PHP マニュアル
プログラミング基礎 for Windows, Linux
関数編

1.マルチバイト文字列関数
・部分文字列を取得する
・特定の文字位置を検索する
・文字列を変換する
・文字コードを変換する

2.マルチバイト正規表現関数
・正規表現関数で利用できるパターン識別子
・正規表現による検索
・正規表現による置換
・正規表現による分割

3.日付・時刻関数
・タイムスタンプ
・time、microtime 関数
・mktime 関数
・strtotime 関数
・日付データを特定のフォーマットに整形する
・日付の妥当性を検証する

4.ディレクトリ関数＆ファイルシステム関数
・例）ディレクトリ情報を一覧表示する
・例）テキストファイルへの書き込み
・例）テキストファイルの読み込み

1.マルチバイト文字列関数

マルチバイト文字とは、その名のとおり、2バイト以上のデータで表現される文字のことを言い、ひらがな、カタカナ、漢字など、いわゆる全角文字はすべてマルチバイト文字です。これに対して、半角文字など1バイトで表現される文字のことをシングルバイト文字と言います。たかだか表現の違いと思われるかもしれませんが、実はマルチバイト文字の処理にはなにかと厄介なことが少なくありません。たとえば、シングルバイト文字とマルチバイト文字が混在している場合は、マルチバイト文字とシングルバイト文字とをそれぞれ識別させなければいけないからです。

マルチバイト文字の世界では、文字コードの識別も重要です。マルチバイト文字に対応した文字コードとしては、Windows 系 OS の標準である Shift_JIS（SJIS）、Unix 系 OS で使用される（EUC-JP）、電子メールで標準的に使用される JIS、はたまた、XML や Java の標準コードとして採用されている UTF-8などさまざまです。しかも、これらの文字コードの間にはほとんど互換性はないので、もし Shift_JIS で書かれた情報を EUC-JP として参照してしまうと、正しく文字が処理されない。いわゆる文字化けが起きてしまいます。我々が日常的に使用しているマルチバイト文字ですが、このように、つくる側の立場から見ると非常に厄介な存在です。さて、PHP ではこのようなマルチバイト文字列を含んだデータを扱うために、専用のマルチバイト文字列関数が用意されています。マルチバイト文字列関数を利用することで、マルチバイト特有の煩雑な操作をほとんど意識する必要がなく処理できるようになります。

マルチバイト文字列関数を使用するには、php.ini に次の記述のコメントアウトを外してマルチバイト文字列関数を有効にしておく必要があります（Windows の場合）。Linux 環境では、PHP 本体の configure 時に--enable-mbstring オプションを付加することで静的にマルチバイト文字列関数が組み込まれますので特別な設定などは必要ありません。

```
623 extension=php_mbstring.dll↵
```

➢ 部分文字列を取得する

オリジナルの文字列から部分的な文字列を抽出するのは、mb_substr 関数の役割です。mb_substr 関数の一般的な構文は次のとおりです。

mb_substr 関数
string mb_substr (string str, int start [, int length [, string encoding]])
str : 任意の文字列
start : 開始位置
length : 抽出する文字数
encoding : 文字コード名

引数 encoding を省略した場合は、php.ini の mbstring.internal_encoding パラメータで指定した内部文字エンコーディングが使用されます。たとえば「Windows はオペレーティングシステムです」という文字列からオペレーティングシステムだけを取り出してみましょう。

substr.php
```
<?php
$str='Windows はオペレーティングシステムです';
print(mb_substr($str,8,12));
?>
```

なお、同じような役割を持った関数として、mb_strcut 関数もあります。mb_strcut 関数の構文は mb_strsub 関数と同様ですが、引数 start, length には文字数ではなく、バイト数を指定する点が異なります。もしも引数 start がマルチバイト文字の2バイト目以降を示していた場合、mb_strcut 関数は1バイト目にさかのぼって文字列を抽出します。

➢ 特定の文字位置を検索する

ある文字列の中で特定の部分文字列が登場する位置を取得するには、mb_strpos 関数または、mb_strrpos 関数を使用します。mb_strpos 関数は、文字列の先頭（もしくは指定されたオフセット位置）から後方に向けて検索し、指定された部分文字列がはじめて登場した文字位置を返します。mb_strrpos 関数は、文字列の末尾から前方に向けて検索し、指定された部分文字列がはじめて登場した文字位置を返します。mb_strpos, mb_strrpos 関数の一般的な構文は次のとおりです。

```
mb_strpos, mb_strrpos 関数
int mb_strpos(string haystack, string needle [ , int offset [ , string encoding ] ] )
int mb_strrpos(string haystack, string needle [ , string encoding] )
  haystack : 検索対象の文字列
  needle : 検索文字列
  offset : 検索開始位置
  encoding : 文字コード名
```

たとえば、「にわにはにわにわとりがいる」という文字列から「にわ」という文字列を検索してみましょう。

```
strpos.php
<?php
$str='ねずみうしとらひつじ';
print('前方検索：'.mb_strpos($str, 'とら', 1, 'SJIS').'文字目<br>');
print('後方検索：'.mb_strrpos($str, 'とら', 'SJIS').'文字目<br>');
?>

前方検索：5 文字目
後方検索：5 文字目
```

mb_strpos 関数は1文字目から検索しているので始めて「とら」が登場するのは5文字目です。mb_strrpos 関数は末尾から検索を開始するので、初めて「とら」が登場するのは5文字目です。それぞれ返される文字位置は「0をスタート点にしている」ことに注目してください。また、mb_strrpos 関数は検索自体は後方から行いますが、戻り値の文字位置は先頭から数えた文字数になります。

➢ 文字列を変換する

マルチバイト文字固有の機能として、マルチバイト文字列をカタカナからひらがな、全角文字から半角文字などに変換するときに使用する mb_convert_kana 関数が提供されています。mb_convert_kana 関数の一般的な構文は次のとおりです。

mb_convert_kana 関数
string mb_convert_kana (string str, string option [, mixed encoding])
str : 任意の文字列
option : 変換オプション
encoding : 文字コード名

オプション	概要
r	「全角」英文字→「半角」英文字
R	「半角」英文字→「全角」英文字
n	「全角」数字→「半角」数字
N	「半角」数字→「全角」数字
a	「全角」英数字→「半角」英数字
A	「半角」英数字→「全角」英数字
s	「全角」スペース→「半角」スペース
S	「半角」英数字→「全角」英数字
k	「全角」カタカナ→「半角」カタカナ
K	「半角」カタカナ→「全角」カタカナ
h	「全角」ひらがな→「半角」カタカナ
H	「半角」カタカナ→「全角」ひらがな
c	「全角」カタカナ→「全角」ひらがな
C	「全角」ひらがな→「全角」カタカナ
V	濁点付きの文字を1文字に変換（K、H とあわせて）

➢ 文字コードを変換する

マルチバイト文字を扱う場合、文字コードは重要な要素です。一般的に、無用な混乱を防ぐためにも、アプリケーション内の文字コードはあらかじめ特定の一個に制約するのが原則ですが、外部のテキストファイルやデータベース、あるいは、外部サイトのコンテンツを引用するようなケースでは、必ずしも文字コードを統一できないケースは少なくありません。そのような場合には、アプリケーション内部で明示的に文字コードを変換する必要があります。

マルチバイト文字列関数において、文字コード変換の機能を担うのは mb_convert_encoding 関数と mb_convert_variables 関数です。mb_convert_encoding 関数は指定されたスカラー変数の文字コードを変換するのに対して、mb_convert_variables 関数は非スカラー関数の文字コードを変換します。配列やオブジェクトのような構造化データの文字コードを変換したい場合には、mb_convert_variables 関数を利用することで配下の要素(プロパティ)値を一括して変換できます。mb_convert_encoding、mb_convert_variables 関数の一般的な構文は次のとおりです。

```
mb_convert_encoding、mb_convert_variables 関数
string mb_convert_encoding ( string str, string to [ , mixed from ] )
string mb_convert_variables ( string to , mixed from , mixed vars )
  str, vars : 変換対象の変数
  to : 変換後の文字コード
  from : 変換前の文字コード
```

変換前後の文字コードには、UTF-32, UTF-16, UTF-7, UTF-8, ASCII, EUC-JP, SJIS, eucJP-win, SJIS-win, ISO-2022-JP, JIS, ISO-8859-1 などを指定することが可能です。変換前の文字コードには、「可能性のある」文字コードをカンマ区切りまたは配列で列挙することができ、また、"auto" を指定して文字コードを自動検出することも可能です。

たとえば、以下は変数 $data の内容を EUC-JP に変換するためのコードです。変換前の文字コードは Shift-JIS, UTF-8, JIS の順番で検出するものとします。

```
print ( mb_convert_encoding ( $data, 'EUC-JP', 'Shift0-JIS, UTF-8, JIS' ) );
```

2.マルチバイト正規表現関数

一言でいうなれば、正規表現とは文字列パターンを表現するための記法です。などと言ってしまうと、いかにも難しいものに聞こえるかもしれませんが、語弊を承知で言うならば、「ワイルドカードをより発展させたもの」と言ってもよいかもしれません。

ワイルドカードとは、ファイルの検索などでもよく使う「*.php」、「*data*.php」といった表現の「*」の部分を指します。「*」は「0文字以上の任意の文字列」を意味しています。つまり、「*.php」であれば、「a.php」や「xyz.php」といった任意の PHP ファイルを検索しますし、「*data*.php」なら「data.php」や「data01.php」、「_data00.php」のように、ベース名に「data」という文字を含む PHP ファイルを意味します。

ワイルドカードは、おそらく多くの皆さんにとって日常的に利用しているしくみだと思いますが、あくまで分かりやすさを旨としたしくみですから、それほど複雑なパターンを表現することはできません。そこで登場するのが「正規表現」です。たとえば、421-0002 のような郵便番号を表す正規表現パターンは、「[0-9]{3}-[0-9]{4}」です。「0～9 の数値 3 桁」＋「-」＋「0～9 の数値 4 桁」という文字列パターンを、これだけ短い表現の中で端的に定義しているわけです。

わずかこれだけの定義でも正規表現を使わずにチェックしようとしたら、なかなか面倒な手順を踏まなければなりません（おそらく、文字列長が8桁であること、4桁目に「-」を含むこと、それ以外の各桁が数値で構成されていることを、何段階かに分けてチェックしなければならないはずです）。

このような文字列パターンの検証だけではありません。正規表現を利用することで E-Mail アドレスや URL、HTML のタグなど、より複雑な文字列パターンを抽出したり、さらに抽出した文字列を置き換えたりすることが可能になります。これによって、たとえばフリーのドキュメントに含まれる URL 文字列に対して動的に<a>タグを割り当てたり、はたまた、HTML 上から<meta>タグの情報だけを取り出してリストを作成したり、などといった応用も考えられるでしょう。前節で紹介したマルチバイト文字列関数には正規表現も含まれており、基本的な正規表現による検索/置換/分割を利用することが可能です。

➢ 正規表現関数で利用できるパターン識別子

マルチバイト文字列関数で利用できる正規表現パターンにはさまざまな識別子が用意されています。以下はパターンの中のほんの一部にすぎませんが、これらを理解するだけでもかなりのパターンを表現できるようになるはずです。

正規表現	意味
[abc]	a, b, c のいずれか
[^abc]	a, b, c 以外
[a-zA-Z]	a から z、A から Z（大文字小文字を問わないすべてのアルファベット）
[a-z&&[^lmn]]	l, m, n を除く a から z までのすべての文字列（[a-ko-z]と同じ）
X?	X と 0～1 回一致（"fo?"は、"f"または"fo"とマッチ）
X*	X と 0 回以上一致（"fo*"は、"f"、"fo"、"foo"などとマッチ）
X+	X と 1 回以上一致（"fo+"は、"fo"、"foo"などとマッチ。"f"とはアンマッチ）
X{n}	X とn回一致（[a-z]{2}は、小文字のアルファベット 2 文字と一致）
X{n,}	X とn回以上一致（[0-9]{2,}は、数字2文字以上と一致）
X{n,m}	X とn～m回一致（[A-Z]{3,5}は、大文字アルファベット3～5文字と一致）
.	任意の一文字に一致
^	行の最初に一致
$	行の最後に一致
¥t	タブ文字に一致
¥n	ラインフィードに一致
¥r	キャリッジリターンに一致
¥d	数値に一致（[0-9]と同じ）
¥D	数値以外に一致（[^0-9]と同じ）
¥s	空白文字に一致（[¥t¥n¥xOB¥f¥r]と同じ）
¥S	空白以外の文字に一致（[^¥s]と同じ）
¥w	大文字小文字のアルファベット、数字、アンダーバーに一致（[a-zA-Z_0-9]と同じ）
¥W	文字以外に一致（[^¥w]と同じ）

➢ 正規表現による検索

正規表現による検索処理を行うには、mb_ereg 関数を使用します。mb_ereg 関数の一般的な構文は次のとおりです。

mb_ereg 関数
int mb_ereg (string pattern , string string [, array regs])
pattern : 正規表現パターン
string : 検索文字列の文字列
regs : 検索結果を格納する配列

mb_ereg 関数による検索結果は、戻り値として返されるのではなく、第 3 引数に指定された変数(配列)に格納されるという点に注意してください。この配列にはマッチングした文字列の情報がセットされます。ただ単にマッチングした文字列全体を取得したい場合は、配列の0番目の要素を参照します。正規表現パターン中の丸カッコで囲まれた部分にマッチした部分文字列(サブマッチ文字列)を取得したい場合は、1番目以降の要素を参照します。

たとえば、以下はテキストから URL 文字列を検索するためのコード例です。

```
ereg.php
<?php
$str = '検索サイトの代表的なものとしてグーグル(http://www.google.co.jp/)があげられます';
if ( mb_ereg ( ' (http:// | https://) [a-zA-z0-9 . / _¥-]+' ,$str ,$data ) ) {
   print ( $data[0] );
}
?>
```

実行結果
http://www.google.co.jp/

(http:// | https://) [a-zA-z0-9 . / _¥-]+ は URL 文字列を表す正規表現パターンです。具体的には、http://または、https://ではじまり、英数字、./_-のような文字が 1 文字以上続く文字列を表しています。この正規表現パターンで変数$str を検索し、その結果を変数$data に保存していると言うわけです。

なお、マルチバイト文字列関数には大文字小文字を区別しない mb_eregi 関数も用意されています。mb_eregi 関数は検索時に大文字小文字の違いを無視するだけで、構文は mb_ereg 関数と同様です。

➢ 正規表現による置換

正規表現による置換処理を行うのは、mb_ereg_replace 関数の役割です。mb_ereg_replace 関数の一般的な構文は次のとおりです。

mb_ereg_replace 関数
string mb_ereg_replace (string pattern, string replacement, string string)
pattern : 正規表現のパターン
replacement : 置き換え後の文字列
string : 置き換え対象の文字列

引数 replacement の中で、置き換え前のマッチングした文字列を引用することも可能です。たとえば、以下は URL 文字列を検索し、その部分を適切な<a>タグで囲むためのコード例です。

ereg_replace.php

```
<?php
$str = '検索サイトの代表的なものとしてグーグル（http://www.google.co.jp/）があげられます';
print ( mb_ereg_replace (  ' (http:// | https://) [a-zA-z0-9 . / _¥-]+',
      '<a href="¥0">¥0</a>' , $str ) );
?>
```

¥0 はマッチングした文字列全体を、¥1～9 はサブマッチング文字列を表しています。ちょうど mb_ereg 関数でマッチングした結果を配列に格納した場合のイメージと同様なのでわかりやすいとおもいます。mb_ereg 関数の場合と同様、mb_ereg_replace 関数には大文字小文字の違いを無視する mb_eregi_replace 関数が用意されています。

➢ 正規表現による分割

正規表現による分割処理を行うには、mb_split 関数を使用します。mb_split 関数の一般的な構文は以下のとおりです。

mb_split 関数
array mb_split (string pattern , string string [, int limit])
pattern : 正規表現パターン
string : 分割対象の文字列
limit : 分割回数

引数 limit を指定した場合は、子定数を上限として文字列が分割されます。
たとえば次のスクリプトでは、「YYYY-MM-DD」、「YYYY/MM/DD」、「YYYY.MM.DD」という形式の日付データを「/」、「-」、「.」などの区切り文字で分割し、その結果を再結合した結果を「YYYY 年 MM 月 DD 日」として返しています。

split.php

```
<?php
$str='2009/05/13';
$aryDate=mb_split('[/.¥-]' ,$str);
print ( $aryData[0] . '年' . $aryData[1] . '月' . $aryData[2] . '日' );
?>
```

実行結果
2009 年 05 月 13 日

3.日付・時刻関数

PHP では、日付/時刻データの演算や整形を行うための関数が「日付・時刻関数」として数多く取り揃えられています。日付・時刻関数は PHP の標準モジュールであり、使用するための特別な設定は必要ありません。

➢ タイムスタンプ

日付・時刻関数の多くは、日付・時刻データを「タイムスタンプ」値として管理します。タイムスタンプについては日付・時刻データを 1970 年 1 月 1 日からの経過秒（整数値）で表現したものです。日付・時刻データは年月日、時分秒など複数の情報を持つ構造的なデータであり、これを直接に演算したり、別形式のフォーマットに変換するのは意外と難しいものです。しかし、タイムスタンプという整数値で日付・時刻を表現することで日付の加算/減算、比較なども整数値を演算/比較するのとまったく同じ要領で行えます。タイムスタンプを生成するための関数には、次のようなものがあります。

✓ time、microtime 関数

現在日付のタイムスタンプ値を、それぞれ秒単位、マイクロ秒単位で返します。microtime 関数はデフォルトで "msec sec" 形式の文字列を返しますが、引数に TRUE を設定することで、float 値として値を取得することも可能です。microtime 関数の引数は PHP 5.0.0 以降で利用可能です。

✓ mktime 関数

与えられた日付・時刻情報に基づいて、対応するタイムスタンプ値を返します。mktime 関数の一般的な構文は次のとおりです。

```
mktime 関数
int mktime ( [ int hour [ ,int minute [ ,int second [ ,int month [ ,int day [ ,int year [ ,int is_dst] ] ] ] ] ] ] )
  hour : 時   minute : 分  second : 秒
  month : 月   day : 日      year : 年      is_dst : サマータイム
```

mktime 関数は指定された日時がありえない日付であった場合、適当な日付への変換を自動的に行います。たとえば、「2005 年 10 月 32 日」のような日付を指定した場合、内部的には「2005 年 11 月 1 日」であるとみなされます。

✓ strtotime 関数

strtotime 関数は、前述の関数に比べると、やや変り種の機能を提供します。strtotime 関数を利用すると、英文形式の文字列からタイムスタンプ値を取得することができます。たとえば、1年前のタイムスタンプ値を取得したいならば次のようにします。

```
$timestamp = strtotime ( '+1 year' );
```

その他、"+3 day"（3日後）や、"+2 week 3days 4 hours"（2週間3日4時間後）、"next Friday"（次の金曜日）、"last Sunday"（先週の日曜日）などの表現が可能です。直感的に日付を指定できるので、なにかしら固定的な日付データを生成したい場合には便利でしょう（たとえば、クッキーの有効期限など）

➢ 日付データを特定のフォーマットに整形する

あくまでタイムスタンプは内部的な演算のための表現形式です。人間が視認するのに向いた形式であるとは言えません。そこで登場するのが date 関数です。date 関数を利用することで、指定されたタイムスタンプ値を日付文字列に整形することができます。data 関数の一般的な構文は次のとおりです。

data 関数
string date (string format [, int timestamp])
format : フォーマット文字列
timestamp : 整形するタイムスタンプ値

引数 timestamp には、time、mktime 関数などを使って取得したタイムスタンプ値を指定します。この引数を省略した場合は、現在日時のタイムスタンプがデフォルトで適用されます。引数 format で利用できるフォーマット記述子を以下に示します。これらの記述子を組み合わせることで、フォーマット文字列を作成できます。

記述子	概要	値
a	午前/午後	am|pm
A	午前/午後	AM|PM
d	日	01～31
D	曜日（省略形）	Mon～Sun
F	月（長い形式）	January～December
h	時（12 時間単位）	01～12
H	時（24 時間単位）	01～23
g	時（12 時間単位）	1～12
G	時（24 時間単位）	1～23
i	分	00～59
j	日	1～31
l	曜日（長い形式）	Monday～Sunday
L	うるう年であるかどうか	0|1
m	月	01～12
n	月	1～12
M	月（省略形）	Jan～Dec
r	RFC822 フォーマットの日付	（たとえば）"Wed, 21 Jan 2005 11:34:19 +0900"
s	秒	00～59
S	序数を表す接尾辞	st|nd|th
t	月の日数	28～31
T	タイムゾーン	（たとえば）"MDT"
U	タイムスタンプ	–
w	曜日	0（日曜）～6（土曜）
Y	年（4桁）	（たとえば）"2005"
y	年（2桁）	（たとえば）"05"
z	年間の通算日	0～365
Z	タイムゾーンのオフセット秒数	-43200～43200

たとえば、以下は現在の日時を「YYYY 年 MM 月 DD 日（曜日）」の形式で表示します。

```
<?php
print ( date ( 'Y 年 m 月 d 日（D）' );
?>
```

もちろん、time、mktime、strtotime 関数を利用して、特定の日時を整形して表示することも可能です。

```
<?php
print ( date ( 'Y 年 m 月 d 日（D）', time() + 60*60*24*15 ) . '<br>' ); //15 日後の日付
print ( date ( 'Y 年 m 月 d 日（D）', mktime(0, 0, 0, 10, 25, 2005)) . '<br>'); //2005 年 10 月 25 日を指定
print ( date ( 'Y 年 m 月 d 日（D）', strtotime('+2week')) . '<br>'); //2 週間後の日付
?>
```

ちなみに、data 関数と類似した関数として gmdate 関数もあります。gmdate 関数の構文は date 関数と同様ですが、引数 timestamp が省略された場合の挙動が異なります。

```
<?php
print ( '現在時刻：' . date( 'Y 年 m 月 d 日（D） H:i:s' ) . '<br>' );
print ( 'グリニッジ標準時：' . gmdate( 'Y 年 m 月 d 日（D） H:i:s') . '<br>');
?>
```

このプログラムを実行してもらえればおわかりだと思いますが、gmdate 関数の結果は date 関数よりも 9 時間遅れています。このように、gmdate 関数はタイムスタンプ値が省略された場合、グリニッジ標準時（GMT）を計算して返します。これによって、サーバのタイムゾーンなどによる差を意識することなく、常に等しい標準時を得ることが可能になります。

➢ 日付の妥当性を検証する

ここまでに登場した関数とはやや毛色こそ異なりますが、日付関数に含まれる重要な関数のひとつとして、checkdate 関数があります。checkdate 関数は、与えられた日付が「実際に存在する日付」であるかどうかをチェックします。

checkdate 関数
bool checkdate (int month, int day, int year)
month : 月 day : 日 year : 年

checkdate 関数がチェックする基準は次のとおりです。
・年が1～32767 の範囲であること
・月が1～12 の範囲であること
・日が指定された月の日数に含まれること(うるう年も認識)

次の例では、指定された年月のカレンダをテキスト表示する簡単なユーザ定義関数を作成しています。

```
<?php
function calendar ( $year, $month ) {
  for ( $i=1;$i<32;$i++){
    if (checkdate ($month, $i, $year)){ print (″$i  ″); }
  }
}
print ( '2006 年 2 月のカレンダ:<br>');
calendar(2006,2);
?>
```

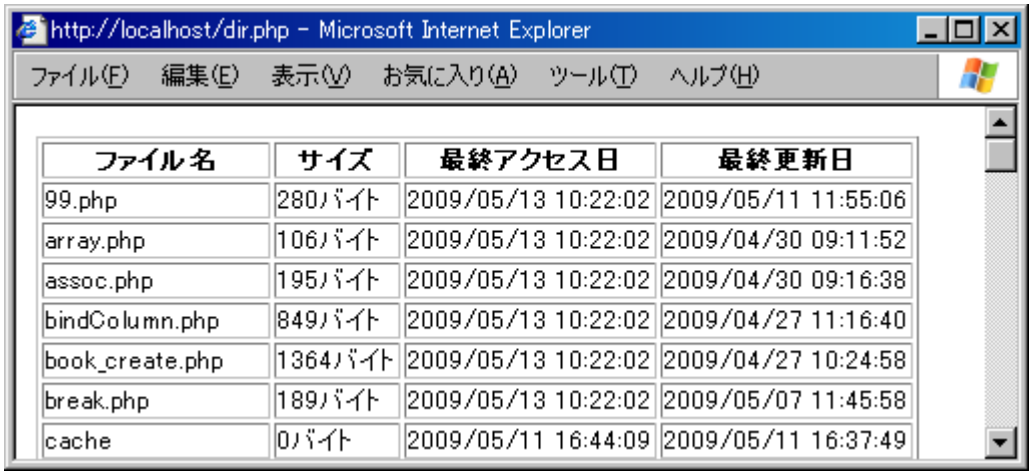
4.ディレクトリ関数＆ファイルシステム関数

ディレクトリ関数とファイルシステム関数は、サーバ上のファイルシステムを操作するための関数です。サーバ上の特定のディレクトリ構造やファイル情報を取得したり、ファイルの読み書きを行うために使用します。ディレクトリ関数とファイルシステム関数はいずれも PHP 本体に組み込まれているため、使用するために特別な準備は不要です。

➢ 例）ディレクトリ情報を一覧表示する

次のサンプルでは、サーバ上の特定のディレクトリ配下のファイル情報を一覧表示します。

```
dir.php
<table border="1">
<tr>
  <th>ファイル名</th><th>サイズ</th>
  <th>最終アクセス日</th><th>最終更新日</th>
</tr>
<?php
clearstatcache();    //キャッシュをクリア
$dir=opendir('./'); //カレントディレクトリをオープン
while($file=readdir($dir)){   //ディレクトリの内容を読み込み
  if($file!='..' && $file!='.'){
    print('<tr>');
    print('<td>'.$file.'</td>');
    print('<td>'.filesize($file).'バイト</td>');
    print('<td>'.date('Y/m/d H:i:s', fileatime($file)).'</td>');
    print('<td>'.date('Y/m/d H:i:s', filemtime($file)).'</td>');
    print('</tr>');
  }
}
closedir($dir); //ディレクトリをクローズ
?>
</table>
```



ファイル名	サイズ	最終アクセス日	最終更新日
99.php	280バイト	2009/05/13 10:22:02	2009/05/11 11:55:06
array.php	106バイト	2009/05/13 10:22:02	2009/04/30 09:11:52
assoc.php	195バイト	2009/05/13 10:22:02	2009/04/30 09:16:38
bindColumn.php	849バイト	2009/05/13 10:22:02	2009/04/27 11:16:40
book_create.php	1364バイト	2009/05/13 10:22:02	2009/04/27 10:24:58
break.php	189バイト	2009/05/13 10:22:02	2009/05/07 11:45:58
cache	0バイト	2009/05/11 16:44:09	2009/05/11 16:37:49

clearstatcache 関数は、ファイルシステムに関するキャッシュをクリアします。ファイルシステムへのアクセスは往々にして負荷の高い処理であるため、ファイルシステム関数は取得した情報をキャッシュし、システム上に維持します。しかし、ここでは、最終更新日などの情報を常に最新に保っておきたいので、clearstatecache 関数でキャッシュをクリアする必要があります。

特定のディレクトリにアクセスするのは opendir 関数の役割です。opendir 関数は指定されたディレクトリの取得に成功すると、ディレクトリハンドラを返します。ディレクトリハンドラとは、指定されたディレクトリをハンドル（操作）するためのキーとなる情報と思っていただければよいでしょう。後続のディレクトリ関数では、このディレクトリハンドラをキーにしてディレクトリへの操作を行います。

readdir 関数は、ディレクトリは以下のファイル（サブフォルダ）を読み込むための関数です。読み込みに成功した場合は次のファイルのファイル名を返し、次のファイルがない場合は FALSE を返します。ここでは、readdir 関数のこの性質を利用して、readdir 関数が FALSE を返すまで while ループを繰り返し、ディレクトリ配下のすべてのファイルを取得しています。ちなみに、readdir 関数は特別なファイルとしてカレントディレクトリと上位のディレクトリを返します。ここではこれらの特殊ファイルを省いて出力しています。

各ファイルに関する情報を取得するには、ファイルシステム関数に属する次の関数を使用します。

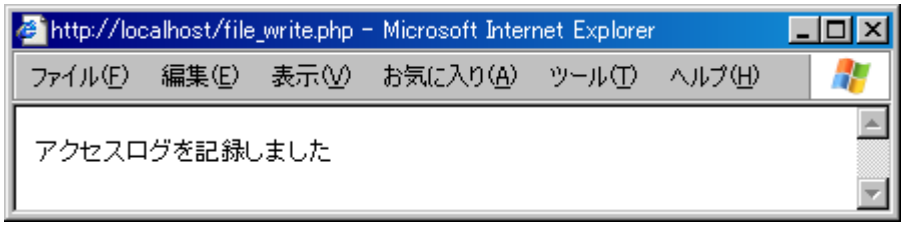
関数	概要
int filesize (string filename)	ファイルサイズ（バイト単位）を返す
int fileatime (string filename)	ファイルの最終アクセス日時（タイムスタンプ）を返す
int filemtime (string filename)	ファイルの最終更新日時（タイムスタンプ）を返す

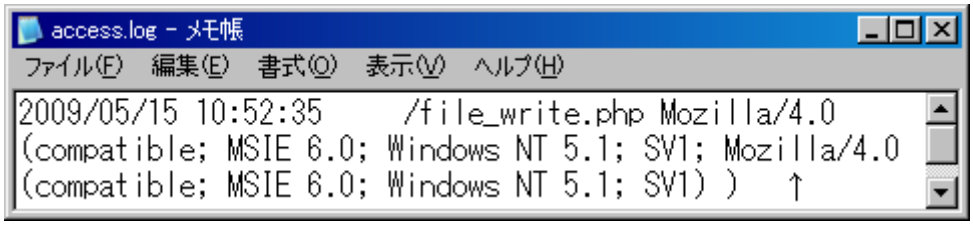
fileatime 関数と filemtime 関数はタイムスタンプ値を返すので、date 関数を使用して適切なフォーマットへ整形します。処理の終わったディレクトリハンドラは closedir 関数でクローズします。

➢ 例）テキストファイルへの書き込み

ここからは、ファイルシステム関数が提供する主要な機能であるファイルの読み書きを扱っていきます。次のサンプルは、ユーザがサイトにアクセスしたタイミングで、「アクセス日時」、「アクセスした URL」、「クライアントの種類」、「リンク元の URL」をテキストファイルに書き込むためのコードです。

```
file_write.php
<?php
$data=date('Y/m/d H:i:s').″¥t″;
$data.=$_SERVER['SCRIPT_NAME'].″¥t″;
$data.=$_SERVER['HTTP_USER_AGENT'].″¥t″;
$data.=$_SERVER['HTTP_REFERER'];
$file=fopen('access.log', 'a');
flock($file, LOCK_EX);
fwrite($file, $data.″¥n″);
flock($file, LOCK_UN);
fclose($file);
print('アクセスログを記録しました');
?>
```





コードを実行した結果、access.log に情報が記録されれば成功です。ファイルシステム関数でファイルを操作するには、まず fopen 関数でファイルを開く必要があります。fopen 関数の一般的な構文は次のとおりです。

```
fwrite 関数
int fwrite ( resource handle, string string [, int length ] )
  handle : 使用するファイルハンドラ
  string : 書き込む文字列
  length : 書き込むバイト数
```

引数 length を指定した場合は、指定の文字数の先頭から length バイト分だけ書き込みが行われます。今回の例では、「アクセス日時」、「アクセスした URL」、「クライアントの種類」、「リンク元の URL」をタブ文字（¥t）区切りで連結したものを書き込みます。行の終わりには、必ず「改行文字（¥n）」を付加します。ただし、改行文字は使用しているプラットフォームによって異なりますので、注意してください。今回は Linux 環境を想定して「¥n」としていますが、Windows 環境では「¥r¥n」に変更してみてください。

以上がファイル書き込みの最も基本的な構文です。これらの構文のなかで注意しなければならない点として、複数の人間が同時に同じファイルにアクセスする可能性というものも考慮しなければなりません。ユーザ A がファイルを開いている最中にユーザ B が変更を保存するといったことになれば、ファイルの内容が

変わってしまうことになるので、この場合、ユーザ A がファイルを開いた時点でファイルに排他制御をかけなければならないのです。つまり、ファイルを占有しておく必要があるのです。このようなファイルのロックを PHP で行うには、flock 関数を使用します。

flock 関数
bool flock (resource handle, int operation)
handle : ファイルハンドラ
operation : ロックモード

flock 関数で使用できるロックの種類は以下のとおりです。

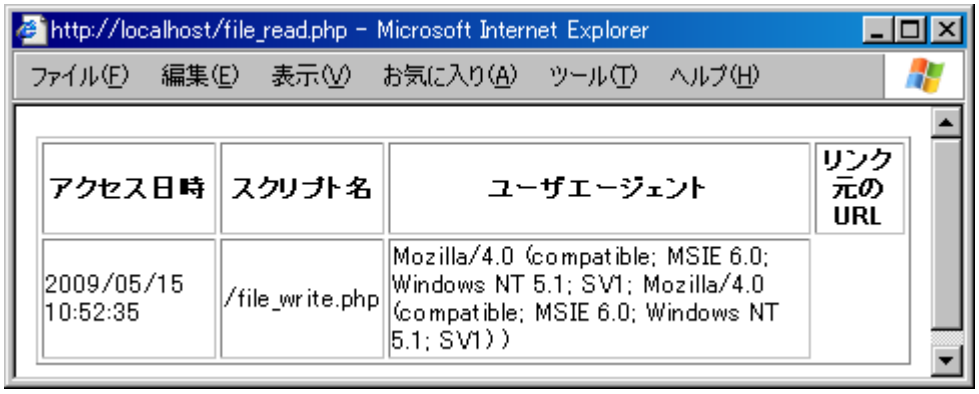
定数	値	概要
LOCK_SH	1	共有ロック
LOCK_EX	2	排他ロック
LOCK_UN	3	ロックの解除
LOCK_NB	4	ロック中に flock によるブロックを禁止

共有ロック（LOCK_SH）はファイル読み込みの場合に使用するロックで、ファイル書き込みを行う場合には、より厳密な排他ロック（LOCK_EX）を使用します。排他ロックがかかっている間は、ロックが解除されるまで、他のプログラムはそのファイルを操作することはできないので、同時アクセスによる不整合を未然に回避することができます。書き込みが終了したあとはすみやかにロックを解除（LOCK_UN）し、fclose 関数でファイルをクローズしてください。

➢ 例）テキストファイルの読み込み

今度は、書き込んだアクセスログを読み込み、HTML テーブルとして、一覧表示してみましょう。

```
file_read.php
<table border="1">
<tr>
  <th>アクセス日時</th><th>スクリプト名</th>
  <th>ユーザエージェント</th><th>リンク元の URL</th>
</tr>
<?php
$file=fopen('access.log', 'r'); //ログファイルを読み取り専用でオープン
while($line=fgets($file, 1024)){
  $data=explode("¥t", $line); //タブで文字列を分割
  print('<tr>');
  foreach($data as $value){
    print('<td>'.$value.'</td>'); //分割した結果を順番に出力
  }
  print('</tr>');
}
fclose($file);
?>
</table>
```



ファイルを書き込む関数にはいくつか種類がありますが、最も汎用的に利用できるのが fgets 関数です。fgets 関数の一般的な構文は次のとおりです。

fgets 関数
string fgets (resource handle [, int length])
handle : 書き込み対象のファイルハンドラ
length : 読み込むバイト数

fgets 関数はファイルポインタから現在の行を取得し、かつ、ファイルポインタを次の行に移動します。ファイルポインタとは、ファイルの読み書きを行う際の作業位置を示す「しおり」のようなものだと考えればよいでしょう。ファイルハンドラは内部的にファイルポインタを管理することで、現在の作業位置を記録し、先頭からシーケンシャル(順番)にファイルを処理することができます。

fgets 関数はファイルポインタが次に勧めない(次の行が存在しない)場合に FALSE を返します。今回の例では fgets 関数のこの性質を利用して、ファイルポインタが最終行に達するまで while ループを繰り返しています。explode 関数は指定された文字列で文字列を分割するための関数です。ここでは、タブ文字(¥t)で文字列を分割し、分割結果の配列を順にテーブルセルとして出力します。

explode 関数
array explode (string separator, string string [, int limit])
separator : 区切り文字
string : 分割対象の文字列
limit : 分割の最大回数

explode 関数の代わりに、mb_split 関数を利用してもよいと思われるかもしれませんが、mb_split 関数は正規表現を使用しているため、オーバーヘッドも大きいという欠点があります。この例のように区切り文字を固定文字として指定できるならば、より負荷が小さい explode 関数を使用することをお勧めします。

ちなみに、ファイルシステム関数には行読み込みと区切り文字分割とを同時に行う fgetcsv 関数も用意されています。今回の例のように、タブ区切りテキストやカンマ区切りテキストを読み込む場合には、こちらの関数を利用するとよりシンプルにコードを記述することができます。

fgetcsv 関数
array fgetcsv (resource handle, int length [, string delimiter [, string enclosure]])
handle : 読み込み対象のファイルハンドラ
delimiter : 分割時の区切り文字
enclosure : 囲み文字

fgetcsv 関数を利用して、先ほどの file_read.php を書き換えてみましょう。fgetcsv 関数を利用すると、explode 関数による分割処理を省略できるので、よりコードがシンプルになったことがお分かりでしょう。

file_read2.php

```
<table border="1">
<tr>
  <th>アクセス日時</th><th>スクリプト名</th>
  <th>ユーザエージェント</th><th>リンク元の URL</th>
</tr>
<?php
$file=fopen('access.log', 'r'); //ログファイルを読み取り専用でオープン
while($data=fgetcsv($file, 1024, "¥t")){
  print('<tr>');
  foreach($data as $value){
    print('<td>'.$value.'</td>'); //分割した結果を順番に出力
  }
  print('</tr>');
}
fclose($file);
?>
</table>
```