2**.

```
for ($i=1;; $i++) {
   if ($i > 10) break;
   echo $i;
}
```

**Пример 3**.

```
$i = 1;
for (;;) {
   if ($i > 10) break;
   echo $i;
   $i++;
}
```

**Пример 4**.

```
for ($i=1; $i<=10; echo $i, $i++) {}
```

Несмотря на все разнообразие, рекомендуется использовать "классический" способ, приведенный в первом примере. Операторы `break` и `continue` будут рассмотрены немного ниже.

## Оператор `foreach`

Этот оператор появился в PHP версии 4. Он позволяет выполнять заданные действия над всеми элементами массива. Массивы более подробно будут рассмотрены ниже в этой главе. Так что при необходимости к этому пункту можно еще раз вернуться позже.
При использовании оператора `foreach` применяется следующий синтаксис.

```
foreach (массив as $value) {
}
foreach (массив as $key=>$value) {
}
```

В первом случае на каждой итерации значение текущего элемента массива присваивается переменной `$value`, и текущий указатель массива перемещается на один элемент вперед (т.е. на следующей итерации данной переменной будет присвоено значение следующего элемента). Во втором случае выполняется то же самое, но значение ключа массива присваивается переменной `$key`. Например, так.

```
<?php
   $arr = array('one', 'two', three');
   foreach ($arr as $value) {
      echo 'Value: '.$value.'<br>';
   }
?>
```

В приведенном фрагменте выводятся значения элементов массива.

## Оператор `switch`

Данный оператор аналогичен нескольким инструкциям `if`. Зачастую значение одной и той же переменной необходимо сравнить с несколькими другими значениями и в зависимости от этого выполнить различные действия. Тогда лучше всего воспользоваться именно оператором `switch`.

```
<?php
switch ($value) {
     case 'apple' :
          echo 'This is an apple!';
          break;
     case 'pear' :
          echo 'This is a pear!';
          break;
     default :
          echo 'Hm.. Who knows...';
}
?>
```

Если значение переменной `$value` равно `apple` или `pear`, будет выполнен соответствующий блок кода с последующим выходом из оператора `switch` (с использованием оператора `break`). Если же значение не совпадает ни с одним из перечисленных, то выполнится блок инструкций, задаваемый с помощью ключевого слова `default`.

## Операторы `break` и `continue`

Оператор `break` позволяет прервать выполнение циклов и других управляющих конструкций, реализованных с помощью операторов `for`, `foreach`, `while`, `do...while`, `switch`.

Оператор `continue` можно применять внутри циклов, чтобы прервать выполнение оставшейся части текущей итерации и перейти к следующей итерации.

Если значение переменной совпало со значением одного из блоков `case`, то будут выполнены все последующие инструкции вплоть до ближайшего оператора `break`. Таким образом, при отсутствии этого оператора будут последовательно выполнены все операторы до завершающей скобки блока `switch`.

```
switch() ($i) {
   case 0:
      echo $i;
   case 1:
      echo $i;
   case 2:
      echo $i;
}
```

Если в переменной `$i` содержится значение `0`, то при интерпретации этого фрагмента значение переменной `$i` будет выведено три раза. Чтобы этого не случилось, необходимо воспользоваться оператором `break`.

```
switch() ($i) {
   case 0:
      echo $i;
      break;
   case 1:
      ...
}
```

## Оператор `return`

При использовании выражения `return (arg)` внутри функции ее выполнение будет немедленно завершено, а аргумент `arg` будет использоваться в качестве возвращаемого значения.

При использовании оператора `return` в глобальной области видимости выполнение данного сценария немедленно будет прекращено.

## Альтернативная форма записи управляющих операторов

При использовании управляющих операторов `if`, `switch`, `for` и `while` можно применить альтернативную форму записи. Вместо круглых скобок, с помощью которых определяется тело управляющего оператора, можно воспользоваться открывающим символом-разделителем `:`. Однако следует помнить о том, что каждый из четырех управляющих операторов имеет собственный закрывающий оператор — `endif`, `endswitch`, `endfor` и `endwhile`. Например, следующие два оператора `while` абсолютно идентичны.

```
while ($a < 10) {
   $a =a$ +$b;
   ++$b;
}

while ($a < 10) :
   $a =a$ +$b;
   ++$b;
endwhile;
```

Основным преимуществом такой формы записи является повышение читабельности программного кода.

Если альтернативный синтаксис используется в операторе `if`, в состав которого входит и оператор `else`, то считается, что тело оператора `if` открывается с помощью двоеточия, а закрывается оператором `else`. В свою очередь, тело оператора `else` открывается с использованием двоеточия, а закрывается с помощью оператора `endif`. Например, так.

```
if ($a > $b) :
   $maximum = $a;
   print("maximum is $a");
else :
   &large = $b;
   print("maximum is $b");
endif;
```

# Массивы

Массивы в языке PHP отличаются от аналогичных конструкций любого другого языка программирования. Их лучше всего рассматривать как комбинацию традиционного массива и хэш-таблицы Perl. Это делает их чрезвычайно гибкой встроенной структурой данных. Отдельные элементы массива аналогичны элементам хэш-таблицы языка Perl: каждый из них представляет собой пару "ключ-значение". Если логическая структура массива PHP аналогична массиву в любом другом языке, то соответствующие ключи являются простыми положительными числами. Эти числа всегда расположены по возрастанию. Если же логическая структура массива PHP аналогична хэш-таблице Perl, то ключи представляют собой строки, а порядок элементов массива определяется с использованием хэширующей функции. Интересен тот факт, что в состав массива одновременно могут входить элементы как с целочисленными ключами, так и со строковыми.

## Создание массива

В языке PHP массив можно создать двумя способами. Для создания скалярных переменных используется операция присваивания. Ее можно применять и для создания массивов. Присваивание значения элементу массива, который до этого еще не существовал, приведет к созданию этого массива. Например, предположим, что в текущий момент массив `$mass` еще не существует. Тогда следующее выражение приведет к его созданию.

```
$mass[0] = 5;
```

Обратите внимание на то, что, хотя массивы PHP напоминают хэш-таблицы Perl, их имена начинаются с символа доллара (`$`), как и любая другая переменная PHP. Поэтому если в сценарии до выполнения присваивания уже существовала скалярная переменная `$mass`, то после выполнения этой операции она будет преобразована в массив. Если при присвоении значения элементу массива не указывается его индекс, он выбирается неявным образом. При этом используемый индекс будет на единицу больше, чем максимальное значение целочисленного индекса, которое использовалось до сих пор. Если же окажется, что до присваивания в массиве отсутствовали элементы с числовыми ключами, то будет использоваться ключ `0`. Например, в следующей строке в качестве второго ключа элемента будет применяться значение `2`.

```
$mass[1] = "Today is my birthday!";
$mass[] = 50;
```

Из приведенного примера также следует, что элементы массива не обязательно должны иметь один и тот же тип.

Второй способ создания массива заключается в использовании оператора `array`. Параметры этого оператора задают значения, которые будут размещены в новом массиве, а при необходимости — соответствующие ключи. Если массив планируется использовать традиционным способом (т.е. без ключей), то в операторе `array` можно задать только значения. (При этом интерпретатор PHP автоматически присвоит ключам целые значения.) Например,

```
$mass = array(10, 20, 30, 40);
```

При таком присваивании будет создан обычный массив с четырьмя элементами и ключами `0`, `1`, `2` и `3`. Если нужно указать другие ключи, то это можно осуществить следующим образом.

```
$mass = array(1 => 10, 2 => 20, 3 => 30, 4 => 40);
```

Если при создании массива с использованием оператора `array` не указаны параметры, будет создан пустой массив.

```
$mass = array();
```

В следующей строке создается массив, который в точности будет соответствовать хэш-таблице Perl.

```
$ages = array("Ivan" => 40, "Mary" => 17, "Alex" => 25);
```

Массивы могут возвращаться некоторыми функциями. В частности, функциями, которые предназначены для доступа к базам данных.

В языке PHP массивы не обязательно должны соответствовать либо традиционному массиву, либо хэш-таблице. Они могут представлять собой некоторую "смесь" таких конструкций. Например, следующая запись синтаксически будет абсолютно корректной.

```
$auto = array("make" => "AZLK", "model" => 2110, "year" => 1992, 3 => "sold");
```

## Доступ к элементам массива

К отдельным элементам массива доступ можно получить по индексу, как и в других традиционных языках программирования. При этом заключенный в квадратные скобки индекс является ключом искомого значения. Скобки вставляются независимо от того, целочисленный или строковый ключ используется. Например, значение элемента массива `$ages` с ключом "`Mary`" можно получить следующим образом.

```
print("Mary is $ages['Mary'] years old <br />");
```

Значения нескольких элементов массива можно присвоить скалярным переменным в одном операторе. Например, так.

```
$trees = array("oak", "pine", "binary");
$list($hardwood, $softwood, $data_structure) = $trees;
```

В этом фрагменте переменным `$hardwood`, `$softwood` и `$data_structure` присваиваются значения элементов массива "`oak`", "`pine`" и "`binary`" соответственно.

С использованием встроенных функций PHP из массива можно извлечь отдельно набор ключей и набор значений. В качестве параметра функция `array_keys` получает массив, а возвращает массив ключей. При этом ключами результирующего массива являются числа `0`, `1` и т.д. Функция `array_values` позволяет получить аналогичный массив со значениями массива-параметра. Например, так.

```
$mass = array("one" => 74, "two" => 70, "three" => 67,
              "four" => 62, "five" => 65);
$numbers = array_keys($mass);
$values = array_values($mass);
```

Теперь в массиве $numbers содержатся значения “one”, “two”, “three”, “four”, “five”, а в массиве $values — 74, 70, 67, 62, 65.

## Манипуляции с массивами

Как и любую скалярную переменную, массив можно удалить с помощью функции unset(). С использованием этой функции можно удалить также и отдельные элементы массива. Например, следующим образом.

```
$mass = array(2, 4, 6, 8);
$unset($mass[2]);
```

Теперь в массиве $mass осталось три элемента 2, 4 и 8 с ключами 0, 1 и 3.

Функция is_array аналогична is_int: в качестве параметра ей передается имя переменной, а возвращается значение true, если эта переменная — массив, и false — в противном случае. Две идентичные функции count и sizeof получают в качестве параметра массив, а возвращают количество его элементов. Функция in_array имеет два входных параметра, выражение и массив, а возвращает значение true, если результат выражения является одним из значений массива, и false — в противном случае.

Иногда бывает полезным выполнить преобразование между строками и массивами. Это можно осуществить с помощью функций implode и explode. Функция explode разделяет строки на подстроки, используя в качестве символа разделителя первый параметр, и размещает их в массиве. Вторым параметром является строка, которую нужно разместить в массиве. Например, рассмотрим следующий пример.

```
$str = "April in Kiev, Sevastopol is nice";
$words = explode(" ", $str);
```

После интерпретации приведенного фрагмента в массиве $words будут содержаться значения “April”, “in” “Kiev,”, “Sevastopol”, “is” и “nice”.

Действие функции implode противоположно функции explode. Получив в качестве параметров символ-разделитель и массив, эта функция выполнит конкатенацию элементов массива. При этом между этими элементами будет вставлен символ-разделитель. Возвращаемое значение будет строкой. Например,

```
$words = array("Are", "you", "working", "today");
$str = implode(" ", $words);
```

После обработки этого фрагмента в строке $str будет содержаться значение Are you working today.

Внутри массива элементы хранятся в виде связного списка “ячеек”, в которых одновременно содержатся как ключ, так и значение. Сами ячейки размещаются в памяти с использованием функции хэширования, так что они произвольным образом размещаются в рамках зарезервированного блока памяти. Доступ к элементам с использованием строковых ключей реализуется через функцию хэширования. Однако в то же время все элементы содержат связи друг с другом в том порядке, в котором они были созданы. Это обеспечивает возможность доступа к элементам массива в порядке их создания при использовании как строковых ключей, так и числовых. Доступ к элементам массива в порядке их создания более подробно рассматривается в следующем разделе.

На рис. 4.1 показана внутренняя логическая структура массива, которая иллюстрирует два различных способа доступа к элементам массива.

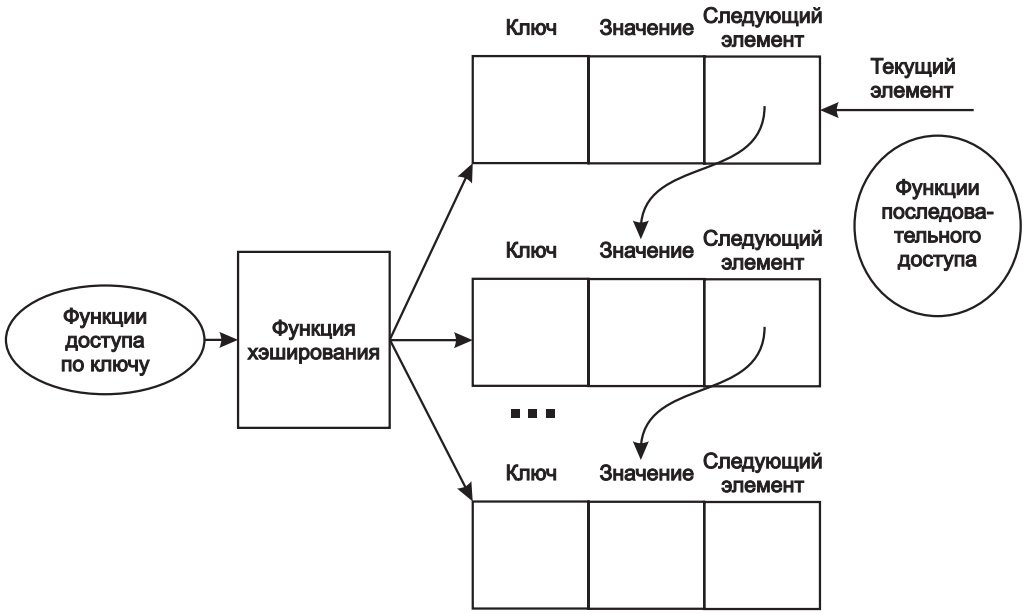


**Рис. 4.1.** Внутренняя логическая структура массива в PHP

## Последовательный доступ к элементам массива

В языке PHP последовательный доступ к элементам массива можно получить различными способами. С каждым массивом связан внутренний указатель, или маркер, который ссылается на один элемент массива. Такой указатель называется *текущим*. При создании массива этот маркер инициализируется таким образом, чтобы он ссылался на первый элемент массива. Элемент массива, на который ссылается такой указатель, можно получить с помощью функции `current`. Например, рассмотрим следующий код.

```
$cities = array("Kiev", "Odessa", "Rovno", "Simferopol");
$city = current($cities);
print("The first city is $city <br />");
```

После интерпретации этого фрагмента будет получена строка `The first city is Kiev`.

Текущий указатель можно переместить с помощью функции `next`. Эта функция перемещает указатель к следующему элементу массива, а также возвращает его значение. Если ранее текущий указатель ссылался на последний элемент массива, то при вызове функции `next` будет возвращено значение `false`. Например, если текущий указатель ссылается на первый элемент массива `$cities`, то следующий код позволит получить список всех элементов этого массива.

```
$city = current($cities);
print("$city <br />")
while($city = next($cities))
     print("$city <br />");
```

При использовании функции `next` возникает одна проблема, связанная с управлением циклом. Если в массиве имеется элемент со значением `false`, то цикл будет прерван не после достижения конца массива, а из-за того, что был найден элемент с этим значением.

Описанную выше проблему позволяет обойти функция each, которая возвращает двухэлементный массив, состоящий из ключа и значения текущего элемента. Эта функция вернет значение false только в том случае, если текущий указатель "пройдет" последний элемент массива. Ключами двух элементов результирующего массива являются строки "key" и "value". Еще одно различие между функциями each и next заключается в том, что функция each сначала возвращает элемент, на который ссылается текущий указатель, и лишь затем перемещает сам указатель, а функция next — наоборот. Пример использования функции each приведен в следующем фрагменте.

```
$salary = array("Ivan" => 400, "Mary" => 550,
                "Alex" => 320);
while($employee = each($salaries))
{
  $name = $employee["key"];
  $salary = $employee["value"];
  print("The salary of $name is $salary <br />");
}
```

После интерпретации этого фрагмента кода будет получен следующий результат.

```
The salary of Ivan is 400
The salary of Mary is 550
The salary of Alex is 320
```

С помощью функции prev текущий указатель можно переместить в обратном направлении. Как и функция next, prev возвращает значение элемента, на который ссылается текущий указатель, и лишь после этого перемещает сам указатель.

Текущий указатель можно переместить на первый элемент массива с помощью функции reset. Эта функция также возвращает и значение первого элемента массива. С использованием функции last текущий указатель можно переместить на последний элемент. Эта функция возвращает также и значение самого элемента.

Функция key позволяет получить ключ текущего элемента массива, если в качестве параметра ей передать имя самого массива.

Функции array_push и array_pop предоставляют простой способ реализации в массиве стека. Первым параметром этой функции является массив, после которого можно указать любое количество дополнительных параметров. Значения, содержащиеся в этих параметрах, размещаются с конца массива. Функция array_push возвращает новое количество элементов массива. Функция array_pop имеет единственный параметр — имя массива. Она возвращает последний элемент массива, а затем удаляет его. Если массив-стек пуст, то возвращается значение NULL.

Оператор foreach специально предназначен для построения циклов, в которых обрабатываются все элементы массива. Этот оператор можно использовать в двух формах.

```
foreach(<массив> as <скалярная_переменная>) <тело_цикла>
foreach(<массив> as <ключ> => <значение>) <тело_цикла>
```

При использовании первой формы оператора foreach на каждой итерации цикла скалярной переменной присваивается одно из значений массива. При этом перед выполнением первой итерации текущий указатель массива неявно инициализируется как при использовании функции reset. Например, так.

```
foreach($mass as $temp)
{
  print("$temp <br />");
  ...
}
```

При выполнении этого фрагмента будут получены значения всех элементов массива $mass.

Вторая форма оператора foreach позволяет получить как ключи, так и значения всех элементов массива. Например, следующим образом.

```
$temperature = array("January" => -23, "February" => -18, "March" => 10);
foreach($temperature as $month => $temp)
{
  print("The low temperature on $month was $temp <br />");
...
}
```

## Сортировка массивов

В качестве входного параметра функция `sort` получает имя массива, выполняет сортировку хранящихся в этом массиве значений, а затем удаляет ключи. В массиве могут содержаться как строковые, так и числовые значения. При этом строковые значения перемещаются в начало массива в алфавитном порядке. Затем по возрастанию размещаются числовые значения. При этом независимо от ключей исходного массива в отсортированном массиве используются ключи `0`, `1`, `2` и т.д. Функция `sort` лучше всего подходит для сортировки традиционных массивов, в которых содержатся только строковые или числовые значения.

Функция `sort` имеет и второй параметр — `SORT_NUMERIC`. При его использовании по возрастанию будут отсортированы лишь числовые значения входного массива, а строковые значения останутся без изменений. Однако следует учитывать, что строковые значения все же будут перемещены в начало массива.

Функция `asort` позволяет выполнять сортировку массивов, которые аналогичны хэш-таблицам языка Perl. Эта функция сортирует элементы заданного массива, однако при этом сохраняет исходные связи "ключ-значение". Как и функция `sort`, `asort` перемещает строковые значения в начало массива, а после них по возрастанию размещает числовые значения. Функция `asort` также позволяет использовать параметр `SORT_NUMERIC`.

Функции `rsort` и `arsort` аналогичны функциям `sort` и `asort` соответственно. Единственное различие состоит в том, что при их использовании сортировка выполняется в обратном порядке.

## Пример

В приведенном ниже сценарии `example4-1.php` демонстрируется применение функций `sort`, `rsort`, `asort` и `arsort`. При этом следует помнить, что файл `example3-1.php` должен быть размещен в каталоге Web-сервера, который по умолчанию используется для хранения документов. Кроме того, не забывайте, что основное назначение сценариев PHP состоит в динамической генерации содержимого клиентских HTML-страниц. То есть результаты интерпретации таких сценариев вставляются в текст страницы, которая затем загружается в клиентский браузер.

**Исходный код сценария `example4-1.php`**

```
<!DOCTYPE html PUBLIC "//w3c/DTD XHTML 1.0 Strict//EN"
"http://www.w3c.org/TR/xhtml/DTD/xhtml-strict.dtd">

<!-- example4-1.php - Simple example to illustrate several sorting
functions -->
<html>
<head> <title> example4-1.php (Sorting) </title>
</head>
<body>
<?php
   $original = array("one" => 10, "two" => 20,
                     "three" => 30, "four" => "tree");
?>
<h4> Original Array </h4>
```

```
<?php
   foreach($original as $key =>$value) {
      print("[$key] => $value <br />");
   }

   $new_array = $original;
   sort($new_array);
?>
<h4> Array sorted with sort function </h4>
<?php
   foreach($new_array as $key => $value) {
      print("[$key] => $value <br />");
   }

   $new_array = $original;
   sort($new_original, SORT_NUMERIC);
?>
<h4> Array sorted with sort function and SORT_NUMERIC parameter </h4>
<?php
   foreach($new_array as $key => $value) {
      print("[$key] => $value <br />");
   }

$new_array = $original;
   rsort($new_original);
?>
<h4> Array sorted with rsort function </h4>
<?php
   foreach($new_array as $key => $value) {
      print("[$key] => $value <br />");
   }

$new_array = $original;
   asort($new_original);
?>
<h4> Array sorted with asort function </h4>
<?php
   foreach($new_array as $key => $value) {
      print("[$key] => $value <br />");
   }

$new_array = $original;
   arsort($new_original);
?>
<h4> Array sorted with arsort function </h4>
<?php
   foreach($new_array as $key => $value) {
      print("[$key] => $value <br />");
   }
?>
</body>
</html>
```

Фрагмент результата выполнения сценария `example4-1.php` в окне браузера представлен на рис. 4.2.

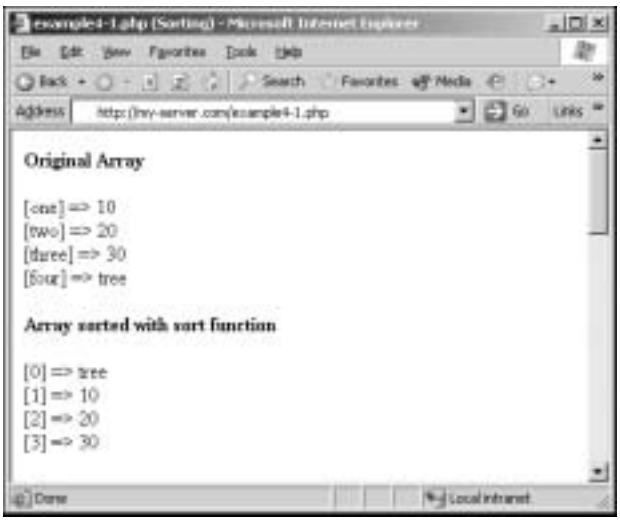


**Рис. 4.2.** Результат выполнения сценария `example4-1.php`

## Резюме

- В языке PHP, как и в любом другом C-подобном языке программирования, определен набор операций для обработки скалярных данных. Во многом операции PHP аналогичны операциям других языков. В то же время существуют некоторые уникальные операции, которые в других языках отсутствуют, например, конкатенация строк.
- Для изменения хода выполнения сценариев можно использовать различные управляющие операторы. Для многократного выполнения заданной последовательности команд можно применять циклы. В языке PHP определено три различных оператора циклов `while`, `do-while` и `for`. Имеется также оператор `foreach`, который специально предназначен для обработки массивов.
- К другим управляющим конструкциям PHP относятся `if`, `else`, `elseif`, `break`, `continue` и `switch`.
- Массивы PHP заслуживают самого пристального изучения, поскольку они предоставляют гораздо больше возможностей, чем язык C++. Массивы PHP являются комбинацией традиционных массивов, которые можно использовать во многих языках, и ассоциативных массивов языка Perl. В частности, любой массив PHP является ассоциативным и может содержать пары "ключ-значение". Одновременно в массиве могут храниться значения различных типов.
- Для обработки массивов в языке PHP предусмотрен богатый выбор специализированных функций, которые можно использовать для перебора значения элементов в любом направлении, а также для их сортировки.

## Контрольные вопросы

1. Назовите основные управляющие операторы языка PHP. Сравните их синтаксис с написанием аналогичных операторов C++.
2. В чем состоит особенность массивов в PHP?

3. Охарактеризуйте основные функции PHP для работы с массивами.
4. Приведите примеры трех типов циклов в языке PHP.
5. Назовите основные способы обращения к элементам массива в PHP.