Тема 3.5
«Строковые функции PHP»

Вступление

Прежде, чем рассматривать массивы, мы рассмотрим строковые функции PHP, поскольку они будут активно использоваться в примерах, посвящённых массивам.

В PHP присутствует огромное количество строковых функций, применение которых позволяет быстро и легко решать множество разнообразных задач.

Также следует помнить несколько фактов о строках в PHP:

- на длину строки нет никаких ограничений (в PHP строка может быть такой длины, какой объём памяти может выделить операционная система, естественно, с учётом ограничений, установленных для отдельно выполняющегося скрипта);
- PHP умеет рассматривать строки как массивы символов;
- PHP рассматривает как строки любые последовательности байт (так, например, прочитанный в память графический файл будет с точки зрения PHP строкой).

Рассмотрим функции…

Работа с HTML: htmlspecialchars

string htmlspecialchars (string string [, int quote_style [, string charset]]) – преобразует специальные символы в «HTML сущности»:

& (амперсанд) преобразуется в «&»;

“ (двойная кавычка) преобразуется в «"»;

‘ (одиночная кавычка) преобразуется в «'»;

< (знак «меньше») преобразуется в «<»;

> (знак «больше») преобразуется в «>»;

Эта функция полезна при отображении данных, введённых пользователем, которые могут содержать нежелательные HTML-теги, например в форуме или гостевой книге.

Работа с HTML: htmlspecialchars

Необязательный второй аргумент quote_style определяет режим обработки одиночных и двойных кавычек:

- в режиме по умолчанию (ENT_COMPAT), преобразуются двойные кавычки, одиночные остаются без изменений;
- в режиме ENT_QUOTES преобразуются и двойные, и одиночные кавычки;
- в режиме ENT_NOQUOTES и двойные, и одиночные кавычки остаются без изменений.

Пример:

```
$new = htmlspecialchars("<a href='test'>Test</a>", ENT_QUOTES);
echo $new;
// выведется: &lt;a href=&#039;test&#039;&gt;Test&lt;/a&gt;
```

Необязательный третий аргумент charset определяет кодировку, используемую при преобразовании. По умолчанию используется кодировка ISO-8859-1.

Для обратного преобразования существует функция htmlspecialchars_decode() (начиная с PHP 5.1.0).

Работа с HTML: htmlentites

Если вам нужно преобразовать все возможные сущности, используйте функцию

string htmlentities (string string [, int quote_style [, string charset]])

которая производит преобразование символов в их представление в виде HTML-сущностей для ВСЕХ символов, для которых существуют HTML-сущности.

В остальном поведение этой функции полностью аналогично поведению функции htmlspecialchars().

Работа с HTML: htmlentites

Если вам нужно преобразовать все возможные сущности, используйте функцию

string htmlentities (string string [, int quote_style [, string charset]])

которая производит преобразование символов в их представление в виде HTML-сущностей для ВСЕХ символов, для которых существуют HTML-сущности.

В остальном поведение этой функции полностью аналогично поведению функции htmlspecialchars().

Для обратного преобразования существует функция

string html_entity_decode (string string [, int quote_style [, string charset]])

Работа с HTML: strip_tags

string strip_tags (string str [, string allowable_tags]) – возвращает строку, из которой удалены HTML- и PHP-теги.

Необязательный второй аргумент может быть использован для указания тегов, которые не должны удаляться.

С версии PHP 4.3.0 удаляются также HTML комментарии.

Внимание! Так как strip_tags() не проверяет корректность HTML кода, незавершённые тэги могут привести к удалению текста, не входящего в тэги!

Пример:

```
$text = '<p>Параграф.</p> <!-- Comment --> Еще немного текста';
echo strip_tags($text).”<br />”;
echo strip_tags($text, '<p>'); // не удалять <p>
```

Результат:

```
Параграф.  Еще немного текста
<p>Параграф.</p>  Еще немного текста
```

Эта функция не изменяет атрибуты тегов, указанных в аргументе allowable_tags.

С версии PHP 5.0.0 strip_tags() безопасна для обработки данных в двоичной форме («binary safe»).

Техническое отступление

В руководстве по PHP можно часто увидеть что так или иная функция является или не является «binary safe» (безопасной для двоичных данных).

«binary safe» функции корректно воспринимают все 256 значений байта «просто как значения», не рассматривая некоторые из них как символы переноса строк и т.д.

Это значит, что такими функциями можно обработать любой «двоичный поток», например, картинки, аудиоролики и т.п.

Поиск в тексте

int strpos (string haystack, string needle [, int offset]) -- возвращает позицию первого вхождения подстроки needle в строку haystack.

Внимание! Эта функция может возвращать как логическое значение FALSE, так и не относящееся к логическому типу значение, которое приводится к FALSE, т.е. 0. Используйте оператор === для проверки значения, возвращаемого этой функцией.

Пример:

```
$mystring = ‘lala hahaha abc lala haha abc';
$findme   = 'abc';
$pos = strpos($mystring, $findme);
if ($pos === false)
        echo "Строка '$findme' не найдена в строке '$mystring'";
 else echo "Строка '$findme' найдена в строке '$mystring’ в позиции $pos";
```

Если needle не является строкой, он приводится к целому и трактуется как код символа.

Необязательный аргумент offset позволяет указать, с какого по счёту символа строки haystack начинать поиск (однако найденная позиция искомой строки будет отсчитываться всё равно от начала строки).

Поиск в тексте

int strrpos (string haystack, string needle [, int offset]) – возвращает позицию последнего вхождения needle в строку haystack.

Если подстрока needle не найдена, возвращает FALSE.

Если needle не является строкой, он приводится к целому и трактуется как код символа.

Начиная с PHP 5.0.0, необязательный аргумент offset позволяет указать, с какого по счёту символа строки haystack начинать поиск. Отрицательное значение предписывает прекратить поиск при достижении определённой позиции до конца строки.

Поиск в тексте

Функции stripos и strripos выполняют действия, аналогичные действиям функций strpos и strrpos, но не учитывают регистр символов.

Примечание: возможно, вы уже заметили, что аргументы рассматриваемых нами сейчас функций называются needle (иголка) и haystack (стог сена). Такие названия даны им для простоты понимания, «что в чём ищется» согласно пословице «искать иголку в стоге сена».

Поиск в тексте

string strstr (string haystack, string needle) – возвращает подстроку строки haystack с первого вхождения needle до конца строки.

Если подстрока needle не найдена, возвращает FALSE.

Если needle не является строкой, он приводится к целому и трактуется как код символа.

Пример:

```
$email = 'user@example.com';
$domain = strstr($email, '@');
echo $domain; // выводит @example.com
```

Функция

string stristr (string haystack, string needle)

выполняет аналогичные действия, но не чувствительна к регистру.

С версии PHP 4.3.0 эти функции безопасны для обработки данных в двоичной форме.

Поиск в тексте

int strspn (string str1, string str2 [, int start [, int length]]) -- возвращает длину участка в начале строки str1, любой символ которого входит в строку str2.

Пример:

```
$var = strspn("42 is the answer, what is the question ...", "1234567890");
```

Переменная $var получит значение 2, т.к. «42» - это самый длинный участок строки, состоящий только из символов «1234567890».

Начиная с версии PHP 4.3.0, поддерживаются два необязательных аргумента, задающие начальную позицию (start) и длину участка строки (length), в котором производится поиск.

Пример:

```
echo strspn("foo", "o", 1, 2); // 2
```

Поиск в тексте

int strcspn (string str1, string str2 [, int start [, int length]]) -- возвращает длину участка в начале строки str1, который не содержит ни одного символа из строки str2.

Начиная с версии PHP 4.3.0, поддерживаются два необязательных аргумента, задающие начальную позицию (start) и длину участка строки (length), в котором производится поиск.

Функции strspn и strcspn безопасны для обработки данных в двоичной форме.

Поиск в тексте

string substr (string string, int start [, int length]) -- возвращает подстроку строки string длиной length, начинающегося со start символа по счёту.

Если start неотрицателен, возвращаемая подстрока начинается в позиции start от начала строки, считая от нуля.

Например, в строке 'abcdef', в позиции 0 находится символ 'a', в позиции 2 - символ 'c', и т.д.

Пример:

```
$rest = substr("abcdef", 1);     // возвращает bcdef
$rest = substr("abcdef", 1, 3); // возвращает bcd
$rest = substr("abcdef", 0, 4); // возвращает abcd
$rest = substr("abcdef", 0, 8); // возвращает abcdef
```

Поиск в тексте

Если start отрицательный, возвращаемая подстрока начинается со start символа с конца строки string.

Пример:

```
$rest = substr("abcdef", -1);    // возвращает f
$rest = substr("abcdef", -2);    // возвращает ef
$rest = substr("abcdef", -3, 1); // возвращает d
```

Если length положительный, возвращаемая строка будет не длиннее length символов. Если длина строки string меньше или равна start символов, возвращается FALSE.

Если length отрицательный, будет отброшено указанное этим аргументом число символов с конца строки string.

Если при этом позиция начала подстроки, определяемая аргументом start, находится в отброшенной части строки, возвращается пустая строка.

Пример:

```
$rest = substr("abcdef", 0, -1);  // возвращает abcde
$rest = substr("abcdef", 2, -1);  // возвращает cde
$rest = substr("abcdef", 4, -4);  // возвращает пустую строку
$rest = substr("abcdef", -3, -1); // возвращает de
```

Поиск в тексте

int substr_count (string haystack, string needle) -- возвращает число вхождений подстроки needle в строку haystack.

Поиск ведётся с учётом регистра символов.

Пример

```
echo substr_count("This is a test", "is"); // 2
```

Поиск в тексте

mixed count_chars (string string [, int mode]) – подсчитывает количество вхождений каждого из символов с ASII кодами в диапазоне (0..255) в строку string и возвращает эту информацию в различных форматах.

Необязательный аргумент mode по умолчанию равен 0.

В зависимости от его значения возвращается:

- 0 - массив, индексами которого являются ASCII коды, а значениями - число вхождений соответствующего символа;
- 1 - то же, что и для 0, но информация о символах с нулевым числом вхождений не включается в массив;
- 2 - то же, что и для 0, но в массив включается информация только о символах с нулевым числом вхождений;
- 3 - строка, состоящая из символов, которые входят в исходную строку хотя бы раз;
- 4 - строка, состоящая из символов, которые не входят в исходную строку.

Поиск в тексте

Пример:

```
$data = "Две в и одна с";
$result = count_chars($data, 0);
for ($i=0; $i < count($result); $i++)
 {
  if ($result[$i] != 0) echo chr($i)." встречается в строке $result[$i] раз(а).<br />";
 }
```

Выведет :

```
 встречается в строке 4 раз(а).
Д встречается в строке 1 раз(а).
а встречается в строке 1 раз(а).
в встречается в строке 2 раз(а).
д встречается в строке 1 раз(а).
е встречается в строке 1 раз(а).
и встречается в строке 1 раз(а).
н встречается в строке 1 раз(а).
о встречается в строке 1 раз(а).
с встречается в строке 1 раз(а).
```

Задание: напишите скрипт, собирающий информацию о том, какие символы и сколько раз использовались для написания строки. Функцию count_chars НЕ ИСПОЛЬЗОВАТЬ.

Поиск в тексте

mixed str_word_count (string string [, int format]) -- подсчитывает количество слов, входящих в строку string.

Если необязательный аргумент format не передан, возвращается целое число, равное количеству слов. В случае, если аргумент format передан, возвращается массив, содержимое которого зависит от значения этого аргумента:

1 - возвращается массив, содержащий все слова, входящие в строку string;

2 - возвращается массив, индексами которого являются позиции в строке, а значениями - соответствующие слова.

Эта функция считает словами последовательности алфавитных символов, возможно, включающие, но не начинающихся с «‘» и «-».

Поиск в тексте

Пример:

```
$str = "Hello friend, you're       looking       good today!";
$a   = str_word_count($str, 1);
$b   = str_word_count($str, 2);
$c   = str_word_count($str);
print_r($a);
print_r($b);
echo $c;
```

Вывод:

```
Array ( [0] => Hello [1] => friend [2] => you're [3] => looking [4] => good [5] => today )
Array ( [0] => Hello [6] => friend [14] => you're [29] => looking [46] => good [51] => today )
6
```

Определение длины строки

int strlen (string string) – возвращает длину строки string.

Пример:

```
$str = 'abcdef';
echo strlen($str); // 6
$str = ' ab cd ';
echo strlen($str); // 7
```

Внимание! Как уже было сказано ранее, PHP не различает строки и последовательности байт, т.о. данная функция, фактически, возвращает количество байт, занимаемых строкой. Для однобайтовых кодировок (windows-1251, cp-866 и т.п.) эти значения совпадают, но они НЕ совпадают для «мультибайтовых» кодировок (UTF-8 и т.п.)

Для работы с «мультибайтовыми кодировками» в PHP есть специальное расширение mbstring, подключение которого делает доступным набор строковых функций, имена которых начинаются с mb_ , а поведение аналогично рассматриваемым нами сейчас.

Преобразование «символ» <-> «код»

Для взаимных преобразований между символами таблицы ASCII и их кодами существуют две функции:

int ord (string string)

и

string chr (int ascii)

которые возвращают код (номер) символа по значению символа и значение символа по его коду (номеру) соответственно:

```
$str = chr(36);
echo ($str); // возвращает символ $

$str = ord('$');
echo($str); // возвращает 36
```

См. ASCII-таблицу в файле ascii.pdf

Trim-функции

string trim (string str [, string charlist]) – возвращает строку str с удалёнными из начала и конца строки «пробелами».

Если второй параметр не задан, trim() удаляет следующие символы:

- " " (ASCII 32 (0x20)), символ пробела;
- "\t" (ASCII 9 (0x09)), символ табуляции;
- "\n" (ASCII 10 (0x0A)), символ перевода строки;
- "\r" (ASCII 13 (0x0D)), символ возврата каретки;
- "\0" (ASCII 0 (0x00)), NUL-байт;
- "\x0B" (ASCII 11 (0x0B)), вертикальная табуляция.

Можно также задать список символов для удаления с помощью аргумента charlist.

Функции

string ltrim (string str [, string charlist])

и

string rtrim (string str [, string charlist])

удаляют «пробелы» в начале и конце строки соответственно.

Trim-функции

Пример:

```
$a="   Lalala text text      text     ||||||      ";
echo trim($a); // «Lalala text text      text     ||||||»
echo trim($a,"|"); // «Lalala text text      text»
echo ltrim($a); // «Lalala text text      text     ||||||      »
echo rtrim($a); // «   Lalala text text      text     ||||||»
```

Функцию удобно применять для удаления случайно введённых пробелов в начале и конце строки, а также для проверки того, что значение какого-то поля, которое пользователь хотел оставить пустым, не содержит пробелов.

Функции форматного вывода

Потребность в этих функциях возникает достаточно редко, т.к. они, фактически, являются «тяжёлым наследием языка C», однако иногда их применение может быть удобным. Впрочем, если вы хотите написать универсальное переносимое приложение, лучше обойтись без них.

int printf (string format [, mixed args]) – выводит строку, отформатированную в соответствии с аргументом format, возвращает длину выведенной строки;

string sprintf (string format [, mixed args]) – возвращает строку, созданную в соответствии с аргументом format;

mixed sscanf (string str, string format [, mixed & ...]) – разбирает строку в соответствии с аргументом format;

int vprintf (string format, array args) – выводит значения массива args, отформатированные в соответствии с аргументом format;

mixed fscanf (resource handle, string format [, mixed &...]) – похожа на sscanf(), но берёт данные для обработки из файла, ассоциируемого с handle, и интерпретирует их согласно формату format. Если в функцию переданы только два аргумента, обработанные значения будут возвращены в виде массива. В ином случае, если были переданы необязательные аргументы, функция вернёт количество присвоенных значений. Необязательные аргументы должны быть переданы по ссылке. Любое пустое пространство в строке формата эквивалентно любому пустому пространству во входящем потоке. Это означает, что даже табуляция \t в строке формата может быть сопоставлена одному символу пробела во входящем потоке данных.

Функции форматного вывода

Строка формата (format) состоит из директив: обычных символов (за исключением %), которые копируются в результирующую строку, и описатели преобразований, каждый из которых заменяется на один из параметров.

Каждый описатель преобразований состоит из знака процента (%), за которым следует один или более дополнительных элементов:

- Необязательный описатель заполнения, который определяет, какой символ будет использоваться для дополнения результата до необходимой длины.
- Необязательный описатель выравнивания, определяющий выравнивание влево или вправо. По умолчанию выравнивается вправо, «-» используется для выравнивания влево.
- Необязательное число, описатель ширины, определяющий минимальное число символов, которое будет содержать результат этого преобразования.

Функции форматного вывода

- Необязательный описатель точности, определяющий, сколько десятичных разрядов отображать для чисел с плавающей точкой. Имеет смысл только для числовых данных типа float. (Для форматирования чисел удобно также использовать функцию number_format().)
- Описатель типа, определяющий, как трактовать тип данных аргумента. Допустимые типы:
  - % - символ процента;
  - b - аргумент трактуется как целое и выводится в виде двоичного числа;
  - c - аргумент трактуется как целое и выводится в виде символа с соответствующим кодом ASCII;
  - d - аргумент трактуется как целое и выводится в виде десятичного числа со знаком;
  - e - аргумент трактуется как float и выводится в научной нотации (например 1.2e+2);
  - u - аргумент трактуется как целое и выводится в виде десятичного числа без знака;
  - f - аргумент трактуется как float и выводится в виде десятичного числа с плавающей точкой;
  - o - аргумент трактуется как целое и выводится в виде восьмеричного числа;
  - s - аргумент трактуется как строка;
  - x - аргумент трактуется как целое и выводится в виде шестнадцатеричного числа (в нижнем регистре букв);
  - X - аргумент трактуется как целое и выводится в виде шестнадцатеричного числа (в верхнем регистре букв).

Функции форматного вывода

Начиная с PHP 4.0.6 в строке формата поддерживается нумерация и изменение порядка параметров.

Пример:

```
$format = "There are %d monkeys in the %s";
printf($format, $num, $location);
```

Этот код выведет "There are 5 monkeys in the tree".

Теперь представьте, что строка формата содержится в отдельном файле, который потом будет переведен на другой язык, и мы переписываем её в таком виде:

```
$format = "The %s contains %d monkeys"; printf($format, $num, $location);
```

Появляется проблема: порядок описателей преобразования не соответствует порядку аргументов. Мы не хотим менять код, и нам нужно указать, какому аргументу соответствует тот или иной описатель преобразования:

```
$format = "The %2\$s contains %1\$d monkeys";
printf($format, $num, $location);
```

Нумерация аргументов имеет ещё одно применение: она позволят вывести один и тот же аргумент несколько раз без передачи функции дополнительных параметров:

```
$format = "The %2\$s contains %1\$d monkeys. That's a nice %2\$s full of %1\$d monkeys.";
printf($format, $num, $location);
```

Пример sprintf() с заполнением нулями:

```
$isodate = sprintf("%04d-%02d-%02d", $year, $month, $day);
```

Пример sprintf() на форматирование денежных величин:

```
$money1 = 68.75;
$money2 = 54.35;
$money = $money1 + $money2;
// echo $money выведет "123.1";
$formatted = sprintf("%01.2f", $money);
// echo $formatted выведет "123.10"
```

Функции форматного вывода

string number_format (float number [, int decimals])

и

string number_format (float number, int decimals, string dec_point, string thousands_sep)

возвращает отформатированное число number. Функция принимает один, два или четыре аргумента (не три!).

Если передан только один аргумент, number будет отформатирован без дробной части, но с запятой («,») между группами цифр по 3.

Если переданы два аргумента, number будет отформатирован с decimals знаками после точки («.») и с запятой («,») между группами цифр по 3.

Если переданы все четыре аргумента, number будет отформатирован с decimals знаками после точки и с разделителем между группами цифр по 3, при этом в качестве десятичной точки будет использован dec_point, а в качестве разделителя групп – thousands_sep. Используется только первый символ строки thousands_sep.

Например, при передаче «abcde» в качестве thousands_sep для форматирования числа 1000, number_format() возвращает 1a000.

Функции форматного вывода

Пример.

Во Франции обычно используются 2 знака после запятой (','), и пробел (' ') в качестве разделителя групп. Такое форматирование получается при использовании следующего кода:

```
$number = 1234.56;
// английский формат (по умолчанию)
$english_format_number = number_format($number);
// выведет 1,234
// французский формат
$nombre_format_francais = number_format($number, 2, ',', ' ');
// выведет 1 234,56
$number = 1234.5678;
// английский формат без разделителей групп
$english_format_number = number_format($number, 2, '.', '');
// выведет 1234.57
```

Задание для самоподготовки 1: написать скрипт, который будет вставлять в число «разделители тысяч», используя в качестве разделителя указанный символ.

Задание для самоподготовки 2: написать скрипт, выводящий т.н. «сумму прописью».

Функции конвертации кодировок

string convert_cyr_string (string str, string from, string to) – преобразует строку str из одной кириллической кодировки в другую.

Аргументы from и to задают входную и выходную кодировки соответственно, и состоят из одного символа.

Поддерживаются следующие кодировки:

- k - koi8-r;
- w - windows-1251;
- i - iso8859-5;
- d - x-cp866;
- m - x-mac-cyrillic.

Эта функция безопасна для обработки данных в двоичной форме.

Пример:

```
echo convert_cyr_string(“Привет!”,”w”,”K”);
```

Конвертация кодировок: юникод

Появление т.н. «мультибайтовых кодировок» и юникода с одной стороны упростило жизнь (появилась возможность представлять в Интернет тексты на различных языках с минимумом проблем), а с другой стороны – усложнила разработку ПО, т.к. сейчас приходится учитывать, какое ПО и как умеет (или не умеет) работать с той или иной «новомодной кодировкой».

Рекомендация. Если вы планируете делать сайт только на русском и английском – не изобретайте велосипед и используйте кодировку windows-1251. Если же вы хотите следовать последнему слову техники и моды, используйте юникод, о котором мы сейчас и поговорим.

Конвертация кодировок: юникод

Юникод (unicode) – стандарт кодирования символов, позволяющий представить знаки практически всех письменных языков.

Стандарт предложен в 1991 году некоммерческой организацией «Консорциум Юникода» («UnicodeR Consortium, Unicode Inc.»), объединяющей крупнейшие IT-корпорации.

Применение этого стандарта позволяет закодировать очень большое число символов из разных письменностей: в документах unicode могут соседствовать китайские иероглифы, математические символы, буквы греческого алфавита, латиницы и кириллицы, при этом становятся ненужными кодовые страницы.

Стандарт состоит из двух основных разделов:

- универсальный набор символов (UCS, Universal Character Set);
- семейство кодировок (UTF, Unicode Transformation Format).

Конвертация кодировок: юникод

Универсальный набор символов задаёт однозначное соответствие символов кодам – элементам кодового пространства, представляющим неотрицательные целые числа.

Семейство кодировок определяет машинное представление последовательности кодов UCS. Коды в стандарте Unicode разделены на несколько областей:

- Область с кодами от U+0000 до U+007F содержит символы набора ASCII с соответствующими кодами.
- Далее расположены области знаков различных письменностей, знаки пунктуации и технические символы.
- Часть кодов зарезервирована для использования в будущем.
- Под символы кириллицы выделены области знаков с кодами от U+0400 до U+052F, от U+2DE0 до U+2DFF, от U+A640 до U+A69F.

Конвертация кодировок: юникод

Предпосылки создания и развитие юникода

К концу 1980-х годов стандартом стали 8-битные символы, при этом существовало множество разных 8-битных кодировок и постоянно появлялись всё новые.

Это объяснялось как постоянным расширением круга поддерживаемых языков, так и стремлением создать кодировку, частично совместимую с какой-нибудь другой (характерный пример – появление альтернативной кодировки для русского языка, обусловленное эксплуатацией западных программ, созданных для кодировки CP437).

В результате появилась необходимость решения нескольких задач:

- Проблема «кракозябров» (отображения документов в неправильной кодировке): её можно было решить либо последовательным внедрением методов указания используемой кодировки, либо внедрением единой для всех кодировки.

Конвертация кодировок: юникод

- Проблема ограниченности набора символов: её можно было решить либо переключением шрифтов внутри документа, либо внедрением «широкой» кодировки. Переключение шрифтов издавна практиковалось в текстовых процессорах, причём часто использовались шрифты с нестандартной кодировкой, т.н. «dingbat fonts» – в итоге при попытке перенести документ в другую систему все нестандартные символы превращалось в «кракозябры».
- Проблема преобразования одной кодировки в другую: её можно было решить либо составлением таблиц перекодировки для каждой пары кодировок, либо использованием промежуточного преобразования в третью кодировку, включающую все символы всех кодировок.
- Проблема дублирования шрифтов: традиционно для каждой кодировки делался свой шрифт, даже если эти кодировки частично (или полностью) совпадали по набору символов: эту проблему можно было решить, делая «большие» шрифты, из которых потом выбираются нужные для данной кодировки символы – однако это требует создания единого реестра символов, чтобы определять, чему что соответствует.

Конвертация кодировок: юникод

Было признано необходимым создание единой «широкой» кодировки. Кодировки с переменной длиной символа, широко использующиеся в Восточной Азии, были признаны слишком сложными в использовании, поэтому было решено использовать символы фиксированной ширины. Использование 32-битных символов казалось слишком расточительным, поэтому было решено использовать 16-битные.

Таким образом, первая версия юникода представляла собой кодировку с фиксированным размером символа в 16 бит, то есть общее число кодов было 2^16 (65 536). Отсюда происходит практика обозначения символов четырьмя шестнадцатеричными цифрами (например, U+0410).

При этом в юникоде планировалось кодировать не все существующие символы, а только те, которые необходимы в повседневном обиходе. Редко используемые символы должны были размещаться в «области символов для частного использования» (Private Use Area), которая первоначально занимала коды U+D800:U+F8FF.

Конвертация кодировок: юникод

Чтобы использовать юникод также и в качестве промежуточного звена при преобразовании разных кодировок друг в друга, в него включили все символы, представленные во всех более-менее известных кодировках.

В дальнейшем, однако, было принято решение кодировать все символы и в связи с этим значительно расширить кодовую область.

Одновременно с этим, коды символов стали рассматриваться не как 16-битные значения, а как абстрактные числа, которые в компьютере могут представляться множеством разных способов.

Поскольку в ряде компьютерных систем (например, Windows NT) уже были реализованы 16-битные символы, было решено всё наиболее важное кодировать только в пределах первых 65 536 позиций (т.н. «Basic Multilingual Plane, BMP»). Остальное пространство используется для «дополнительных символов» (Supplementary Characters): систем письма вымерших языков или очень редко используемых китайских иероглифов, математических и музыкальных символов.

Конвертация кодировок: юникод

Для совместимости со старыми 16-битными системами была изобретена система UTF-16, где первые 65 536 позиций отображаются непосредственно как 16-битные числа, а остальные представляются в виде «суррогатных пар» (первый элемент пары из области U+D800:U+DBFF, второй элемент пары из области U+DC00:DFFF).

Для суррогатных пар была использована часть кодового пространства, ранее отведённого для «символов для частного использования».

Поскольку в UTF-16 можно отобразить только 2^20+2^16 (1 114 112) символов, то это и было выбрано в качестве окончательной величины кодового пространства юникода.

Хотя кодовая область юникода была расширена за пределы 2^16 уже в версии 2.0, первые символы в «верхней» области были размещены только в версии 3.1.

Конвертация кодировок: юникод

Версии юникода

По мере изменения и пополнения таблицы символов системы юникода и выхода новых версий этой системы, – а эта работа ведётся постоянно.

Система юникод существует в общей сложности в следующих версиях:

- 1.1 (соответствует стандарту ISO/IEC 10646-1:1993);
- 2.0, 2.1 (тот же стандарт ISO/IEC 10646-1:1993 плюс дополнения: «Amendments» с 1-го по 7-е и «Technical Corrigenda» 1 и 2);
- 3.0 (стандарт ISO/IEC 10646-1:2000);
- 3.2 (стандарт 2002 года);
- 4.0 (стандарт 2003 года);
- 4.01 (стандарт 2004 года);
- 4.1 (стандарт 2005 года);
- 5.0 (стандарт 2006 года);
- 5.1 (стандарт 2008 года).

Конвертация кодировок: юникод

Способы представления

Юникод имеет несколько форм представления (Unicode Transformation Format, UTF): UTF-8, UTF-16 (UTF-16BE, UTF-16LE) и UTF-32 (UTF-32BE, UTF-32LE).

Была разработана также форма представления UTF-7 для передачи по семибитным каналам, но из-за несовместимости с ASCII она не получила распространения и не включена в стандарт.

Чаще всего вам придётся сталкиваться с UTF-8.

UTF-8 – это представление юникода, обеспечивающее наилучшую совместимость со старыми системами, использовавшими 8-битные символы.

Текст, состоящий только из символов с номером меньше 128, при записи в UTF-8 превращается в обычный текст ASCII. И наоборот, в тексте UTF-8 любой байт со значением меньше 128 изображает символ ASCII с тем же кодом.

Остальные символы юникода изображаются последовательностями длиной от 2 до 6 байт, в которых первый байт всегда имеет вид 11xxxxxx, а остальные - 10xxxxxx.

Конвертация кодировок: юникод

Порядок байтов

В потоке данных UTF-16 старший байт может записываться либо перед младшим (UTF-16 Big Endian), либо после младшего (UTF-16 Little Endian).

Иногда кодировку юникода Big Endian (UTF-16BE) называют юникодом с обратным порядком байтов.

Аналогично существует два варианта четырёхбайтной кодировки – UTF-32BE и UTF-32LE.

Для определения формата представления юникода в текстовом файле используется приём, по которому в начале текста записывается символ U+FEFF (неразрывный пробел с нулевой шириной), также именуемый меткой порядка байтов (Byte Order Mark, BOM).

Этот способ позволяет различать UTF-16LE и UTF-16BE, поскольку символа U+FFFE не существует. Также он иногда применяется для обозначения формата UTF-8, хотя к этому формату и неприменимо понятие порядка байтов.

Конвертация кодировок: юникод

Файлы, следующие этому соглашению, начинаются с таких последовательностей байтов:

UTF-8 = EF BB BF

UTF-16BE = FE FF

UTF-16LE = FF FE

UTF-32BE = 00 00 FE FF

UTF-32LE = FF FE 00 00

К сожалению, этот способ не позволяет надёжно различать UTF-16LE и UTF-32LE, поскольку символ U+0000 допускается юникодом (хотя реальные тексты редко начинаются с него).

Конвертация кодировок: юникод

Внимание! ОЧЕНЬ ВАЖНО!

В posix-совместимых операционных системах повсеместно применяются текстовые конфигурационные файлы. Боже вас упаси использовать для их редактирования редакторы, работающие в юникоде.

Такое необдуманное действие приводит к появлению в начале файла «признака формата юникода U+FEFF», который, естественно, трактуется тем ПО, чей конфигурационный файл вы только что покалечили, самым непредсказуемым образом.

Конвертация кодировок: юникод

Проблемы юникода

- Некоторые системы письма всё ещё не представлены должным образом в юникоде. Изображение «длинных» надстрочных символов, простирающихся над несколькими буквами, как например, в церковнославянском языке, пока не реализовано.
- Тексты на китайском, корейском и японском языке имеют традиционное написание сверху вниз, начиная с правого верхнего угла. Переключение горизонтального и вертикального написания для этих языков не предусмотрено в юникоде – это должно осуществляться средствами языков разметки или внутренними механизмами текстовых процессоров.
- Первоначальная версия юникода предполагала наличие большого количества готовых символов, в последующем было отдано предпочтение сочетанию букв с диакритическими модифицирующими знаками (Combining diacritics). Например, русские буквы «ё» (U+0401) и «й» (U+0419) существуют в виде монолитных символов, хотя могут быть представлены и набором базового символа с последующим диакритическим знаком, то есть в составной форме (Decomposed): е+«две точки вверху» (U+0415 U+0308), и+«дуга вверху» (U+0418 U+0306). В то же время множество символов из языков с алфавитами на основе кириллицы не имеют монолитных форм.

Конвертация кодировок: юникод

- Юникод предусматривает возможность разных начертаний одного и того же символа в зависимости от языка. Так, китайские иероглифы могут иметь разные начертания в китайском, японском (кандзи) и корейском (ханчча), но при этом в юникоде обозначаться одним и тем же символом (так называемая CJK-унификация), хотя упрощённые и полные иероглифы всё же имеют разные коды. Часто возникают накладки, когда, например, японский текст выглядит «по-китайски». Аналогично, русский и сербский языки используют разное начертание курсивных букв *п* и *т* (в сербском они выглядят как *и* и *ш*). Поэтому нужно следить, чтобы текст всегда был правильно помечен как относящийся к тому или другому языку.
- Файлы с текстом в юникоде занимают больше места в памяти, так как один символ кодируется не одним байтом, как в различных национальных кодировках, а последовательностью байтов (исключение составляет UTF-8 для языков, алфавит которых укладывается в ASCII). Однако с увеличением мощности компьютерных систем и удешевлением памяти и дискового пространства эта проблема становится всё менее существенной.
- Хотя поддержка юникода реализована в наиболее распространённых операционных системах, не всё прикладное программное обеспечение поддерживает корректную работу с ним. В частности, не всегда обрабатываются метки BOM и плохо поддерживаются диакритические символы. Проблема является временной.

Функции конвертации кодировок

В PHP для конвертации текста между кодировками ASCII и UTF-8 используются функции

string utf8_encode (string data)

и

string utf8_decode (string data)

Однако, как показывает практика, значительно меньше проблем возникает, если использовать функцию расширения mbstring:

string mb_convert_encoding (string str, string to_encoding [, mixed from_encoding])

Пример:

```
$str = mb_convert_encoding($str, "UTF-8", “CP1251");
```

Функции работы с блоками текста

string substr_replace (string string, string replacement, int start [, int length]) – заменяет часть строки string начинающуюся с символа с порядковым номером start и длиной length строкой replacement и возвращает результат в виде строки.

Если start – положительное число, замена начинается с символа с порядковым номером start.

Если start – отрицательное число, замена начинается с символа с порядковым номером start, считая от конца строки.

Если аргумент length – положительное число, то он определяет длину заменяемой подстроки.

Если аргумент length – отрицательное число, то он определяет количество символов от конца строки, на котором заканчивается замена. Этот аргумент необязателен и по умолчанию равен длине строки.

Функции работы с блоками текста

Пример:

```
$var = 'ABCDEFGH:/MNRPQR/';
echo "Оригинал: $var";
// обе следующих строки заменяют всю строку $var на 'bob'
echo substr_replace($var, 'bob', 0);
echo substr_replace($var, 'bob', 0, strlen($var));
// вставляет 'bob' в начало $var
echo substr_replace($var, 'bob', 0, 0);
// обе следующих строки заменяют 'MNRPQR' в $var на 'bob‘
echo substr_replace($var, 'bob', 10, -1);
echo substr_replace($var, 'bob', -7, -1);
// удаляет 'MNRPQR' из $var
echo substr_replace($var, '', 10, -1);
```

Эта функция безопасна для обработки данных в двоичной форме.

Функции работы с блоками текста

string wordwrap (string str [, int width [, string break [, boolean cut]]]) – возвращает строку str с переносом в столбце с номером, заданном аргументом width.

Строка разбивется с помощью аргумента break.

Аргументы width и break необязательны и по умолчанию равны 75 и \n соответственно.

Если аргумент cut установлен в 1, разрыв делается точно в заданной колонке. Поэтому если исходная строка содержит слово длиннее, чем заданная длина, в этом случае слово будет разорвано. Аргумент cut был добавлен в PHP 4.0.3.

Рассмотрим пример…

Функции работы с блоками текста

Пример:

```
$a="Это длинный текст. Это длинный текст. Это длинный текст. Это длинный текст. Это длинный текст. Это длинный текст.";
echo wordwrap($a); echo “<hr />”;
echo wordwrap($a,10); echo “<hr />”;
echo wordwrap($a,5,"<br />"); echo “<hr />”;
echo wordwrap($a,5,"<br />",1);
```

Это длинный текст. Это длинный текст. Это длинный текст. Это длинный текст.
Это длинный текст. Это длинный текст.**<hr />**Это
длинный
текст.
Это
длинный
…
длинный
текст.
Это
длинный
текст.**<hr />**Это
длинный
текст.
Это
длинный
текст.
Это
длинный
текст.
Это
длинный
текст.
Это
длинный
текст.
Это
длинный
текст.**<hr />**Это
длинн
ый
текст
. Это
длинн
ый
текст
. Это
длинн
ый
текст
. Это
длинн
ый
текст
. Это
длинн
ый
текст
. Это
длинн
ый
текст

Функции работы с блоками текста

mixed str_replace (mixed search, mixed replace, mixed subject [, int &count]) – возвращает строку или массив subject, в котором все вхождения search заменены на replace.

Если не нужны сложные правила поиска/замены, использование этой функции предпочтительнее функций, работающих с регулярными выражениями.

Если subject – массив, поиск и замена производится в каждом элементе этого массива, и возвращается также массив.

Если и search, и replace – массивы, то str_replace() использует все значения массива search и соответствующие значения массива replace для поиска и замены в subject.

Если в массиве replace меньше элементов, чем в search, в качестве строки замены для оставшихся значений будет использована пустая строка.

Если search – массив, а replace – строка, то replace будет использована как строка замены для каждого элемента массива search.

Функции работы с блоками текста

Пример:

```
// присваивает <body text='black'>
$bodytag = str_replace("%body%", "black", "<body text='%body%'>");
// присваивает: Hll Wrld f PHP
$tokill = array("а", "б", "в", "г", "д", "12", "135");
$killed = str_replace($tokill, "", "Это был пример 12345676 текста 1513578");
// присваивает: Это ыл пример 345676 текст 1578
$phrase  = "Не читайте советских газет перед завтраком";
$old_words = array("читатйе", "перед", "завтраком");
$new_words   = array("выбрасывайте", "в", "окно");
$newphrase = str_replace($old_words, $new_words, $phrase);
// Не выбрасывайте советских газет в окно
// начиная с версии 5.0.0 доступен аргумент count
$str = str_replace("нн", "!!", "стеклянный, оловянный", $count);
echo $count; // 2
```

Эта функция безопасна для обработки данных в двоичной форме.

Начиная с PHP 5.0.0 количество произведённых замен может быть получено в необязательном аргументе count, который передается по ссылке. В версиях до PHP 5.0.0 этот аргумент недоступен.

Для проведения замены без учёта регистра существует функция str_ireplace() (начиная с PHP 5.0.0).

Функции работы с блоками текста

При работе с формами или SQL-запросами периодически возникает необходимость заэкранировать некоторые символы, а потом вернуть исходное состояние текста. Это позволяют сделать функции:

string addslashes (string str)

и

string stripslashes (string str)

addslashes – возвращает сроку str, в которой перед каждым спецсимволом добавлен обратный слеш \ («бэкслеш»).

Экранируются: одиночная кавычка ', дойная кавычка ", обратный слэш \ и NUL (байт NULL).

stripslashes – удаляет экранирующие бэкслэши, поставленные функцией addslashes.

Функции работы с блоками текста

Пример:

```
$str = "Is your name O'reilly?";
// выводит: Is your name O\'reilly?
echo addslashes($str);

$str = "Is your name O\'reilly?";
// выводит: Is your name O'reilly?
echo stripslashes($str);
```

Внимание! Не полагайтесь на функцию addslashes() для защиты от атак вида code-injection и SQL-injection. Какие функции следует использовать, мы рассмотрим в теме, посвящённой безопасности программ на PHP.

Функции работы с блоками текста

string strrev (string string) – возвращает строку string, в которой порядок символов изменён на обратный («инвертирует» строку).

Пример:

```
echo strrev("Hello world!");
// !dlrow olleH
```

Задача для самоподготовки: написать скрипт, выводящий все слова-палиндромы (читаются одинаково в обоих направлениях, например: «как», «оно»), содержащиеся в тексте.

Функции работы с блоками текста

int crc32 (string str) – вычисляет контрольную сумму по алгоритму CRC32 для строки str.

Обычно используется для контроля правильности передачи данных.

В PHP целые числа имеют знак, и эта функция может возвращать отрицательные числа.

Пример:

```
$checksum = crc32("The quick brown fox jumped over the lazy dog.");
echo $checksum;
// -2103228862
```

Функции работы с блоками текста

string md5 (string str [, bool raw_output]) – вычисляет MD5 хэш строки str, используя алгоритм MD5 (RSA Data Security, Inc.) и возвращает этот хэш. Обычно используется для хэширования паролей.

Хэш представляет собой 32-значное шестнадцатеричное число.

Если необязательный аргумент raw_output имеет значение TRUE, то возвращается бинарная строка из 16 символов.

Необязательный аргумент raw_output был добавлен в PHP 5.0.0 и по умолчанию равен FALSE.

Пример:

```
$str = 'apple';
if (md5($str) === '1f3870be274f6c49b3e31a0c6728957f')
{
 echo "Would you like a green or red apple?";
}
```

Задание для самостоятельной проработки: посмотрите в руководстве по PHP описание функций md5_file(), sha1() и crypt().

Функции изменения регистра строки

Для изменения регистра символов строки в PHP существует четыре основные функции: strtolower(), strtoupper(), ucfirst(), ucwords().

string strtolower (string str) – возвращает строку string, в которой все буквенные символы переведены в нижний регистр.

Пример:

```
$str = "Mary Had A Little Cat and She LOVED It So";
$str = strtolower($str);
echo $str; // выводит: mary had a little cat and she loved it so
```

Для мультибайтовых строк следует использовать mb_strtolower().

Функции изменения регистра строки

string strtoupper (string string) – возвращает строку string, в которой все буквенные символы переведены в верхний регистр.

Пример:

```
$str = "Mary Had A Little Cat and She LOVED It So";
$str = strtoupper($str);
echo $str; // выводит: MARY HAD A LITTLE CAT AND SHE LOVED IT SO
```

Для мультибайтовых строк следует использовать mb_strtoupper().

Функции изменения регистра строки

string ucfirst (string str) – возвращает строку string, в которой первый символ переведён в верхний регистр.

Пример:

```
$s = 'hello world!';
$s = ucfirst($s); // Hello world!
```

string ucwords (string str) – возвращает строку string, в которой первый символ каждого слова переведён в верхний регистр. Эта функция считает словами последовательности символов, разделённых пробелом, переводом строки, возвратом каретки, горизонтальной или вертикальной табуляцией.

Пример:

```
$s = 'hello world!';
$s = ucwords($s); // Hello World!
```

Функции работы с URL

Поскольку URL представляет собой строку, сейчас мы рассмотрим строковые функции, позволяющие выполнять обработку URL.

mixed parse_url (string url [, int component]) – возвращает массив элементов URL или отдельный фрагмент, если указан второй параметр. В случае, если string является совершенно нераспознаваемым в качестве URL, эта функция возвращает FALSE.

Возможные значения параметра component: PHP_URL_SCHEME, PHP_URL_HOST, PHP_URL_PORT, PHP_URL_USER, PHP_URL_PASS, PHP_URL_PATH, PHP_URL_QUERY, PHP_URL_FRAGMENT. Их суть будет понятна из примера, который мы сейчас рассмотрим.

Внимание! Эта функция НЕ предназначена для проверки корректности URL.

Функции работы с URL

Пример:

```
$url="https://user1:password1@www.site.com:8080/dir1/dir2/page.php?a=12&b=OK&c=yes#chapter12";
print_r ( parse_url ( $url ) );
```

Выведет:

```
Array (
[scheme] => https
[host] => www.site.com
[port] => 8080
[user] => user1
[pass] => password1
[path] => /dir1/dir2/page.php
[query] => a=12&b=OK&c=yes
[fragment] => chapter12 )
```

Если какого-то элемента в URL нет, соответствующий элемент массива будет отсутствовать.

Функции работы с URL

void parse_str (string str [, array arr]) – разбирает строку str, которая должна иметь формат строки запроса URL, и присваивает значения переменным (в текущей области видимости; этот вопрос мы рассмотрим в теме, посвящённой функциям), если не передан второй аргумент arr. Если он передан, значения будут сохранены в нём как элементы массива.

Пример:

```
$str = "first=value&arr[]=foo+bar&arr[]=baz";
parse_str($str);
echo $first;  // value
echo $arr[0]; // foo bar
echo $arr[1]; // baz
parse_str($str, $output);
echo $output['first'];  // value
echo $output['arr'][0]; // foo bar
echo $output['arr'][1]; // baz
```

Функции работы с URL

Для того, чтобы передать в виде URL данные, изначально для этого не предназначенные (допустим, нам нужно указать в виде значения переменной часть текста), используются функции: rawurlencode(), rawurldecode(), urlencode(), urldecode().

string rawurlencode (string str) – преобразует все неалфавитно-цифровые символы (за исключением - _ .) в последовательности вида %xx, где xx – двухразрядное число в 16-теричной системе счисления.

string rawurldecode (string str) – выполняет обратное преобразование.

Эти функции удобно применять для того, чтобы часть URL не была проинтерпретирована как служебные символы, а также для того, чтобы защитить передаваемый текст от автоматической перекодировки.

Функция работает согласно стандарту RFC 1738 (см. http://www.ietf.org/rfc.html).

Функции работы с URL

string urlencode (string str) – действует аналогично функции rawurlencode() за исключением того, что пробелы преобразует в символ + .

Т.о. эта функция, фактически, формирует данные в формате, в каком они передаются из формы, у которой enctype установлено в application/x-www-form-urlencoded.

string urldecode (string str) – выполняет обратное преобразование.

А сейчас – примеры…

Функции работы с URL

Пример:

```
$url="page.php?name=Дункан Мак-Лауд&age=478&pl_of_birth=горы Шотландии";

echo rawurlencode($url);
```

Выведет:

page.php%3Fname%3D%84%E3%AD%AA%A0%AD%20%8C%A0%AA-%8B%A0%E3%A4%26age%3D478%26pl_of_birth%3D%A3%AE%E0%EB%20%98%AE%E2%AB%A0%AD%A4%A8%A8

```
echo urlencode($url);
```

Выведет:

page.php%3Fname%3D%84%E3%AD%AA%A0%AD+%8C%A0%AA-%8B%A0%E3%A4%26age%3D478%26pl_of_birth%3D%A3%AE%E0%EB+%98%AE%E2%AB%A0%AD%A4%A8%A8

Обратите внимание: символы, действительно являющиеся служебными для URL, тоже оказались преобразованными. Поэтому обработке данными функциями следует подвергать ТОЛЬКО значения переменных, переданных в URL, а не весь URL целиком:

```
$url="page.php?name=".rawurlencode("Дункан Мак-Лауд")."&age=".rawurlencode("478")."&pl_of_birth=".rawurlencode("горы Шотландии");
```

Понятие «локали» и соотв. функции

Локаль (locale) – набор параметров, включая набор символов, язык пользователя, страну, часовой пояс, а также другие предустановки, которые пользователь ожидает видеть в пользовательском интерфейсе, и ПО использует в своих вычислениях.

Идентификатор локали может быть определён несколькими способами:

Идентификатор локали **в Win32 API** называется LCID и представляет собой число – например, 1033 для английского языка (США) или 1049 для русского (Россия).

В XML, Microsoft .NET и Java, а также в других окружениях, поддерживающих Unicode, локаль обозначается согласно RFC 3066 или его преемников. Обычно используются коды ISO 639 и ISO 3166-1 alpha-2. В частности, в .NET используется строка «ru-RU» для России и «en-US» для США.

В Unix, GNU/Linux и других платформах POSIX локаль определяется подобно RFC 3066, но варианты локали определяются по-другому, набор символов также включается в идентификатор. Таким образом, он имеет следующий вид:

[language[_territory][.codeset][@modifier]], например ru_RU.KOI8-R.

Понятие «локали» и соотв. функции

В PHP для работы с локалью используются функции setlocale() и localeconv().

Внимание! Несмотря на всю кажущуюся мощность локали и её удобство, на практике с ней бывает огромное количество проблем. Лучший выход из данной ситуации – **НЕ использовать локаль**, а выполнять все необходимые преобразования (время, дата, форматы записи чисел, валют и т.д.) вручную. Это даёт гарантию работоспособности и переносимости кода.

Итак, рассмотрим функции…

Понятие «локали» и соотв. функции

string setlocale (mixed category, string locale [, string ...])

или

string setlocale (mixed category, array locale)

Данная функция устанавливает локаль. Параметры:

category – строка или константа, задающая категорию функций, на которые будет влиять установка локали:

- LC_ALL – все функции;
- LC_COLLATE – функция сравнения строк strcoll() (используется редко);
- LC_CTYPE – функции преобразования строк, например strtoupper();
- LC_MONETARY – функция localeconv();
- LC_NUMERIC – задаёт символ «десятичной точки»;
- LC_TIME – форматирование дат функцией strftime() (используется редко).

Понятие «локали» и соотв. функции

Если в качестве locale передан NULL или 0, локаль изменена не будет, а будет возвращено текущее значение локали.

Если в качестве locale передан массив, или после этого аргумента следуют дополнительные аргументы, функция будет использовать элементы массива или аргументы по порядку в качестве имён локали до тех пор, пока установка локали не будет успешной.

Это удобно, если одна и та же локаль имеет разное имя в различных системах, или необходимая локаль может отсутствовать в системе.

Функция setlocale() возвращает имя вновь установленной локали или FALSE, если система не поддерживает установку локали, указанная локаль не существует или передано недопустимое имя категории.

Имена локалей и категорий описаны в RFC 1766 и ISO 639.

Замечание: возвращаемое функцией setlocale() значение зависит от системы – это значение, возвращаемое системной функцией setlocale.

Понятие «локали» и соотв. функции

Пример:

```
// установка голландской локали
setlocale(LC_ALL, 'nl_NL');
echo strftime("%A %e %B %Y", mktime(0, 0, 0, 12, 22, 1978));
// выводит: vrijdag 22 december 1978

// попытка использовать различные локали для  немецкого языка
$loc_de = setlocale(LC_ALL, 'de_DE@euro', 'de_DE', 'de', 'ge');
echo "На этой системе немецкая локаль имеет имя '$loc_de‘ ";

// установка голландской локали в Windows
setlocale(LC_ALL, 'nl_NL');
echo strftime("%A %d %B %Y", mktime(0, 0, 0, 12, 22, 1978));
// выводит: vrijdag 22 december 1978

// попытка использовать различные локали для  немецкого языка
$loc_de = setlocale(LC_ALL, 'de_DE@euro', 'de_DE', 'deu_deu');
echo "На этой системе немецкая локаль имеет имя '$loc_de‘ ";
```

Понятие «локали» и соотв. функции

array localeconv (void) – возвращает ассоциативный массив с информацией о числовых и денежных форматах в текущей локали.

Возвращаемый массив содержит следующие элементы…

Понятие «локали» и соотв. функции

Элемент	Описание
decimal_point	Символ десятичной точки
thousands_sep	Разделитель групп
grouping	Массив, содержащий количества цифр в группах для числовых данных
int_curr_symbol	Международное обозначение валюты (например RUR)
currency_symbol	Национальное обозначение валюты (например р.)
mon_decimal_point	Символ десятичной точки в денежном формате
mon_thousands_sep	Разделитель групп в денежном формате
mon_grouping	Массив, содержащий количества цифр в группах для денежных данных
positive_sign	Знак для положительных чисел
negative_sign	Знак для отрицательных чисел
int_frac_digits	Число разрядов после точки (международное)
frac_digits	Число разрядов после точки (национальное)
p_cs_precedes	**TRUE** если currency_symbol записывется перед положительным значением, иначе **FALSE**
p_sep_by_space	**TRUE** если currency_symbol отделяется от положительного значения пробелом, иначе **FALSE**
n_cs_precedes	**TRUE** если currency_symbol записывется перед отрицательным значением, иначе **FALSE**
n_sep_by_space	**TRUE** если currency_symbol отделяется от отрицательного значения пробелом, иначе **FALSE**
p_sign_posn	Для положительных чисел 0 - Число и обозначение валюты заключаются в скобки; 1 - Знак записывается перед числом и обозначением валюты; 2 - Знак записывается после числа и обозначения валюты; 3 - Знак записывается перед обозначением валюты; 4 - Знак записывается после обозначения валюты.
n_sign_posn	Для отрицательных чисел 0 - Число и обозначение валюты заключаются в скобки; 1 - Знак записывается перед числом и обозначением валюты; 2 - Знак записывается после числа и обозначения валюты; 3 - Знак записывается перед обозначением валюты; 4 - Знак записывается после обозначения валюты.

Понятие «локали» и соотв. функции

Пример:

```
setlocale(LC_ALL, "ru_RU");
$locale_info = localeconv();

echo "int_curr_symbol:   {$locale_info["int_curr_symbol"]} <br />";
echo "currency_symbol:   {$locale_info["currency_symbol"]} <br />";
echo "mon_decimal_point: {$locale_info["mon_decimal_point"]} <br />";
echo "mon_thousands_sep: {$locale_info["mon_thousands_sep"]} <br />";
echo "positive_sign:     {$locale_info["positive_sign"]} <br />";
echo "negative_sign:     {$locale_info["negative_sign"]} <br />";
echo "int_frac_digits:   {$locale_info["int_frac_digits"]} <br />";
echo "frac_digits:       {$locale_info["frac_digits"]} <br />";
echo "p_cs_precedes:     {$locale_info["p_cs_precedes"]} <br />";
echo "p_sep_by_space:    {$locale_info["p_sep_by_space"]} <br />";
echo "n_cs_precedes:     {$locale_info["n_cs_precedes"]} <br />";
echo "n_sep_by_space:    {$locale_info["n_sep_by_space"]} <br />";
echo "p_sign_posn:       {$locale_info["p_sign_posn"]} <br />";
echo "n_sign_posn:       {$locale_info["n_sign_posn"]} <br />";
```

Функции сравнения строк

Несмотря на то, что в PHP операторы сравнения применимы и для строк, существует некоторый набор функций, который иногда полезно использовать для решения специфических задач:

int strcoll (string str1, string str2) – сравнивает строки, учитывая текущую локаль. Возвращает отрицательное число, если str1 меньше, чем str2; положительное число, если str1 больше, чем str2, и 0 если строки равны.

Функция strcoll() при сравнении использует установки текущей локали. Если установлена локаль C или POSIX, эта функция аналогична strcmp().

Обратите внимание:

1) эта функция учитывает регистр символов;

2) В отличие от strcmp() НЕ безопасна для обработки данных в двоичной форме.

Функции сравнения строк

int strcmp (string str1, string str2) – возвращает отрицательное число, если str1 меньше, чем str2; положительное число, если str1 больше, чем str2, и 0 если строки равны. **Учитывает регистр символов.**

int strcasecmp (string str1, string str2) – в возвращает отрицательное число, если str1 меньше, чем str2; положительное число, если str1 больше, чем str2, и 0 если строки равны. **Не учитывает регистр символов.**

Пример:

```
$var1 = "Hello";
$var2 = "hello";
if (strcasecmp($var1, $var2) == 0) echo '$var1 равно $var2’;
```

Функции сравнения строк

int strncmp (string str1, string str2, int len) – эта функция подобна strcmp(), за исключением того, что можно указать максимальное количество символов в обеих строках, которые будут участвовать в сравнении (len).

Возвращает отрицательное число, если str1 меньше, чем str2; положительное число, если str1 больше, чем str2, и 0 если строки равны. **Учитывает регистр символов.**

int strncasecmp (string str1, string str2, int len) – эта функция подобна strcasecmp(), за исключением того, что можно указать максимальное количество символов в обеих строках, которые будут участвовать в сравнении (len).

Возвращает отрицательное число, если str1 меньше, чем str2; положительное число, если str1 больше, чем str2, и 0 если строки равны. **Не учитывает регистр символов.**

Функции сравнения строк

int strnatcmp (string str1, string str2) – реализует алгоритм сравнения, упорядочивающий алфавитно-цифровые строки подобно тому, как это сделал бы человек.

Пример, показывающий отличие этого алгоритма от обычных функций сравнения:

```
$arr1 = $arr2 = array("img12.png", "img10.png", "img2.png", "img1.png");
echo "Стандартный алгоритм сравнения";
usort($arr1, "strcmp");
print_r($arr1);
echo “<br />Алгоритм ‘естественного упорядочения’ ";
usort($arr2, "strnatcmp");
print_r($arr2);
```

Выводит:

Стандартный алгоритм сравнения: Array ([0] => img1.png [1] => img10.png [2] => img12.png [3] => img2.png)

Алгоритм "естественного упорядочения" Array ([0] => img1.png [1] => img2.png [2] => img10.png [3] => img12.png)

Подобно другим функциям сравнения строк, strnatcmp() возвращает отрицательное число, если str1 меньше, чем str2; положительное число если, str1 больше, чем str2, и 0 если строки равны. **Учитывает регистр символов.**

Функции сравнения строк

int strnatcasecmp (string str1, string str2) – реализует алгоритм сравнения, упорядочивающий алфавитно-цифровые строки подобно тому, как это сделал бы человек.

Эта функция подобна strnatcmp(), за исключением того, что strnatcasecmp() **не учитывает регистр символов**.

Подобно другим функциям сравнения строк, strnatcasecmp() возвращает отрицательное число, если str1 меньше, чем str2; положительное число, если str1 больше, чем str2, и 0 если строки равны.