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

1.言語仕様

PHP は Common Gateway Interface (CGI) をサポートしており、CGI をサポートする Web サーバ上で PHP スクリプトを実行することができます。この他にも FastCGI を通じて Web サーバから起動したり、利用する Web サーバがモジュール機構を提供していれば、モジュールとして Web サーバに組み込んだ状態で動作させることが可能です。ちなみに、Common Gateway Interface (コモン・ゲートウェイ・インタフェース、CGI)とは、ウェブサーバ上でユーザプログラムを動作させるための仕組みのことを指します。

PHP は、PHP4 以降 Zend Engine をスクリプト言語を処理するエンジンとして利用されてきました。Zend Engine は、PHP3 の開発者によって設立された Zend Technologies Ltd.により開発されたソースコードが公開されているスクリプティングエンジンで、PHP5 では Zend Engine 2.0 になったものを使用しています。Zend Engine は基本的に 1 つのプロセスがインタプリタのコンテキストを 1 つだけ持つものとして設計されているため、現状はマルチスレッドを用いたスクリプティングはサポートしていません。Zend Engine を除く PHP 本体は、PHP 組み込み関数の実装を含むほか、Web サーバやコマンドラインインターフェイスとスクリプティングエンジンの間を仲介する SAPI (Server API)レイヤ、マルチスレッドで動く Web サーバのモジュールとして利用される場合にグローバル変数のセマンティクスを提供する TSRM (Thread Safe Resource Manager)、さまざまなプラットフォームの入出力機構や API の差異を吸収するための Streams レイヤから構成されています。

PHP を動作させる方法には、実行ファイル形式(CGI / FastCGI)、モジュール形式(mod_php / ISAPI)があり、どの形式を使用するかは Web サーバにより異なります。例を挙げるならば、Apache で動作させる場合は mod_php を使用し、IIS で動作させる場合は ISAPI を使用し、lighttpd で動作させる場合は FastCGI を使用し、ANHTTPD で動作させる場合は CGI を使用します。PHP においては一般的に実行ファイル形式よりもモジュール形式の方が高速に動作するため Apache と mod_php の組み合わせがよく用いられています。PHP には Windows、UNIX などのオペレーティングシステムに対応した処理系が存在し、一部の組み込み関数はプラットフォームごとに挙動が違うため、スクリプトによっては移植作業が必要になる場合があります。また、多くの DBMS へのインターフェイスを標準で備えており、DBMS との連携にも高い力を発揮します。HTML 埋め込み型の処理系としては他に ASP、JSP などがあります。

2.構文規則

多くの構文を C 言語、Java、Perl などのプログラミング言語から転用しており、動的に生成させるウェブ・ページを速やかに作成できるのが特徴で"<?php" と "?>"で囲まれた内部を PHP コードと認識し実行され、最終的には HTML ファイルへ変換され出力されます（標準的なウェブ開発利用のとき）。それ以外でも、バッチ処理をこなすためのプログラムを作成しサーバーを介さずに動作させることも可能です。

```
<html>
<body>
<?php
  // この部分に PHP プログラムを書いていく
  print "こんにちは！！";
?>
```

このあたりは、普通の HTML として処理されます。

```
<?php
  // プログラム部分はいくつあってもかまいません。
?>
</body>
</html>
```

基本的な分岐処理や繰り返し処理の命令系は C や Java といったプログラミング言語と同じです。また、処理の流れも他の言語と同じく入力→処理（順次、分岐、繰り返し）→出力といったロジックの流れが決まっているので読みにくいソースやコードがながいものはこの3段階に一度わけて考えを切り替えていったほうが読解力は高まります。以降でも紹介しますが、PHP は他の言語と比べとくにコードのまとまりがきれいなほうなので読みやすく書きやすい言語です。

3.PHP で扱えるデータ型

スカラー型	文字列型 (String) 整数型 (Integer) 浮動小数点数型 (Float) 論理型 (Boolean)
複合型	オブジェクト型 (Object) 配列型 (Array)
特殊型	リソース型 (Resource) ヌル型 (Null)

PHP で使用できるデータの型としてはスカラー型、複合型、特殊型とあります。
また、文字列の値を代入する場合、PHP では値をダブル(シングル)クォーテーションで囲む必要があります。
ダブルクォーテーションとシングルクォーテーションのどちらを使うかは、文字列に含まれる変数を展開するかどうかと言う点で異なります。ダブルが変数の展開、シングルは文字列として出力されます。

var.php
```
<?php
  $data = 'PHP をはじめよう';
  echo '$data の値は、「' . "$data" . '」です。';
?>
```

出力結果
$data の値は、「PHP をはじめよう」です。

➢ データ型のキャスト

PHP はデータ型に比較的寛容な言語です。データの中身から自動的に判断して、適切な入れ物(変数)を割り当ててくれるのが、PHP のいいところです。しかし、値の厳密な比較や演算を行いたい場合には、「型キャスト」という仕組みをしようします。型キャストを行うには、変換したい変数の直前で「(データ型)」の形式で型指定を行います。

cast.php
```
<?php
   $data='123'; //文字型の 123
   var_dump($data);

   $data=(int)$data; //文字型から数値型へキャスト
   var_dump($data);
?>
```

出力結果
String(3) "123" int(123)

型	概要
(bool), (boolean)	TRUE/FALSE 型へのキャスト
(double), (float), (real)	浮動小数点へのキャスト
(int), (integer)	整数型へのキャスト
(string)	文字列型へのキャスト
(object)	オブジェクト型へのキャスト

➢ 配列

ここまで扱った変数は、すべてひとつの変数（入れ物）に対して、ひとつの値をもつばかりでした。このような値のことを「スカラー変数」と言います。一方、PHP では、複数の値を格納する入れ物（変数）を扱うことができます。このような入れ物のことを「非スカラー変数」といい、非スカラー変数の代名詞として配列があります。配列は添え字と呼ばれる仕切りのある入れ物だと思ってください。配列には添え字に 0 から始まる連番からなる通常配列と、添え字に好きな名前をつけられる連想配列があります。array 命令では、配列に格納したい要素をカンマ区切りで記述します。キーが省略された場合には、キー名には「その時点でのキーの最大整数値＋１」が振り分けられます。つまり、先頭から順に「0,1,2,3...」と振られます。

✓ 通常配列

array.php

```
<?php
    $data=array(0=>'山田', 1=>'掛谷', 2=>'日尾', 3=>'本多');
    print($data[3]); //本多を出力
?>
```

✓ 連想配列

assoc.php

```
<?php
    $data=array(
            'PHP'=>'PHP:Hypertext Preprocessor',
            'XML'=>'Extensible Markup Lunguage'
    );
    print($data['PHP']); //PHP:Hypertext Preprocessor を出力
?>
```

➢ 可変変数

可変変数とは、「変数名を変数によって決める」ことができる変数のことを言います。可変変数では、先頭に「＄」マークをふたつ重ねることにより、変数名を別の変数値で動的に決定します。

✓ 可変変数

mvar.php

```
<?php
    $x='title';
    $title='PHP:Hypertext Preprocessor';
    print($$x); //PHP:Hypertext Preprocessor を出力
?>
```

➢ スーパーグローバル変数

PHP で利用できるのは、自前で確保する変数ばかりではありません。PHP には、内部的に生成される変数として「スーパーグローバル変数」が用意されています。スーパーグローバル変数とは、主にクライアントからサーバに送信されたデータ(リクエスト情報)を管理するための変数で、たとえば、HTML フォームから送信されたデータやクッキーなどの情報を取得することができます。

変数名	内容
$_POST	POST 形式の HTML フォームから渡された情報
$_GET	クエリ情報「？キー名＝値」経由を渡された情報
$_FILES	アップロードされたファイルに関する情報
$_SERVER	サーバ環境変数に関する情報
$_ENV	環境変数に関する情報
$_COOKIE	クッキー経由で渡された情報
$_SESSION	セッション経由で渡された情報
$_REQUEST	$_GET, $_POST, $_COOKIE, $_FILES の値をまとめて管理

一見すると、$_REQUEST は、$_GET, $_POST, $_COOKIE, $_FILES の値をまとめて扱えるため便利にも見えますが、「同名のキーがあった場合、片方のキーが上書きされてしまう」または、「どこからデータが送信されたのか曖昧になりやすい」などの理由から、原則として使用するべきではありません。以降からは、$_REQUEST を除く一連のスーパーグローバル変数の概要を紹介していきます。

➢ $_POST

$POST は、POST 形式の HTML フォーム(<form>タグの method オプションが POST)から入力された情報(ポストデータ)を取得するためのスーパーグローバル変数です。たとえていうなら次のようなフォームを想定してみましょう。

✓ post1.php

```
<html>
<head>
<title>スーパーグローバル変数</title>
</head>
<body>
<form method="POST" action="post2.php">
名前:<input type="text" name="name" size="15" /><br>
性別:<input type="text" name="sex" size="15" /></br>
<input type="submit" value="送信" />
</form>
</body>
</html>
```

✓ post2.php

```
名前:<?php print($_POST['name']); ?><br>
性別:<?php print($_POST['sex']); ?>
```

post1.php と post2.php を用意し、post1.php のフォームに入力して送信ボタンをクリックすると、POST でわたったデータが post2.php で見れるようになります。$_POST の中身は連想配列になっているので、要素値にアクセスするには name 属性で指定した値を添え字とし、呼び出し元はそれをもとに値を受け取ります。

➢ $_GET

$_GET は、クエリ情報を取得するためのスーパーグローバル変数です。クエリ情報とは、URL の末尾「？」以降から付加された情報のことを指します。「？」の後ろにつくものが「キー名＝値」のセットになっています。アンカーにべた書きする方法もありますが、<form>タグの method オプションで GET 指定することでもクエリ情報を送信できます。もともと、クエリ（Query）とは「問い合わせ」という意味を持っていますが、このクエリ情報も、もっぱらアプリケーションに対する問い合わせのための短いキーワードを送信する目的で利用されます。クエリ情報では、「キー名＝値」のセットを「＆」で連結して、複数の情報を送信することもできます。
$_GET ではいくつかの制限があります。

1.送信可能なデータサイズに制限がある

ポストデータには原則としてサイズ制限がないのに、対して、クエリ情報のデータサイズは Web サーバが受け取ることができる URL の長さに制約されます。具体的なサイズは使用している環境によりますが、一般的には最大でも数百バイト程度にとどめておくのが無難でしょう。

2.予約文字やマルチバイト文字は使用できない

ポストデータでは原則としてあらゆる文字データを扱うことが可能です。一方、クエリ情報ではファイル名との区切り文字である「？」をはじめ、「＆」や「%」、空白、マルチバイト文字（日本語）などは使用できません。これらがクエリ情報に含まれている可能性がある場合には、あらかじめ無害な文字列に変換する必要があります。PHP では urlencode という命令が用意されており、こうした変換処理（URL エンコードと言います）を簡単に行うことができます。

3.データが露出する

ポストデータはリクエスト情報の本体として送信されるので、特別なツールを利用しない限り、送信内容がエンドユーザの目に触れることはありません。しかし、クエリ情報はその性質上、ブラウザのアドレス欄にそのまま露出してしまいますし、サーバのログなどにも記録される場合があります。ログインパスワードのように、本来秘密にするべき情報をクエリ情報を介してやりとりするのは好ましくありません。

✓ get1.php

```
<html>
<head>
<title>スーパーグローバル変数</title>
</head>
<body>
<form method="GET" action="get2.php">
名前:<input type="text" name="name" size="15" /><br>
性別:<input type="text" name="sex" size="15" /></br>
<input type="submit" value="送信" />
</form>
</body>
</html>
```

✓ get2.php

```
名前:<?php print($_GET['name']); ?><br>
性別:<?php print($_GET['sex']); ?>
```

➢ $_FILES

$_FILE は、アップロードしたファイルに関する情報を取得するためのスーパーグローバル変数です。$_FILE を利用すると、ファイルのアップロード機能を簡単に作成することができます。ファイルをアップロードする場合には、<form>タグの enctype オプションに必ず multipart/form-data をセットする必要があります。PHP は、クライアントからアップロードファイルを受け取ると、これを所定のフォルダに一時ファイルとして保存するので、これを本当に保存したいフォルダに移動するだけです。ファイルを移動するには move_uploaded_file 命令を使用します。move_uploaded_file 命令の一般的な構文は次のとおりです。

move_uploaded_file(対象となるファイルパス, 移動先のパス)

対象となるファイルのパスは、スーパーグローバル変数$_FILES を介して取得することができます。$_FILES は、$_POST や$_GET などと異なり、2次元配列として与えられ、以下のような情報を保持しています。"file" は、HTML フォーム内で指定された要素名とします。

要素名	概要
$_FILES['file']['name']	オリジナルのファイル名
$_FILES['file']['tmp_name']	サーバ上で管理された一時的なファイル名
$_FILES['file']['size']	アップロードファイルのサイズ（バイト）
$_FILES['file']['type']	アップロードファイルの MIME タイプ（ファイルの種類）
$_FILES['file']['error']	オリジナルのファイル名

✓ file1.php

```
<html>
<head>
<title>スーパーグローバル変数</title>
</head>
<body>
<form method="POST" action="file2.php" enctype="multipart/form-data">
ファイルのパス:<input type="file" name="uploaded" size="50" /><br>
<input type="submit" value="アップロード" />
</form>
</body>
</html>
```

✓ file2.php

```
<?php
    move_uploaded_file($_FILES['uploaded']['tmp_name'],
     './doc/' .$_FILES['uploaded']['name']);
    print('アップロードに成功しました');
?>
```

file2.php の例では移動先のパスに「./doc/」を指定していますが、「.」は現在のスクリプトが存在するカレントディレクトリを表します。つまり、「./doc/」はカレントディレクトリ直下の doc ディレクトリを表し、「' ./doc/ ' . $_FILES['uploaded']['name']」で doc ディレクトリの配下にオリジナルのファイル名でアップロードファイルを保存することを表すわけです。なお、「' ./doc/ '」と「$_FILES['uploaded']['name']」とをつなぐ「.」は、文字列連結演算子と呼ばれ、2つの文字列を連結する際に使用します。

以上を理解したら、file1.php を実行し、なにかファイルを選択してアップロードしてみましょう。成功の趣旨のメッセージが表示され、ディレクトリにアップロードされていることを確認してください。

うまくいかない場合は、いくつかの原因が考えられます。アップロードがうまくいかない場合の原因を知るには、$_FILES['file']['error']を介してエラーコードを取得するのが便利です。$_FILES によって返されるエラーコードは以下のとおりです。

定数	値	概要
UPLOAD_ERR_OK	0	アップロード成功
UPLOAD_ERR_INI_SIZE	1	アップロードファイルのサイズが upload_max_filesize パラメータ(php.ini)の指定値を超過
UPLOAD_ERR_FILE_SIZE	2	アップロードファイルのサイズが HTML フォームの hidden フィールド MAX_FILE_SIZE で指定された値を超過
UPLOAD_ERR_PARTIAL	3	アップロードファイルの一部しかアップロードされていない
UPLOAD_ERR_NO_FILE	4	アップロード失敗

UPLOAD_ERR_IN_SIZE, UPLOAD_ERR_NO_FILE が返された場合には、php.ini における以下のパラメータを変更してください。

パラメータ	概要
upload_max_filesize	アップロードのサイズ制限
memory_limit	スクリプトが使用可能なメモリサイズ
post_max_size	ポストデータのサイズ制限
max_execution_time	スクリプトのタイムアウト時間(秒)

アップロードに直接関係するのは upload_max_filesize パラメータですが、たとえばアップロードサイズの制限には引っかかっていなくても、ポストデータのサイズ制限に引っかかっているということは意外とよくある話なので注意が必要です。また、移動先のディレクトリに書き込み権限がない場合もアップロードに失敗するので、パーミッションを確認してください。UPLOAD_ERR_SIZE が返された場合には、HTML フォーム内に以下の記述がないかどうかを確認し、値を編集してください。

```
<input type="hidden" name="MAX_FILE_SIZE" value="30000" />
```

これは php.ini よりも手軽なファイルサイズ制限のアプローチです。ただし、こちらの方法はクライアント側で自由に変更できるという意味で信頼性は低いアプローチでもあります。確実にファイルサイズを制限したい場合には、upload_max_filesize パラメータを使用してください。

➢ $_SERVER

クライアントサーバー間でのやりとりされる情報は、なにも URL やコンテンツのように目にみえるものばかりではありません。たとえば、クライアントの種類や対応言語、あるいはサーバからそう芯されたコンテンツの種類やデータサイズなどの情報が、クライアント/サーバそれぞれで内部的に生成され、リクエスト/レスポンス時に送信されています。このような付随情報は「ヘッダ情報」と呼ばれ、クライアント側から送信されるものは「リクエストヘッダ」、サーバ側から送信されるものは「レスポンスヘッダ」と呼ばれます。スーパーグローバル変数$_SERVER で取得できるのものはこのうちのリクエストヘッダです。ヘッダ情報は、ふだん開発者が(ましてやエンドユーザが)意識することはほとんどないので、なかなかイメージしにくい概念かもしれませんが、使い方次第ではいろいろと役立つ重要な情報を含んでいます。

たとえば、Referer ヘッダはリンク元ページの URL を含みます。これをアクセスログとして記録しておくと、自分のサイトにいったいどんなサイトがリンクしているのかを把握することができます。

また、User-Agent ヘッダはクライアント(ブラウザ)の種類を含みます。昨今では、デスクトップ端末はもちろん、さまざまな形態の携帯電話がつかわれていますが、User-Agent ヘッダを利用することで、それぞれの端末に応じた最適なコンテンツを出力できるようになります。
もちろん、リクエストヘッダは上で挙げたものばかりではありません。

ヘッダ名	概要
Accept	クライアントがサポートしているコンテンツの種類(優先順位)
Accept-Language	クライアントが対応している言語(優先順位)
Authorization	認証情報
Cookie	クライアントに保存されているクッキーデータを送信
Host	要求先のホスト名
If-Modified-Since	指定された日時以降にコンテンツが更新されている場合にのみ、サーバからデータを受信
Proxy-Authorization	プロキシサーバ用の認証情報
Range	要求するリソースの範囲
Referer	リンク元の URI
User-Agent	クライアントの種類

これらのリクエストヘッダを取得するのは、$_SERVER の役割ですが、$_SERVER を使用して特定のヘッダを取得するときには、ヘッダ名をそのまま指定するのではなく、次の規則で整形した上で指定する必要がある点に注意してください。

- 「 - 」は「 _ 」に変換
- 接頭辞として「HTTP_ 」を付加

ですから、たとえば User-Agent ヘッダを取得するコードは次のようになります。

server.php
```
<?php
    print($_SERVER['HTTP_USER_AGENT']);
?>
```

出力結果
Mozilla/5.0 (Windows; U; Windows NT 5.1; ja; rv:1.9.0.10) Gecko/2009042316 Firefox/3.0.10

なお、$_SERVER を使用して、Web サーバであらかじめ定義された「定義済みサーバ変数」を取得することも可能です。定義済みサーバ変数は、使用している Web サーバによって異なる可能性がありますが、たとえば Apache を利用しているならば、次のような情報を取得することが可能です。

変数名	概要	戻り値（例）
DOCUMENT_ROOT	ドキュメントルート	C:/ProgramFiles/Apache Group/Apache2/htdocs
GATEWAY_INTERFACE	CGI のリビジョン	CGI/1.1
PHP_SELF	実行中のスクリプト	/samples/chap2/server.php
QUERY_STRING	クエリ文字列	category=PHP
REMOTE_ADDR	クライアントの IP アドレス	127.0.0.1
REMOTE_PORT	クライアントのポート番号	2856
REQUEST_URI	指定された URI	/samples/chap2/server.php?category=PHP
SCRIPT_FILENAME	実行中のスクリプト	C:/ProgramFiles/Apache Group/Apache2/htdocs/test.php
SCRIPT_NAME	実行中のスクリプト	/test.php
SERVER_NAME	サーバ名	localhost
SERVER_PORT	サーバのポート	80
SERVER_PROTOCOL	プロトコル名前、リビジョン	HTTP/1.0
SERVER_SIGNATURE	サーバのバージョン	Apache/2.0.55(Win32)PHP/5.1 Server at localhost Port 80
SERVER_SOFTWARE	サーバのソフトウェア	Apache/2.0.55(Win32)PHP/5.1

- $_ENV

$_ENV は、サーバ側で設定された環境変数を取得するためのスーパーグローバル変数です。環境変数とは、サーバ環境固有のパラメータを表すもので、プログラム実行時に参照するパスやオプション値などを設定します。たとえば、環境変数 PATH は、コマンドラインなどでプログラムを呼び出す場合にデフォルトで検索するディレクトリを指定します。以下は、$_ENV を使用したサーバ環境変数の取得例です。

env.php

```
<?php
    print($_ENV['path']);
?>
```

- $_COOKIE

$_COOKIE は、クライアントに保存されているクッキーの値を取得するためのスーパーグローバル変数です。クッキーとは、クライアント側に保存可能な小さなテキストのことを言います。セキュリティ的な理由から、一般的にサーバはクライアント上のファイルを勝手に読み書きすることはできませんが、クッキーだけは例外です。サーバがクライアントに対して自由にテキストを読み書きすることができます。

もっとも、これだけを聞くと、どうしてクッキーが必要なのかと思われる方もいるかもしれません。クッキーの必要性を理解するには、PHP がベースとしている HTTP（Hyper Text Transfer Protocol）の制約を理解しておく必要があります。

HTTP は、クライアントからの要求（リクエスト）に対して、サーバが応答（レスポンス）を返して終わり、というシンプルなプロトコルです。つまり、同じクライアントが発したリクエストであっても、最初に発したリクエストと次に発したリクエストとの間にはなんの関係もありません。ちょっと難しい言い方をすれば、HTTP とはステートレス（状態を保存できない）プロトコルなのです。そのため、かつて開発者はページ間で情報を共有するためにさまざまな工夫を凝らさなければなりませんでした。前述したクエリ情報や隠しフィールドに、ページ間で引き渡したい情報を保存しておくという方法もありました。しかし、この方法では、扱う情報（あるいはページ数）が多くなってきた場合に、コードが煩雑になるという問題があります。

そこで登場したのがクッキーです。クッキーを利用することで、こうした「クライアント単位で維持したい情報」の管理が容易になります。それでは、実際にクッキーを利用して情報の保存/参照を行ってみましょう。

```
cookie1.php
<html>
<head>
<title>スーパーグローバル変数</title>
</head>
<form method="POST" action="cookie2.php">
E-Mail アドレス：<input type="text" name="email" size="30"
    value="<?php print($_COOKIE['email']); ?>" />
<input type="submit" value="送信" />
</form>
</body>
</html>

cookie2.php
<?php
    setcookie('email', $_POST['email'], time()+60*60*24*90);
    print('クッキーemail を保存しました');
?>
```

cookie1.php にアクセスし、E-Mail アドレスを入力してみましょう。送信をおして POST でデータが送信され、cookie2.php へページ遷移できたら、もう一度、cookie1.php1 にアクセスします。すると初回のアクセスで入力した E-Mail アドレスがデフォルト表示されていることが確認できるはずです。

クッキーをクライアントに保存するのは、setcookie 命令を使用します。setcookie 命令の一般的な構文は以下のとおりです。

setcookie (クッキー名, 値 [, 有効期限 [, 対象のホスト [, 対象のパス]]])

対象のホスト/パスを設定した場合には、該当のクッキーをどのホスト/パスから参照可能にするかを限定できます。デフォルトでは現在のホストは以下のすべてのパスから読み込むことが可能です。

有効期限は、タイムスタンプ値として指定します。タイムスタンプとは「1970 年 1 月 1 日からの経過秒」を表す値です。先ほどの cookie2.php では、現在のタイムスタンプ値を取得する time 命令を使用し、取得したタイムスタンプに「60 秒×60 分×24 時間×90 日」の値を加算することで、クッキーの有効期限を 90 日に設定しました。これにより、90 日を経過したクッキーは自動的に破棄されます。クッキーの有効期限を省略した場合には、クッキーはブラウザを閉じたタイミングで破棄され、有効期限を現在より前に設定した場合には、クッキーはその場で破棄されます。繰り返しになりますが、クライアントに送信されたクッキーは$_COOKIE で取得できます。

➢ $_SESSION

前項でクッキーを使った情報維持の方法を紹介しましたが、実はクッキーにはいくつかの問題点があります。

1.データがクライアント側で保存される

クッキーで管理されたデータはクライアント側で自由に削除したり改ざんしたりすることが可能です。そのため、クッキー情報を基にアプリケーション全体の挙動を左右するような設定を行うのは危険です。

2.実データがネットワーク上を流れる

通信経路上にリクエスト情報をロギングするような通信機器やソフトウェアがある場合には、クッキー情報が漏洩してしまう可能性があります。

セキュリティがとりざたされる昨今の情勢を考えれば、このような状態はセキュリティホールの一因となる場合もあり、好ましいことではありません。そこで、ユーザがブラウザを開いている間だけ情報を維持したいという場合には、よりセキュアな「セッション」というしくみを利用することをお勧めします。

先ほどのクッキーとよく似ていますが、異なる点が2つあります。それは、データがサーバ側で保存されるということと、ネットワーク上を流れるのはセッション ID だけ、というクッキーの問題を克服した点です。したがって、データがクライアントによって改ざん/削除される可能性は還俗としてありませんし、実データがネットワーク上を行き来しないので、データを盗聴される危険性も「相対的に」低下します。

もちろん、セッション ID を盗聴できれば、いわゆる「なりすまし」も可能です。しかし、そもそもセッション ID 自体が一時的に生成される ID 値なので、リアルタイムで盗聴を行っていない限り、なりすましを行うのは難しいでしょう。

以上、セッションのしくみを理解したところで、セッション情報の保存と参照とを行ってみることにしましょう。

```
session1.php
<?php
    session_start();
    $_SESSION['sample']='セッション情報（テスト）';
    print('セッション情報 sample が保存されました。');
?>

session2.php
<?php
    session_start();
    print('セッション情報 sample の値は「'.$_SESSION['sample'].'」です。');
?>
```

最初に session1.php にアクセスした後、session2.php にアクセスしてください。session1.php で登録したセッション情報が session2.php で参照できていれば成功です。セッションを利用するには、ページの先頭で session_start 命令を実行する必要があります。もし、ページごとにいちいち session_start 命令を実行するのが面倒ということであれば、php.ini で session.auto_start パラメータを On に設定することで、session_start 命令を省略することも可能です。ただし、session.auto_start パラメータを On にすると、セッションを利用しないページでもセッションが有効になってしまう点に注意してください。必要なページだけでセッションを有効にすれば、サーバリソースをいくらか節約することができます。
セッションを登録/参照するには、いずれも$_SESSION を使用します。登録したセッション情報を破棄するには、session_destroy 命令を使用します。

- 定数

前述したように、変数は「データの入れ物」です。入れ物ですから、当然、その中身をプログラムの実行途中で変更/編集することが可能です。一方、入れ物と中身がセットになっており、途中で中身を変更できない入れ物のことを「定数」と言います。

定数とは、「プログラム上、意味を持つ値」にあらかじめ名前を付けておくしくみだと思っておけばよいでしょう。たとえば、次の例に注目してください。

```
$price=1000;
$sum=$price * 1.05;
```

上の例は、税抜き価格$price に対して、1.05 を乗算して消費税 5%込みの支払い合計を求めるための式です。しかし、このような式がアプリケーションの各所に登場したとしたら、どうでしょう。しかも、将来的に消費税率が7%、10%と変更になったとしたら。そのたびに、アプリケーション内に散らばるすべての 1.05 を変更しなければなりません。これは面倒であるばかりではなく、修正の漏れや誤りの原因ともなり、好ましい状態ではありません。しかし、次のように記述すればどうでしょう。

```
const.php
<?php
define('TAX', 1.05);
$price=1000;
$sum=$price*TAX;
print($sum);
?>
```

define は定数を設定するための命令で、ここでは定数 TAX を 1.05 という値で設定しています。このように定数を利用していると、あとから TAX の値を変更したくなった場合に便利です。定数を設定している1箇所だけを変更すればよいからです。

また、1.05 という数値だけだと、これが消費税なのかサービス料なのか、あるいは、まったく異なる別の意味を持った数値なのか、分からないケースもあるでしょう(このような値のことを「マジックナンバー」と言います)。しかし、定数として名前付けしておけば、データの意味が視覚的にも明確となり、コードが読みやすくなるというメリットがあります。

✓ 定数の命名規則

定数の命名規則は変数に似ていますが、いくつかの点で異なりますので、注意してください。

1.定数名の先頭には「＄」は使わない
2.定数の先頭は英字かアンダースコア（ _ ）であること
3.定数名の2文字目以降は、英数字、アンダースコアのいずれかであること
4.変数名の大文字/小文字は区別されない

とくに、1と4の規則は変数のときとは異なるので注意してください。また、命名規則ではありませんが、変数と区別するために定数は「大文字」とするのが通例です。

✓ 定義済みの定数

PHP にはあらかじめ定義済みの定数が用意されています。主なものは以下のとおりです。

定数	概要
FILE	実行中のファイル名
LINE	実行中の行番号
FUNCTION	実行中の関数名（PHP 4.3.0 以降）
CLASS	実行中のクラス名（PHP 4.3.0 以降）
METHOD	実行中のメソッド名（PHP 5.0.0 以降）
DIRECTORY_SEPARATOR	ファイル区切り文字
PATH_SEPARATOR	パス区切り文字
PHP_VERSION	使用している PHP のバージョン
NULL	未定義
TRUE	真
FALSE	偽

FILE, _LINE_ など、いずれもアンダースコアが前後に2つずつ連なっていることに注意してください。DIRECTORY_SEPARATOR や PATH_SEPARATOR など環境に依存する情報を、ハードコーディングする代わりに定数で記述しておくと、サーバ環境を移行した際にコードを変更する必要がなくなります。

➢ エスケープシーケンス

PHP では、キーボードから直接表現できない特殊文字を「￥+文字」の形式で表すことができます。このような文字のことを「エスケープシーケンス」と呼びます。PHP で利用可能なエスケープシーケンスには次のようなものがあります。

文字	概要
¥r	復帰
¥n	改行
¥t	タブ
¥$	ドル記号
¥¥	円記号／バックスラッシュ
¥″	ダブルクォーテーション

エスケープシーケンスは任意の文字列に含めることが可能ですが、対象の文字列は必ず「ダブル」クォーテーションでかこまれていなければならない点に注意してください。前項でも説明したように、シングルクォーテーションの中では、文字列は記述されたそのままに解釈されます。

4.演算子

ここでは、PHP で利用可能な演算子を一覧でまとめてみました。参考までにどうぞ

代数演算子

演算子	概要	例
+	ふたつの数値の和	$x + $y
−	ふたつの数値の差	$x − $y
*	ふたつの数値の積	$x * $y
/	ふたつの数値の商	$x / $y
%	ふたつの数値の割算を行い、その余りを求める	$x % $y
++	後置加算（代入後に加算）	$i++
++	前置加算（代入前に加算）	++$i
−−	後置減算（代入後に減算）	$i−−
−−	前置減算（代入前に減算）	−−$i

代入演算子

演算子	概要	例
=	値を変数などに代入する	$x=10
+=	左辺の値に右辺の値を演算したものを、左辺に代入	$x+=10
−=		$x−=10
*=		$x*=10
/=		$x/=10
%=		$x%=10
.=	左辺の値に右辺の値を連結した文字列を、左辺に代入	$x.=″A″

比較演算子

演算子	概要	例
==	左辺と右辺が等しい場合に TRUE を返します	$x==1
===	左辺と右辺が等しく、かつ同じデータ型の場合に TRUE を返す	$flag1===TRUE
!=	左辺と右辺が等しくない、または同じデータ型でない場合に TRUE を返す	$x!=10
!==	左辺と右辺とが等しくない、または同じデータ型でない場合に TRUE を返す	$flag!==TRUE
<	左辺か右辺より小さい場合に TRUE を返す	$x<10
>	左辺か右辺より大きい場合に TRUE を返す	$x>10
<=	左辺が右辺以下のときに TRUE を返す	$x<=10
>=	左辺が右辺以上のときに TRUE を返す	$x>=10
?	（条件式）？（式1）:（式2） 条件が TRUE の場合は式1、FALSE の場合は式2を返す	$x>10 ? ″OK″ : ″NG″

論理演算子

演算子	概要	例
&& (and)	左式/右式と双方が TRUE の場合に TRUE を返す	$x==10 && $z==100
|| (or)	左式/右式いずれかが TRUE の場合に TRUE を返す	$x==1 || $x=2
xor	左式/右式いずれかが TRUE で、かつ双方とも TRUE でない場合に TRUE を返す	$x==10 xor $x==100
!	式が FALSE である場合に TRUE を返す	!$flag

ビット演算子

演算子	概要	例
&	左式/右式の双方でセットされているビットをセットする	10&1 → 1010&0001 → 0000 → 0
|	左式/右式のいずれかでセットされているビットをセットする	10|1 → 1010|0011 → 1011 → 11
^	左式/右式のいずれかでセットされており、かつ、双方にセットされていないビットをセットする	10^1 → 1010^0001 → 1011 → 11
~	ビットを反転させる	~10 → ~1010 → 0101 → -11
<<	ビットを左にシフトする	10<<1→1010<<1 → 10100 → 20
>>	ビットを右にシフトする	10>>1 → 1010>>1 → 0101 → 5

その他の演算子

演算子	概要	例
.	文字列結合。左式と右式を結合する	$memo . $title
@	式の先頭に指定することでエラーメッセージを非表示にする	@fopen(″sample.dat″)
`	バッククオートで囲んだブロックをシステムコマンドとして実行する	`dir`

演算子の優先順位

優先順位	演算子	優先順位	演算子
1	new	11	|
2	[	12	&&
3	! ~ ++ -- (int) (double) (string) (array) (object) @	13	||
4	* / %	14	? :
5	+ - .	15	= += -= *= /= .= %= &= |= ^= ~= <<= >>=
6	<< >>	16	print
7	< <= > >=	17	and
8	== != === !==	18	xor
9	&	19	or
10	^	20	,

5.条件分岐

プログラムは上から順番に実行していくばかりではありません。ユーザから入力された値、その他の条件に応じて処理を「分岐」することが可能です。PHP では分岐するための命令として、if 命令、switch 命令、そして三項演算子という3つの命令を用意しています。if 命令は、その名のとおり、「もしも～だったら・・・さもなければ・・・」という構文を作成します。

```
if.php
<?php
$x=10;
if($x==10){
    print('環境変数$x は 10 です');
}else {
    print('環境変数$x は 10 ではありません');
}
?>
```

この例では、変数$x の値が10であった場合に「変数は$x は10です」。10でなかった場合は「変数は10ではありません」というメッセージが表示されます（このとき、条件判断の＄x＝＝10 に注意してください。＄x＝10だと代入になってしまいますので必ず TRUE が返ってきて条件判断として機能しなくなります）。このように、if 命令は指定された条件式が TRUE（真）の場合に条件式直後のブロックを、FALSE（偽）の場合に else 直後のブロックをそれぞれ実行します。演算子の項でも述べましたが比較演算子は「＝」ではなく、「＝＝」なので、間違えないように注意してください。もしも「＄x＝10」とした場合、式全体としては常に TRUE になります。この程度の単純な条件分岐であれば、前節で紹介した3項演算子を使用することでよりシンプルに記述することができます。

```
<?php
    print(($x==10) ? '変数$x は 10 です。' : '変数$x は 10 ではありません。');
?>
```

また、先の例では2分割でしたが、else if ブロックを使用することで、連続して複数の分岐を表現することも可能です。

```
if2.php
<?php
$x=15;
if($x>20){
    print('環境変数$x は 20 より大きいです');
}else if($x>10) {
    print('環境変数$x は 10 より大きいです');
}else if($x>5) {
    print('環境変数$x は 5 より大きいです');
}else {
    print('その他');
}
?>
```

この例では if 命令が前方から順番に条件判断を通っていくという点に注目してください。ここでは if 命令によって分岐されたブロックは常にひとつしか実行されないという点を覚えておいてください。

さらに、これを発展させ、「条件1を満たし、かつ条件2が TRUE であるかどうかによって処理をしたい」といった複合条件の場合はどうでしょう。そのような場合には、if 命令の中にもうひとつ if 命令を記述するネスト（入れ子）

構造にするか複合条件を OR か AND で記述します。

```
if_nest.php
<?php
$x=true;
$y=true;

if($x){ //$x は真ですか？
    if($y){ //では、x が真で y が真の状態ですか？
        echo 'x と y は真の値です。';
    }else {
        echo 'x は真ですが y は偽のようです…';
    }
}
?>


<?php
$x=true;
$y=true;

if($x && $y){ //$x と$y はともに真ですか？
    echo 'x と y は真の値です。';
}else if($x && !$y){
    echo 'x は真ですが y は偽のようです…';
}
?>
```

複雑な条件分岐の場合、複合条件を OR か AND で結合してみると同じ処理をしているのにコードが短いことが分かります。可読性や処理の速度を上げるという観点から同じ処理でもコードはなるべく短く書くことが勧められています。if のネストなどは必要にせまられたときなどできるだけ使用頻度を減らしてください。それと、どうしてもネストしなくてはならないときなどは処理の内容や分岐条件の説明などを他のプログラマが見てわかるようにコメントをひと言いれてください。これだけで可読性がぜんぜんちがってきます。

6.switch 命令

以上のように、if 命令を使用することで、シンプルな2分岐命令から複雑な多岐分岐まで表現できるようになりました。しかし一、コードを見れば分かるように、if 命令では条件式の記述が冗長になりやすく、並列の関係にある条件値が見にくいという難点があります。そこで使用されるのが switch 命令です。

switch.php

```
<?php
$test='46'; //テスト結果
switch($test) {
  case ($test == 100) :
    echo '満点！！　おめでとう、あなたの未来は希望に満ちあふれています！';
    break;
  case ($test >= 80) :
    echo '良。満点めざしてがんばりましょう';
    break;
  case ($test >= 50) :
    echo '平均点。まだまだがんばれるはずです';
    break;
  case ($test >= 30) :
    echo 'どうやら追試は免れたようだな…、だが次はどうかな？';
    break;
  case ($test >= 0) :
    echo '勉強だけが人生じゃないさ！';
    break;
}
```

switch 命令では、先頭に指定された式の値に応じて処理を分岐します。それぞれの case ブロックには複数の命令を記述できます。そして break 句が来るとそこで処理をやめ switch の処理ブロックを抜けます。

7.繰り返し処理

- for 命令

for 命令とは、あらかじめ指定した回数だけ繰り返し処理を行うための命令です。数値配列の作成や編集によく使われます。たとえば、変数$i が1～5 まで変化する間、繰り返しを行いたい場合には、次のようにします。

```
for.php
<?php
    for($i=1;$i<6;$i++) {
        print("$i 番目のループです<br>");
    }
?>
```

文法は、for(初期化； 継続条件； 処理) となっています。また、初期化と処理（カウンタ処理など）は「, 」を打つことでいくつでも追加することが可能になっています。

- while 命令、do...while 命令

あらかじめ繰り返し回数が決まっている場合に便利なのが for 命令ならば、ループ内の特定の状態に基づいてループの終了を制御したい場合には、while/do....while 命令を使用するとよいでしょう。データベースからデータをもってきて展開する場合などによく使われます。

```
while.php
<?php
$i=1;
while($i<6){
    print("$i 番目のループです。<br>");
    $i++;
}
?>
```

```
do.php
<?php
$i=1;
do{
    print("$i 番目のループです。<br>");
    $i++;
}while($i<6);
?>
```

while 命令、do...while 命令の違いは do...while 命令は必ず一回はループの処理を実行するということです（ループ条件が最後についているので一回ループし終わってから条件判断するためです）。使用頻度としてはあまり高くありません。逆に while 命令では条件判断が最初にくるのでループ処理にはいらないこともあります。実際は、while 命令でことたりる場合がほとんどです。

- foreach 命令

ここまで紹介してきた繰り返し命令とはやや異なるのが foreach 命令です。foreach 命令は、連想配列配下の各要素に対して繰り返し処理を行うときにしようします。

```
foreach.php
<?php
    $data=array('山田'=>'男','掛谷'=>'女','日尾'=>'男','薄井'=>'男');
    foreach($data as $key=>$value){
        print("$key : $value <br>");
    }
?>
```

このように foreach 命令では連想配列に特化した処理を行うことができます。

- break 命令、continue 命令

ここまで繰り返し処理そのものの構文について学んできましたが、break 文と continue 命令を使用すると、指定の条件下で繰り返しを中断したり、一部の処理をスキップすることが可能になります。たとえば、以下は1～10 までの「偶数値」のみを加算するスクリプトです。

```
continue.php
<?php
    $sum=0;
    for($i=1;$i<=10;$i++){
        if($i%2!=0){ continue; }
        echo "{$i}";
        $sum+=$i;
        if($i%2==0 && $i<=9){ echo ' + '; }
    }
    echo " = $sum<br>";
?>
```

このように continue 命令を用いることで、特定条件を満たしたときに continue 命令を実行して、現在のループをスキップし、次のループを継続して実行できます。
一方、break 命令は特定条件を満たしたタイミングでループを脱出します。以下は1～100 までの値を加算し、合計値が 100 を越えたときのループ変数を出力します。

```
break.php
<?php
    $sum=0;
    for($i=1;$i<=100;$i++){
        if($sum>100){break;}
        $sum+=$i;
    }
    print('合計が 100 を越えるのは1～'.$i.'までを加算したときです。');
?>
```

8.関数

これまで array や setcokie, move_uploaded_file のようなコマンドのことを単に命令と呼んできました。しかし、これらのように与えられた入力（パラメータ）に基づいて処理を行い、結果（戻り値）を返す命令のことを PHP では「関数」と呼びます。前節で紹介した分岐命令のようなプログラムの流れを制御するための制御命令とは区別する必要があります。

PHP には、あらかじめ豊富な関数が用意されているため、プログラマはこれらの関数を組み合わせることで、自ら原始的なロジックを記述しなくても「やりたいこと」を直感的に記述できます。関数とは、与えられた入力に基づいてなにかしらの結果を返す命令のことを言います。この入力のことを引数、出力のことを戻り値と呼びます。引数が2つ以上存在する場合には、カンマ区切りで記述します。引数が存在しない場合でも、関数名の後ろのカッコは省略できない点に注意してください。また、関数によっては処理を行うのみで戻り値をもたないものがあります。

PHP で利用可能な関数はあまりに膨大なため、そのすべてを紹介することはできません。そのため今回は基本的な関数をメインに紹介していきたいと思います。

関数の呼び出し

```
void asort （array array [ , int sort_flags] ）
```

引数/戻り値のデータ型には、次のようなものがあります。前述したように、PHP はデータ型について曖昧な言語ですが、戻り値/引数のデータ型を把握しておくことは、前後の処理を記述する手がかりとして重要になってきます。また、ソースをはやく読み理解するために必要な知識です。

データ型	概要
array	配列型
bool	真偽型（TRUE/FALSE）
float	浮動小数点型
int	整数型
mixed	複数のデータ型を返す可能性がある（戻り値のみ）
object	オブジェクト型
resource	リソース型
string	文字列型
void	戻り値がない（戻り値のみ）

関数が戻り値を返さない場合、戻り値のデータ型は void となります。また、関数によっては、たとえば、処理に成功した場合は resource 型を、失敗した場合は FALSE を返すようなものがあります。このような関数の戻り値は mixed（混合型）と表記されます。

bool, resource は PHP から利用可能なデータ型です。ドキュメントによってはデータ型である int として表記されている場合もあるので注意が必要です。

➢ ユーザ定義関数

PHP では、あらかじめ用意されている関数を利用するばかりではありません。標準的な関数ではカバーされていない定型的な処理については、アプリケーション開発者が自ら提供することも可能です。このような自前の関数のことを「ユーザ定義関数」と呼びます。

以下に、ユーザ定義関数の例を示します。showString 関数は、与えられた文字列（$value）を$num 回繰り返し表示します。その際、$delim で指定された値を文字列の区切り文字として使用します。

```
usrFunc.php
<?php
function showString($value, $num, $delim){
    for($i=1; $i<=$num; $i++){
       print($value.$delim);
    }
}
showString('だるまさんが転んだ！',5,'<br>');
?>
```

ユーザ定義関数を定義するには、function 命令を使用します。function 命令の一般的な構文は次のとおりです。

```
function 関数名（仮引数1，仮引数2，… ){
      …任意の処理…
      [return 戻り値;]
}
```

仮引数とは、ユーザ定義関数の処理内で参照可能な変数のことを言います。後からユーザ定義関数を利用する場合、外部の情報をユーザ定義関数に引き渡す際に使用します。ユーザ定義関数の引数を定義するための情報、と考えてもいいでしょう。省略可能な引数を設定したい場合には、次のように仮引数の後方にデフォルト値を明記します。これによって、引数が省略された場合には、指定されたデフォルト値が使用されます。

```
usrFunc2.php
<?php
function showString($value, $num, $delim='<br>'){
    for($i=1; $i<=$num; $i++){
       print($value.$delim);
    }
}
showString('だるまさんが転んだ！',5);
?>
```

ただし、省略可能な引数は、必ず必須である引数の後方に配置するようにしてください。ユーザ定義関数が戻り値を持つ場合は、return 命令を使用して処理結果を呼び出し元に返します。return 命令が記述されなかった場合、ユーザ定義関数は戻り値を返しません。たとえば、先ほど挙げた showString 関数は戻り値を持たないユーザ定義関数です。ユーザ定義関数を呼び出すには、通常の関数と同様の記法を使用します。

➢ 変数の有効範囲（スコープ）

ユーザ定義関数が登場したところで、変数の有効範囲（スコープ）というものを意識する必要があります。

scope.php

```
<?php
$x=1;
function sample(){
    $x++;
    return $x;
}
print(sample());
?>
```

実行結果

```
1
```

一見、ユーザ定義関数によって変数$x の値が加算されて、結果としては 2 が返されそうな気がしますが、結果は予想に反して 1 が返されます。これはどういうことか。このしくみを理解するには、スコープという概念について知る必要があります。PHP では、関数内の変数を「ローカル変数」、関数外の変数を「グローバル変数」と呼びます。そして、ローカル変数とグローバル変数は、たとえ同名であっても異なるものとしてみなされるのです。

つまり、上の例で言うならば、最初に初期化された$x はグローバル変数で、次にユーザ定義関数の中で定義された$x（ローカル変数）とはまったくの別物なのです。つまり、ユーザ定義関数内の「$x++;」は、変数の暗黙的な初期値である 0 に 1 を加算しているので、結果として 1 が返されるというわけです。もしも、グローバル変数を強制的にローカル変数として利用したい場合には、global キーワードを使用します。たとえば、次のようにします。

scope2.php

```
<?php
$x=1;
function sample(){
    global $x;
    $x++;
    return $x;
}
print(sample());
?>
```

実行結果

```
2
```

今度は、ユーザ定義関数内の変数$x がグローバル変数であるとみなされるので、グローバル変数として定義された初期値 1 が生かされて、結果は 2 となります。

- **外部ファイルのインクルード**

ユーザ定義関数は、その性質上、特定のスクリプト上でだけりようするよりも、複数のスクリプトで共有することが多いものです。再利用性をたかめるという意味ではこのようなユーザ定義関数を外部ファイルとして保存しておき、個々のスクリプトから必要に応じてインクルードする（取り込む）のが好ましいアプローチでしょう。外部ファイルをインクルードするには、include_once 命令または require_once 命令を使用します。以下は usrFunc.php をそれぞれ include_once 命令、require_once.php 命令を使って書き換えたものです。included.php は、実際にインクルードされる外部ファイルです。

```
included.php
<?php
function showString($value, $num, $delim){
   for($i=1;$i<=$num;$i++){
      print($value.$delim);
   }
}
?>


include.php
<?php
include_once('included.php');
showString('だるまさんが転んだ！',5,'<br>');
?>


require.php
<?php
require_once('included.php');
showString('だるまさんが転んだ！',5,'<br>');
?>
```

include.php, require.php ともに同じ結果が得られると思います。
include_once 命令と require_once 命令はいずれもほとんど同様の働きをしますが、指定された外部ファイルが存在しなかった場合の挙動だけが異なります。include_once 関数は外部ファイルが存在しなかった場合に警告（Warning）を発するだけですが、require_once 関数はエラーとし、スクリプトの実行を中断します。

include_once 関数、require_once 関数ともに、循環呼び出しが行われた場合には、後で行われた呼び出しを自動的にスキップします。循環呼び出しとは、呼び出し元と呼び出し先とで互いに互いをインクルードすることを言います。