PHP
Hypertext
Preprocessor

Part-1

PHP をはじめよう

第 1 部

PHP

Hypertext Preprocessor

HP

Chapter-1 PHPってなに？

本章では、PHPとは何かを説明するにあたり、まず前提知識となるWWW（World Wide Web）システムと、それを構成するコンポーネントについて簡単に説明しておきます。

1.1 WWW（World Wide Web）

WWWはハイパーテキスト形式で情報を表すための、インターネットプロトコルとソフトウェアのセットです。これは1989年にCERN（欧州素粒子物理学研究所）で開発されました。現在では電子メールと並んで、インターネットにおける代表的なサービスのひとつになっています。WWWの普及により、ユーザはマウスをクリックするだけで必要な情報にたどり着けるようになりました。WWWの日本における読み方は「ダブリュー・ダブリュー・ダブリュー」のほかにも「トリプル・ダブリュー」や「ダブリュー・スリー」「ウェブ」などがあります。本書ではこれ以降、特に断りなくWWWをWebと呼ぶことがあります。

WWWはクライアントサーバ・モデルで構築されています。サーバ側ソフトウェアの双璧はIIS（Microsoft Internet Information Server）とApache Web Serverでしょう。前者はMicrosoft社が提供している商用ソフトウェアであり、Windows NT/2000/Xp Server上で動作します。一方後者は各種UNIX上で動作するフリーソフトウェアです。バージョン1.3a1からは各種Windowsでも動作するようになりました。ただし、実用的に使うためには95 / 98 / Meでは厳しいでしょう。

Apache Web Serverは、インターネット上で運用されているWebサーバの中でもトップシェアを誇るソフトウェアであり、IBMのWebSphere Application ServerやOracle Application Serverといった商用製品のベースにもなっているものです。本書でもこのApacheを使って説明していきます。

クライアント側では、Webブラウザを使って情報を閲覧（ブラウズ）します。代表的なブラウザとしては、IE（Microsoft Internet Explorer）とNN/NC（Netscape Navigator/Communicator）があり、どちらも無料で使用することができます。また最近は、Netscapeと同じレンダリングエンジンを搭載したMozillaやOperaといった選択肢も増えてきました。LinuxなどのUnix系では、w3mというテキストブラウザも知っていると非常に便利です。

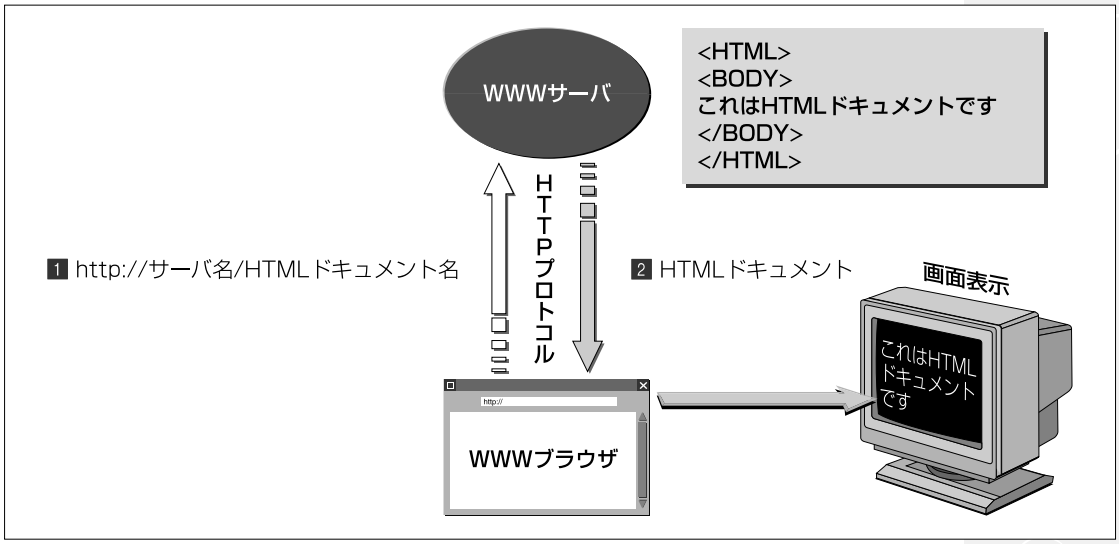
WWWではHTML（HyperText Markup Language）という言語を使用してコンテンツ（文書の内容）を表現します。ブラウザはURL（Uniform Resource Locator）を使って、サーバ上にあるいずれかのコンテンツを要求します。

URLとは、次のような形式の表記方法です。

http://WWWサーバの名前/ドキュメントへのパス

サーバ・クライアント間における情報のやりとりにはHTTP（HyperText Transfer Protocol）というプロトコル（通信規約）が用いられます。HTTPは比較的単純なプロトコルです。**図1-1**にWWWの概念図を示します（詳細については「5.3　HTTP」を参照）。

図1-1　WWWの概念図



1.2 動的なWebページ

WWWで情報発信されるページ（以下、Webページ）は、本来はサーバのディスク内に格納されている***.html**や***.htm**といった静的なファイルでした。当初はこれで十分でしたが、最近では動的なWebページのほうが当たり前になってしまいました。

たとえば、インターネット上で最もよくアクセスされるサイトであろう各種検索サイトの画面を考えてみましょう。検索した結果は毎回異なることが普通なので、一般的には検索結果を前もってファイルで用意しておくことはできません。このため、検索結果にしたがって出力画面を動的に生成するしくみが必要になります。

また、出力画面を動的に生成できるようになると、従来は専用の端末やクライアントソフトウェアを使って行なわれていたような、いわゆる事務処理をはじめとする各種適用業務を、Webサーバを経由してブラウザ上で実行できるようになります。Webサーバはブラウザからの要求を実行し（または要求を別のサーバに送ってその応答を受信し）、結果だけをブラウ

ザに送り返してやればよいわけです。クライアント側にWebブラウザを用意しておくだけで、多くの定型業務がWebベースに移行できる可能性があります。クライアント側にはほとんど費用がかかりませんから、これはやり方によっては大幅なTCO（Total Cost of Ownership）の削減につながります。

技術面から見ると、動的なWebページの実装方法にはいくつかのパターンがあります。本書の主題であるPHPの仲間（ライバル？）ともいえる、これらを簡単に紹介しておきましょう。ただし、これらの方式は排他的なものではなく相互補完的なものであり、必要に応じてこれらを組み合わせて使用することもできます。

1.2.1 クライアントサイド・スクリプト

サーバからクライアントに対して動的な画面を生成するためのスクリプト（ソースプログラム）を送り、クライアント側のブラウザがそれらを逐次解析・実行する形式です。サーバの処理は軽くなりますが、クライアント側にそのスクリプトを処理する能力が必要です。代表的なものとしてJavaScriptやVBScriptがあります。**図1-2**にその概念を示します。

1.2.2 CGI（Common Gateway Interface）

Webサーバが、URLで指定されたファイル（CGIプログラム）を外部プログラムとして起動する形式です。CGIプログラム自体は、そのオペレーティングシステムで動作可能なものであればどんなものでもかまいませんが、Perlをはじめとする各種スクリプト言語が使われることが多いようです。実行時の権限の問題を別にすれば、プログラムでできることは何でもできます。

起動されたプログラムはHTMLを生成し、その出力はWebサーバを通してそのままクライアント側に送られます。プログラムの起動はオペレーティングシステムにとって非常にコストのかかる処理であるため、CGIではWebサーバ側に負荷がかかります。一方クライアント側としてはHTMLの処理能力だけを備えていればよいので、携帯情報端末など比較的低スペックのものでも動作します。

1.2.3 サーバサイド・スクリプト

サーバサイド・スクリプトはCGIとよく似ていますが、スクリプトの実行をWebサーバプロセス自身が行うという点で異なります。このタイプにMicrosoft社のASP（Active Server Pages）があります。本書のターゲットであるPHPもこのタイプです。ApacheのDSO（Dynamic Shared Object）として組み込む場合はプロセスを生成しないため、CGIに比べてスクリプト起動時のサーバ側の処理が軽くなります。その反面、Webサーバ自身に実行ルーチンをモジュールとして組み込んでしまうため、サーバプロセス自体が大きくなる傾向があります。

図 1-2 クライアントサイド・スクリプト

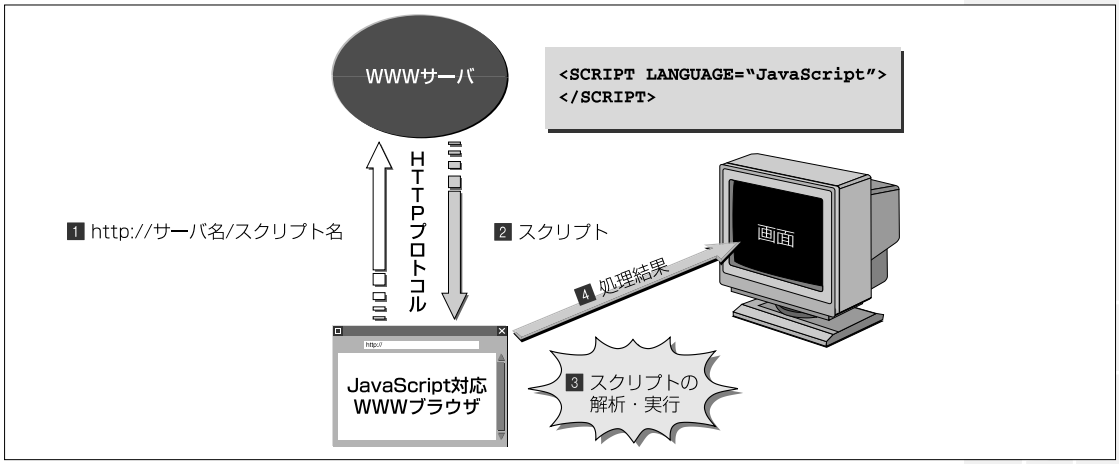


図 1-3 CGI

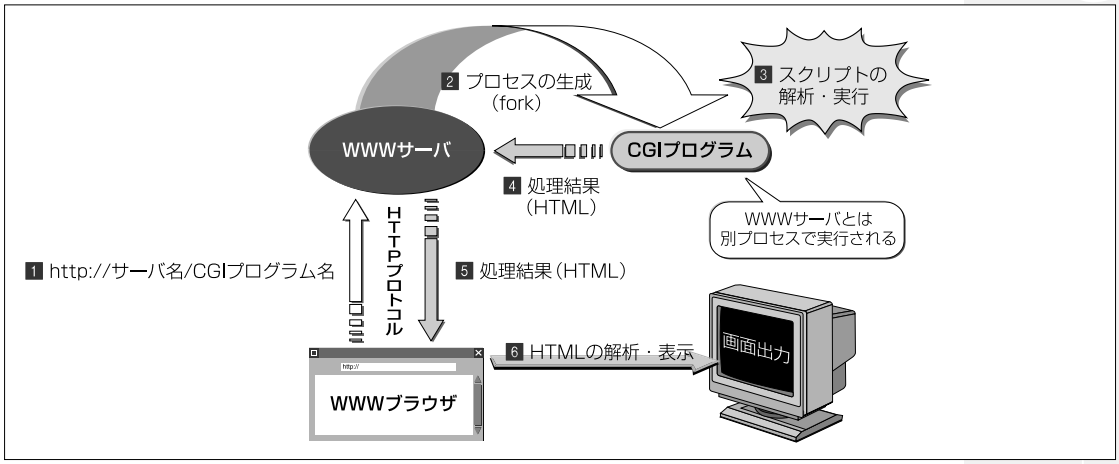
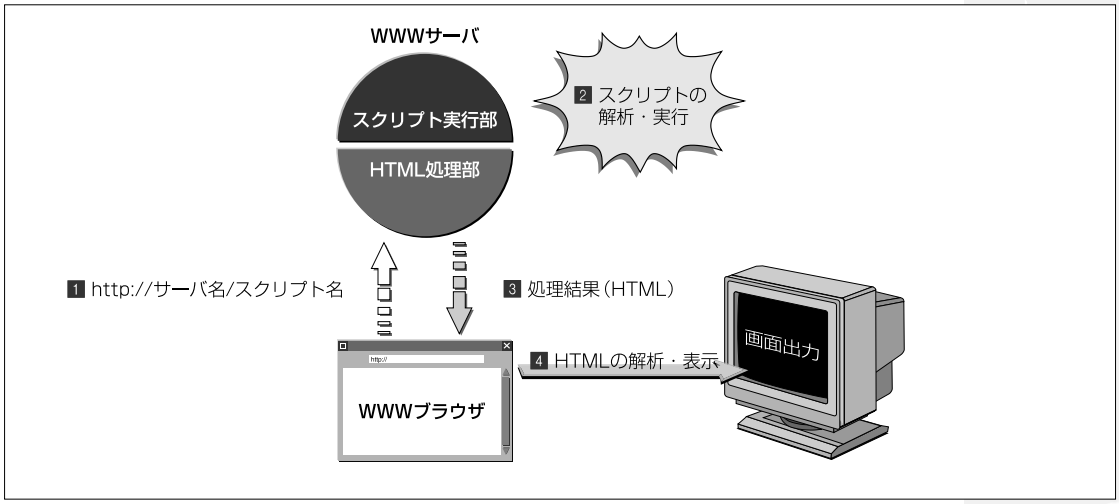


図 1-4 サーバサイド・スクリプト



1.3 PHPの概要

1.3.1 PHPとは

PHPは、HTML埋め込み型のサーバサイド・スクリプト言語です。もともとはRasmus Lerdorf氏がPerlで書いた小さなCGIラッパーであったのですが、プロセスを生成するコストを下げるためにC言語で書き直しを図ったのが最初です。さらにSQLによるDBMS（データベース管理システム）アクセスを提供するツールがリリースされてそれぞれPHP（Personal Home Page Tools）とFI（Form Interpreter）になり、それらが統合されてPHP/FI 2.0となりました。

その後、Andi Gutmans氏およびZeev Suraski氏により、PHP/FIからポーティングされたPHP3（PHP: Hypertext Preprocessor）がリリースされました。これでPHP/FIにあった不安定さがなくなり、動作も高速になりました。ほどなくしてPHP/FIのサポートが打ち切られたので、PHP/FIを使っていた人たちはこぞってPHP3に移行しました。さらに、日本人を中心とした国際化プロジェクトによりPHP3に対する国際化対応が行なわれ、現在もPHP3国際化バージョンとして商用サイトを含む各地で稼働しています。

そして2000年の5月22日に、PHP 4.0.0がリリースされました。このバージョンでは、PHP3に対する膨大なバグフィックスや機能改善が行なわれました。本書の執筆時におけるPHPの最新版はPHP 4.2.2です。このバージョンではXML、DOM-XML、XSLT、SOAP、XML-RPC、WDDX、CORBA、IMAP、LDAP、SSL、cURL、PDF、GD、ShockWaveなど各種機能をサポートしており、広範なWebアプリケーションおよびWebサービスを構築できるようになっています。

PHPのライセンスはGPL（the GNU General Public License）に準じていましたが、PHP4になってからは独自の、より緩やかなPHPライセンスに変更されました。これは商用製品に組み込みやすくするための変更だったようです。PHPライセンスについては「Appendix G」を参照してください。またバージョンごとの差異については「1.3.5 PHPのバージョン」で解説しています。

1.3.2 PHPの特徴

ここでは、開発ツールとしてのPHPの特徴について簡単に紹介します（詳細については第2章以降を参照）。

● スクリプトである

明示的なコンパイルを必要とせず、ソースを修正してすぐにテストというサイクルを繰り返すことができるので、生産性が向上します。スクリプト言語の宿命で実行のたびに文法解

析が行なわれますが、かなり高速に処理されるのであまり気にはならないでしょう。

●HTML文書への埋め込み型言語

ファイル全体をスクリプトとして作成する必要はなく、HTML文書のうちの必要な部分だけをPHPで書くといった使い方ができます。

● 確実なエラーハンドリング

エラーが発生した場合には、発生した行番号やエラー内容などを表示するためのHTML文が自動的に生成され、エラー情報がブラウザ上に表示されるのでデバッグが容易です。Cなどの汎用言語でCGIプログラムを書いたことのある人なら、このありがたみがわかるでしょう。なお、システムの本稼動後はエラーをsyslogだけに吐き出し、ユーザにはエラーを知らせないようにするといった制御もできます。

●Cライクな文法

if、for、while、do-whileなど、C言語などでおなじみの制御構文が用意されており、CやPerlでコードを書いたことのある人ならすんなりと入っていけるでしょう。printf()などのライブラリ関数も充実しています。独自のユーザ定義関数を定義したり、ほかのソースライブラリをインクルードすることも可能です。

●Perlライクな機能

Perlライクな文字列処理や関数群を豊富に備えており、日本語を使った正規表現[*1]も利用できます。リスト変数、連想配列、多次元配列もサポートされています。また限定的ながら、クラスと継承もサポートされています。

●Apacheのモジュールとして動作

CGIとは異なりApacheのモジュールとして動作させることができるので、無駄なリソースを消費せず、処理も高速です。必要に応じてCGIとして動作させることもできますし、単独のコマンドとしてcronなどに組み込んで使うこともできます。

● 各種データベースへのインタフェース

Oracle、Sybase、Informixをはじめとする商用DBMSや、PostgreSQL、MySQLといったオープンソースのDBMSへのインタフェースを標準で備えています。PostgreSQLは本書のもうひとつの柱でもあり、第2部で詳しく解説します。

*1
PHPでは、従来からサポートしているPOSIX1003.2互換の正規表現に加え、バージョン3.0.9よりPerl互換の正規表現を使うこともできるようになりました。

1.3.3 PHPとほかの言語との比較

動的なHTMLを作成するための処理系をすでにいくつか紹介しましたが、ここではPHPとそれ以外の処理系との違いという観点から、もう少し詳しく説明します。

●JavaScriptとPHP

JavaScriptはクライアントサイド・スクリプトの代表的なものです。これは、動的な画面を生成したり、マウスのイベントをハンドリングするためのJavaScriptと呼ばれるソースプログラムを埋め込んだHTMLファイルをクライアント側に送り出す方式です。ブラウザはそ

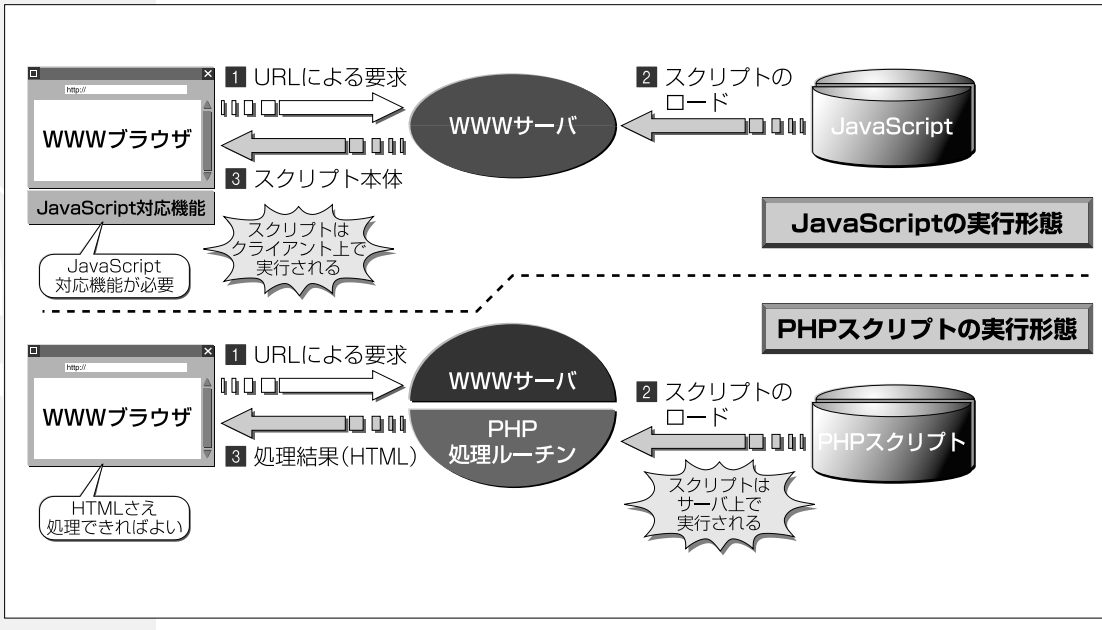
れらを逐次解析・実行します。HTMLファイルは前もってサーバ側に静的に格納されている場合もありますが、CGIなどほかの方式と組み合わせることにより、動的に生成される場合もあります。

JavaScriptのメリットは、入力イベントをクライアント側である程度リアルタイムにハンドリングできることです。たとえば、マウスカーソルが特定の領域に入っているあいだ、特定のウィンドウを表示するといったことができます。アクションを起こすにあたってサーバへの通信が発生しないので、即座に画面を変化させることができます。

デメリットとしては、ブラウザそのものがその言語をサポートしていなければならないため、ブラウザによっては意図したものが表示されないといった問題が発生することが挙げられます。また、スクリプトそのものがクライアントまで転送されるので、スクリプトのサイズによっては通信のオーバーヘッドが大きくなることも考えられます。

PHPの場合、PHPスクリプトがHTMLのみを出力している限りはブラウザを選びません。このためPDA（携帯端末）など比較的低スペックのクライアントに対してもサービスを提供できます。また、大きなスクリプトを実行した場合でも、クライアント側には実行結果だけが送信されるので無駄な通信のオーバーヘッドがかかりません。逆にいえば、それだけサーバ側の処理能力が要求されるということになります。

図1-5　JavaScriptとPHP



●CGIとPHP

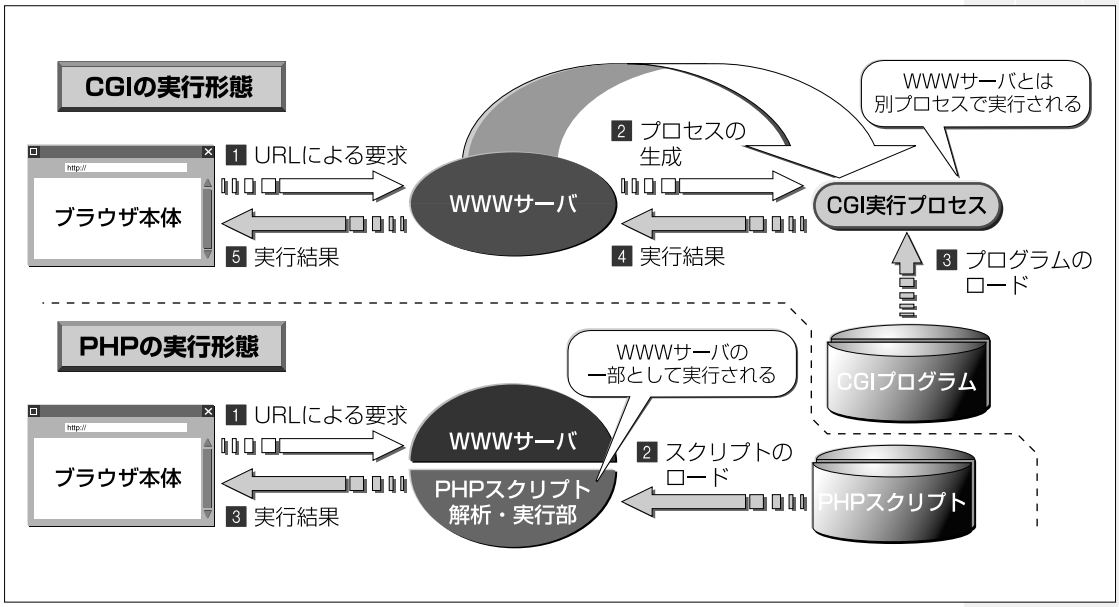
CGIは、動的なHTMLを出力するような独立したプログラムを何らかの言語で前もって作っておき、クライアントからの要求に応じてそれらをWebサーバから動的に起動する形態です。CGIプログラムは通常のプログラム[*2]ですから、権限などの問題を別にすれば、その言語で記述できることであればどんなことでもできます。ただしひとつのリクエストにつき最低でも1プロセスが発生するので、アクセスが集中した場合にはサーバマシンの負荷が非常に高くなります。また、HTTPを直接ハンドリングしてやらなければならないなど、PHPと比較すると多少高度な知識が要求されます。

PHPをはじめとするサーバサイド・スクリプトは、CGIに比べればサーバマシンにプロセスの起動/終了に関する負荷がかからないので、軽量であるというメリットがあります。しかし、Webサーバプログラム自体にスクリプトのインタプリタ機能をモジュール（コンパイル済みのオブジェクト）というかたちで組み込む必要があるため、Webサーバ自身の実行イメージのサイズが大きくなります。またスクリプトエンジン部分で万が一致命的なエラーが発生すると、Webサーバもろとも異常終了してしまうことがあるなどのデメリットもあります。

WebシステムにはHTTPプロトコル（「5.3　HTTP」で解説）にしたがった各種の約束ごとがありますが、PHPではこれらを隠蔽して、比較的簡単にコーディングできるような工夫がなされています。CGIに比べればプログラミングを手軽に始めることができるでしょう。各種サポート関数群の豊富さも他の追随を許しません。

*2
Perlなどのスクリプト言語で作られたCGIプログラムのことを、特にCGIスクリプト（または単にCGI）と呼びます。

図1-6　CGIとPHP



これらの処理パターンをまとめると、**表1-1**のようになります。

表1-1 動的Webページの実装方法

タイプ	クライアントサイドスクリプト	CGI	サーバサイドスクリプト
メリット	・サーバの負荷が軽い ・GUIに関してきめ細かい記述が可能	・ブラウザを選ばない ・記述言語を選ばない ・サーバの種類を選ばない	・サーバの負荷がCGIより軽い ・プログラム作成がCGIより容易
デメリット	・ブラウザを選ぶ	・サーバに負荷がかかる ・プログラム作成がやや難しい	・サーバプログラムが大きくなる ・インストールがやや難しい
実装例	JavaScript VBScript	Perlがよく使われる	PHP ASP

1.3.4 PHPとデータベース

Webの枠組みを使って実用的な業務を行なうには、何らかのデータベースと組み合わせて使用するのが一般的でしょう。PHPではこのあたりのことも考慮されており、商用を含む各種のDBMS（DataBase Management System ― データベース・マネージメント・システム）を直接呼び出すための豊富なインタフェースを備えています。PHPを利用すれば、WebとDBMSを連携させたアプリケーションを比較的簡単に作成できます。DBMSとの連携については第5章でその概念を紹介しています。また第2部ではDBMSとしてPostgreSQLを例にとり、実践的な解説を行なっています。

1.3.5 PHPのバージョン

本項では、PHPにおける各バージョン間の相違点などについて解説します。まず気になる日本語の使用についてですが、PHP3国際化バージョンについてはまったく問題ないといってよいでしょう。なお、PHP3については国際化対応版がphp-3.0.18-i18n.tar.gzまでで開発が終了しています。本書でも、PHP3については特に言及しませんのでご了承ください。

本書のターゲットとなるPHP4については、PHP 4.0.6からmbstringモジュールが、さらにPHP 4.2.0からmbregexのコードがmbstringモジュールにマージされ、やっとデフォルトで日本語が使えるようになりました。その後も日本のボランティアの方々による安定化や機能改善のための作業が進んでいます。

マルチバイト対応機能に関する概要

・PHPスクリプトファイルで使用する文字コードとしてEUC、SJIS[*3]、UTF-8を使用できます。

・HTTP出力で使用する文字コードとしてEUC、SJIS、JIS、UTF-8を使用できます。

*3 藤本氏提供のパッチによりSJIS対応を実現しています。付属CD-ROMに収録してあるRPMパッケージにはこのパッチがあたっています。このパッチは、PHP 4.3.0で正式にマージされる予定です。

・HTTP入力データ（POST/GET/COOKIEで渡されたデータ）を内部コードへ自動変換可能です。
・内部コードにはEUC、UTF-8を指定可能です。
・メールにも日本語およびMIMEサブジェクトを使用できます。
・HTTP出力のContent-Typeがtext/htmlの場合、適切なcharsetが自動的に指定されます。
・configureの際に「--enable-mbstring」および「--enable-mbstr-enc-trans」を追加することにより日本語対応版を作成できます。
・configureの際に「--enable-mbregex」を追加すると、マルチバイト文字列対応の正規表現が使えるようになります。

PHP4では、PHP3に対する膨大なバグフィックスや機能改善に加え、以下のような機能が新たに追加されています。

PHP4で追加された主な機能

・Zend（http://www.zend.com）構文解析エンジンの採用

劇的なパフォーマンスの向上に加え、参照回数のカウント、高度なオブジェクトのサポート、新しいブーリアン型と拡張性など、いくつかの重要な言語機能が提供されます。

・サーバ抽象レイヤー

PHP4の内部ではWebサーバ依存のコードがなくなり、Webサーバへのインタフェースは薄い抽象レイヤーを通して行なわれます。これにより、異なったタイプのWebサーバのサポートが容易になりました。PHP 4.2.xではApache 1.3.x、Apache 2.0、ActiveScript（ASP）、ISAPI（IIS）、 CGI、CLI（Command Line Interface）、TUXなど、19種類のWebサーバーAPI（SAPI）がサポートされています。

・セッション管理機能のサポート

PHPlibでサポートされていたHTTPセッション管理機能がネイティブに組み込まれました。詳細は第4章で解説します。

・UNIX配下での汎用ビルド処理

PHPモジュールを動的に、かつ比較的簡単に作成できるようになりました。

・より簡単で、パワフルな構成管理

php.iniにおけるほとんどの構成ディレクティブが、実行時にも制御できるようになりました。ただし対象プラットフォームとしては、Apacheモジュールを使用する場合（Apacheの構成ファイル経由）、またはWin32を使用する場合（Windowsのレジストリ経由）のみとなります。

PHP3からの非互換性

すでにPHP3を使っておりPHP4への移行を計画している人は、以下の部分が非互換となっているので注意してください。

・スタティック変数およびクラスメンバの初期化では、スカラー値のみを受けつけるようになりました（PHP3では、有効な評価式であればどんなものでも大丈夫でした）。
・breakとcontinueの適用範囲は、インクルードされたファイルまたはeval()された文字列の内部にかぎられるようになりました。
・requireされたファイルからのreturn文は動作しなくなりました。この機能を使いたい場合は、代わりにinclude()を使ってください。
・unset()は関数ではなく文（ステートメント）になりました。
・引用符でくくられた文字列の内部における {$ という文字の並びはサポートされません。PHP3で print "{$somevar"; という記述をしていた場合、これは { という文字と$somevarの中身を表示しますが、PHP4のパーサ（構文解析部）であるZend配下ではパースエラーになります。
・short_tags()関数はもはや動作しません。実行時にPHPの短いタグ（<?～?>）の振る舞いを変更できるのは構成パラメータによる方法だけです（.htaccessの変数は正しく動作します）。
・PHP3の動的拡張（Windows上のphp3_*.dll）は、PHP4では使用できません。
・文字列 "0" は空文字列であるとみなされます。

Hypertext Preprocessor

HP

Chapter-2 PHPでスクリプトを書こう

2.1 はじめに

本章では、まずPHPによるスクリプトプログラミングの基本を紹介し、そのあとで言語仕様について説明します。本章の例題を順に試していくことで、PHPの初心者にも雰囲気をつかんでもらえるように構成してあります。すでにいずれかのプログラミングに精通している人は、この章を読み飛ばしても差し支えありません。

本章の例題を試してみたい人は、前もって第4部を参照のうえ、PHPの動作環境を構築しておいてください。動作確認の際、短いスクリプトについてはコマンドライン版を使うとより手軽に確認できます。ただしシェルのコマンドライン上で実行するため、生のHTMLコードがそのまま出力されます。

PHPの文法はC言語に似せて作られており、一部JavaやPerlの構文を取り入れています。説明の中でHTMLのタグが出てくることがありますが、HTMLの文法説明は他書に譲ります。ただしHTMLのフォームに関しては第5章で解説しています。

注意

ここで紹介するサンプルスクリプトの拡張子は.phpになっています。本書のサンプルプログラムは、PHP 4.1.0で導入されたスーパーグローバル変数($_SERVERなど)を使って書かれているため、PHP3やPHP 4.1.0以前ではもはや動作しません。ご了承ください。

2.2 Hello, World!

最初に紹介する例題は、Hello, World!という1文を出力するプログラムです。PHPではリスト1-1のようになります。

リスト1-1 test1.php

```
<?
    print "Hello, world!¥n";
?>
```

ここではprintという命令を使って文字列を出力しています。PHPではスクリプトは **<?php** で始まって **?>** で終ります。また、各実行文はセミコロン（;）で終ります。画面に出力する改行コードは **¥n** で表します。スクリプトファイルを作成する際は、拡張子を **.php** としてください。拡張子を間違えると、Webサーバはこれをphpスクリプトとして認識せず、スクリプトの中身がそのまま出力されてしまったりします。また、Apacheの設定を誤った場合も同様の結果になることがあります。プログラムの中身を誤って外部にさらしてしまうことはセキュリティホールとなるので気をつけてください。

文字列定数を定義する場合、文字列全体を単一引用符（'）または二重引用符（"）でくくって指定します。引用符を忘れた場合も見かけ上同じ動作をしているように見えますが、内部的な動作は異なります。そのような書き方は推奨されておらず、将来動かなくなる可能性もあるので、避けておいたほうが無難です。

2.3 ブラウザから実行する

ここで実行環境について触れておきます。UNIX環境ではroot（システム管理者）権限で通常の作業やプログラミングを行なうのは好ましくない[*4]ので、ここでは一般ユーザhottaで作業しています。ユーザ名は自分の環境に合わせて適宜読み替えてください。ホームディレクトリ/home/hotta直下にpublic_htmlという名前のサブディレクトリを作り、その中に作成したスクリプトを置くようにします。

スクリプトの実行のやり方は主に2種類あります。まずはブラウザからアクセスする方法です。

```
http://localhost/~hotta/1-2/test1.php
```

というURLでアクセスすると、（Apacheのデフォルトの設定では）/home/hotta/public_html/1-2/test1.phpが呼び出され、ブラウザの画面には**図1-7**のような出力結果が表示されます。

図1-7 test1.php の実行結果

*4 システム管理者権限では一切の保護機能が働かないので、操作ミスをした場合など、システムに重大な影響を及ぼす可能性があります。たとえば重要なファイルを消してしまうと、最悪の場合、OSの再インストールとなります。

Column Windowsのエディタを使うと……

UNIX初心者にとってviは鬼門です。特に初学の人ほどviを毛嫌いする傾向があるように思えます（その気持ちもわかります）。でどうするかというと、Windows上のエディタでスクリプトを書いてftp[*5]で転送したり、Samba[*6]経由でWindowsから直接スクリプトをいじったりする人[*7]を往々にして見かけます。PHPにかぎらず、UNIX系のプログラミングにおいてこれをやってしまうと、日本語を使う際に結構ハマる場合があります。主な原因は、OS間の文字コードと改行コードの違いです。

日本語コードはShift-JIS、JIS、EUC、Unicodeというように、複数存在しています。ところが、全角文字に割り当てられる2バイトコードの関係から、UNIX上のパーサ（構文解析部）はEUCコードを前提としたもの、もしくは日本語を考慮していないが、EUCだとたまたま動いているように見えるものが少なからずあります。Linuxの日本語環境も、デフォルトではEUCが想定されています。しかしWindowsやMacintoshでコンピュータを使い始めた人はShift-JISの世界しか知らない（というか、そもそも文字コードの意識が欠けている）ことが多いので、Shift-JISで書いたプログラムをパーサに与えてしまい、妙な文法エラーに頭を悩ませることになります。

また改行コードについてもDOS/Windows（CR,LF）、UNIX（LF）、Mac（CR）とまちまちであり、UNIXのツールによってはこれらをうまく自動識別できないものも多いのです。一番ありがちなのが、シェルスクリプトの1行目に“#!/usr/bin/perl”などと正しいパスを書いているのに、改行コードがCR+LF（0d,0a）になっているため、システムがシェルコマンド（#!/usr/bin/perl+0x0d）を見つけられずに“No such file or directory”になってしまうというものです。

筆者は何度かこのようなことで痛い目にあってから、秀丸エディタ[*8]におけるファイル保存時の改行コード指定機能や文字コードの自動判別機能を活用するようになりました。でも一番よいのはviの壁を越えることです。たとえばviクローンのjvim[*9]では、挿入モードのままでカーソル移動ができるなどかなり使いやすくなってきているので、PHPに触れる今回をきっかけにして、禁断のviに手を染めてみてはいかがでしょうか。

さらに、漢字コードを変換するnkfと漢字コードを判定するkccは覚えておいて損はありません。たとえばファイルをEUCコードに変換するには

```
nkf -e 変換対象ファイル > 出力ファイル
```

とします。また、すでに存在するファイルの文字コードを調べるには

```
kcc -c ファイル名
```

とします。詳しくはmanコマンドによるオンラインマニュアルを参照してください。

*5 File Transfer Protocol（ファイル転送プロトコル）を使用してファイルを転送するプログラム。改行コードの変換や文字コード変換機能を持つものもあります。

*6 UNIX（互換OS）上で動作する、LanManager互換ファイルサーバ。UNIX（互換OS）をWindows NTに見せることができます。

*7 意味がわかってやっているのであれば、特に問題はありません。便利なので筆者もよくやります。

*8 Windowsでは定番のテキストエディタ。
http://hide.maruo.co.jp/

*9 viクローンのvim（VI Improved）を日本語化したもの。起動後に:hを入力すると、日本語のオンラインマニュアルを参照できます。

環境によっては上記のようにならず、以下のようなエラーが表示される場合があります。

```
Forbidden

You don't have permission to access /~hotta/1-2/test1.php on this server.

---------------------------------------------------------------------------
Apache/1.3.26 Server at star.example.com Port 80
---------------------------------------------------------------------------
```

これはLinux（UNIX）レベルのファイルアクセス権の問題です。Vine Linuxでadduserコマンドを使ってユーザを作成した場合、デフォルトではホームディレクトリのパーミッションは

```
hotta@star ~$ ls -ld /home/hotta
drwx------   41 hotta    hotta        4096 May 30 17:08 /home/hotta/
```

のようになり、本人のみしか中に入れないようになっています。一方、本書推奨の設定では、Apache（およびApacheのプロセスの一部として動くPHP）はapacheユーザ権限で動きます。apacheユーザには /home/hotta ディレクトリに対する通過権がありませんから、当然その配下にある public_html/1-2/test1.php にアクセスできず、上記のようなエラーとなります。このスクリプトを実行できるようにするためには、少なくともapacheに対して

```
hotta@star ~$ chmod 701 /home/hotta
hotta@star ~$ ls -ld /home/hotta
drwx-----x   41 hotta    hotta        4096 May 30 17:08 /home/hotta/
```

のように、otherに対して/home/hottaへの通過権を与えてやる必要があります。ファイルのアクセス権は、そのファイルだけに関する権限だけでなく、/（ルートディレクトリ）からそのファイルへたどり着くまでのすべてのディレクトリに通過権が必要なことを覚えておいてください。

2.4 コマンドラインから実行する

PHPスクリプトはコマンドラインからも実行できます。以下のことを試してみてください。

```
hotta@star ~$ cd public_html/1-2
hotta@star ~/public_html/1-2$ php -q ./test1.php
Hello, World!
```

phpコマンドは、本書の推奨手順では/usr/bin/phpとしてインストールされます。-qオプションは、Webシステムに必要なHTTPヘッダ（後述）の出力を抑制するオプションです。この動かし方を知っておけば、ちょっとしたプログラムなら非常に簡単にテストできます。また、test1.phpの1行目にPHPへのパス名を記述し、test1.php自体に実行権を与えてやれば、スクリプト単体でも実行できます。

```
hotta@star ~/public_html/1-2$ cat test1a.php
#!/usr/bin/php -q
<?php
print "Hello, World!¥n";
?>
hotta@star ~/public_html/1-2$ chmod +x test1a.php
hotta@star ~/public_html/1-2$ ./test1a.php
Hello, World!
```

このような手法により、たとえばバッチ処理をphpスクリプトで書いておき、cronに登録して自動実行させることなども広く行なわれています。シェル（bash）より豊富な機能が使え、DBMS（データベース、後述）にも簡単にアクセスできるので、非常に有用です。

ただし、気をつけなければならないのは、Vine LinuxにおけるX Window標準のターミナルエミュレータであるGNOME Terminal上でphpコマンドを使う場合、出力するコンテンツに日本語が含まれていると文字化けする場合があることです。これは、phpからの出力をSJISコードにしてある場合です。GNOME Terminalでは出力コードはEUCが前提となっています。この場合は、php.iniのmbstring.http_outputを**EUC-JP**にしてください。SJISのままでテストしたい場合は、いずれかの端末から**kterm &**でktermを起動し、Ctrl＋［中ボタン］で「VTオプション」メニューを表示し、「シフトJIS漢字モード」にしてからやってみてください。WindowsからTeraTerm経由でシェルを使う場合は、**設定(S)－端末(T)**で「漢字（受信）」と「漢字（送信）」を適宜mbstring.http_outputの指定と合わせてください。

ところで、前記のtest1.phpの例は、実はあまり適切ではありませんでした。というのは、「Hello, world!」と出力するだけならわざわざPHPの命令を使うまでもなく、「Hello, world!」という内容のテキストファイルを用意してやれば十分だからです。

Column

w3mを活用しよう

ブラウザからのアクセス、およびコマンドラインからのアクセスの両方の利点を兼ね備えるのが、w3mというツールです。これはVine Linuxなら標準でインストールされるテキストブラウザ兼ビュワーです。使い方は簡単で、コマンドラインでURLを引数として起動するだけです。まず、単純にURLを指定してみましょう。

```
hotta@star ~/public_html$ w3m http://
localhost/~hotta/1-2/test1.php
```

▼w3mのブラウズ画面

リンクや入力エリアがあるようなHTML文書でも、ちゃんとブラウザとして使えます。インターネット上のいろいろなサイトを指定してみてください。

入力エリアのあるような画面では、タブキーを押すと入力エリアにカーソルが行きます。そこで[Enter]を押すと、画面の左下に入力エリアが現れ、そこで初めて入力モードになります。w3mの入力モードでは、シェルのコマンドラインと同じく、上下矢印を使った入力ヒストリの呼び出しや入力値の編集などが行なえます。さらにファイルアップロードの対象ファイル選択時（後述）には、タブキーを押すことで候補のファイル一覧が表示されます。

ブラウザの画面で（大文字の）[H]キーを押すとキー入力関連のヘルプを表示し、[o]キーを押すとオプション変更の画面になります。またw3m --help でコマンドラインオプションの一覧が表示されます。

次に、-dumpオプションを使ってみましょう。

```
hotta@star ~$ w3m -dump http://
localhost/~hotta/1-2/test1.php
Hello, World!
```

これは、実際にWebサーバにアクセスしますが、結果を標準出力に表示します。ここでURLではなくファイル名で指定すると、単にビュワーとして動作します。

```
hotta@star ~$ w3m -dump ./public_html/
1-2/test1.php
<?php
print "Hello, World!¥n";
?>
```

test1.phpはHTMLではないので単に中身がそのまま表示されてしまいましたが、HTMLの文法に則ったファイルを指定すれば、ちゃんとHTMLを解釈して表示します。また -dumpオプションでは出力時に文字コードを適切に変換してくれるので、Shift-JISで書かれたHTMLファイルを表示しても文字化けしないで正しく表示されます。

w3mは軽量ながら、プロキシやSSLにも対応し、最近のバージョンではインラインの画像表示までやってくれる優れものです。ぜひman w3mを一読して、w3mをマスターしてください。

2.5 変数を使う

次は単純な計算を行ない、その答えを変数に格納してみましょう。**リスト1-2**をご覧ください。

リスト1-2 test2.php

```
<?php
$a = 1 + 2;
print "1 + 2 = ".$a."¥n";
?>
```

変数とは、値を入れるための入れ物のことで、プログラムを組む際にはなくてはならない要素のひとつです。PHPでは変数名の前に**$**(ドル記号、ダラー)をつけてアクセスします。変数名は後述する命名規則にしたがって、プログラマが自由につけることができます。この例では $a という変数を定義しています。ピリオド(.)は文字列の結合を指示する演算子です。**+** を使うと数字の足し算という意味になるので注意してください。

図1-8 test2.phpの実行結果

```
hotta@star ~/public_html/1-2$ php -q test2.php
1 + 2 = 3
```

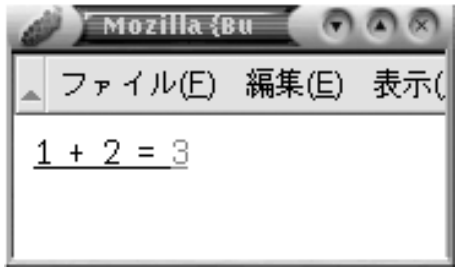
2.6 HTMLに埋め込んで使う

通常、WebコンテンツはHTML言語で記述します。PHPはHTMLの中に部分的にスクリプトを埋め込むことができるという点に大きな特徴があります。**リスト1-3**をご覧ください。

リスト1-3 test3.php

```
<HTML>
<BODY>
<U>1 + 2 = <FONT COLOR="red"><?php print 1 + 2; ?></FONT></U>
</BODY>
</HTML>
```

図 1-9　test3.php の実行結果



このように、HTML ドキュメントの中の特定の部分だけを PHP で書くことができます。PHP の実行時には「解析モード」とでもいうべき概念があります。通常（HTML）モードでは書いてある内容（HTML コード）がそのまま出力されますが、**<?php** に出会うと PHP モードに切り替わり、スクリプトの解釈と実行が行なわれます。**?>** に出会うとまた HTML モードに戻ります。

2.7 すべてを PHP スクリプトの出力として書く

test3.php の別解として、**リスト 1-4** のように書くこともできます。

リスト 1-4　test31.php

```
<?php
$result = 1 + 2;
print   "<HTML>¥n"
    .   "<BODY>¥n"
    .   "<U>1 + 2 = <FONT COLOR=¥"red¥">$result</FONT></U>¥n"
    .   "</BODY>¥n"
    .   "</HTML>¥n";
?>
```

文字列の中では 1+ 2 は単なる文字列として評価されて演算は行なわれないので、あらかじめ計算結果を変数$result に入れています。また、複数の文字列を連結するための . 演算子を使っています。文字列の中で二重引用符（" ）を使う場合は、**¥"** と書く（¥でエスケープする）必要があります。**¥n** は改行を表します。

ふだんから HTML でコンテンツを書いている人なら知っていると思いますが、各行につけている改行コードは、単にブラウザで「ソースの表示」を行なった際の読みやすさのために付加しているもので、HTML としては何ら意味を持ちません。ただ、スクリプトが大きくな

ってくるとデバッグが大変になってくるので、日頃からなるべくソースプログラムのみならずHTMLコードも、読みやすく書く習慣を心がけておいたほうがよいと思います。

文字列内部に変数を埋め込む場合は二重引用符でくくらなければなりません。単一引用符でくくることもできますが、この場合は変数展開（実行時に変数が実際の値に置き換えられること）が行なわれず、意図した結果が得られない場合もあるので、うまく使い分けてください。

次に、上記とはちょっと毛色の違う方法についても紹介しておきましょう。

2.8 ヒアドキュメントを使う

前記のスクリプトは、さらに**リスト1-5**のようにも書けます。

リスト1-5 test32.php

```
<?php
$result = 1 + 2;
print   <<<EOF
<HTML>
<BODY>
<U>1 + 2 = <FONT COLOR="red">$result</FONT></U>
</BODY>
</HTML>
EOF;
?>
```

この書式は「ヒアドキュメント」と呼ばれるものです。**<<<**の直後に任意のIDを置き、その次の行から複数行に渡る文字列を列挙します。IDだけの行が文字列の終了を表します。文字列は二重引用符で囲まれたように扱われ、変数名は値に展開されます。ただし " をエスケープする必要がなくなるので、かなり記述が見やすくなります。

2.9 フォーム画面を作る

次は、多少なりとも実用的なスクリプトを作ってみましょう。筆者は昭和から平成に変わる時点で日本国内にいなかったせいか、平成といわれてもいまだにピンときません。そこで、西暦を入れるとそれが和歴の何年にあたるのかを表示するプログラムを作ってみることにしました。

まず入力するための部分は**リスト1-6**のようになりました。入力するだけならHTML文法の範囲だけで書けるので、これはスクリプトとは呼びません（もちろんこの部分をPHPで書いてもかまいません）。

リスト1-6　test4.html

```
<HTML>
<BODY>
<FORM METHOD="GET" ACTION="test4.php">
西暦<INPUT TYPE="text" NAME="yyyy">年
<INPUT TYPE="submit" VALUE="送信">
</FORM>
</BODY>
</HTML>
```

これを

```
w3m http://localhost/~hotta/1-2/test4.html
```

として実行すると、**図1-10**のような入力エリアと送信ボタンが表示されます[*10]。この例題で、初めて漢字が出てきました。漢字部分が化けたりした場合、スクリプトの文字コードがEUCになっているか、またブラウザの文字コードセットが自動認識になっているかを確認してください。

*10
この例では、w3mによる出力結果を示しています。mozillaやIEなどを使用する場合は、URLとして引数をそのまま入力してください。なお、使用するブラウザによって出力が異なる場合があります。

図1-10　出力結果

HTMLの<FORM>タグについては「5.1　フォーム」で詳しく説明するので、ここでは簡単に触れておきます。<FORM>～</FORM>タグで囲まれた範囲が入力エリアとなり、この中のNAMEで定義された各種変数が、<FORM>タグのACTION属性で指定されたURL（ここではtest4.php）に渡されます。ACTION属性で指定するURLは、少なくとも変数を取り扱うことのできるプログラムでなければ意味がありません。

<INPUT>タグは各種入力フィールドを生成します。TYPE=textはテキストフィールド（文字列の入力エリア）を表示します。ここで入力された文字列は、NAME属性で指定したyyyyをキーとする $_GET という配列に格納され、PHPスクリプト側では $_GET['yyyy']

として参照できます[*11]。FORMタグの入力メソッドがPOSTの場合、格納される配列名も$_POST となります。また両者を区別しない $_REQUEST という配列もあり、この中にはGET、POST両メソッドによる入力値が格納されます。

ここで配列とは、ひとつの変数名で複数の値を保持するためのしくみです。C言語をはじめとする一般のコンパイラ言語[*12]では、配列は変数名に添字（何番目の要素かを指示するための数字、インデックス）をつけてアクセスします。PHPではさらに、添字は数字に限らず文字列でもかまいません。

TYPE=submitは送信ボタンを作ります。ユーザはテキストフィールドに西暦を入力して送信ボタンを押すことにより、<INPUT>タグで指定した入力値をACTION属性で指定したtest4.phpに渡すことができます。

2.10 フォームから変数を受け取る

では、入力エリアに 2000 とタイプして（w3mの場合は[Enter]で確定し）、「送信」を押してみましょう。入力した文字列は $_GET['yyyy'] で受け取ります。受け取り側のスクリプトを**リスト1-7**に示します。

リスト1-7 test4.php

```
<HTML>
<BODY>
<?php
$meiji = $_GET['yyyy'] - 1866;
$taisho = $_GET['yyyy'] - 1911;
$showa = $_GET['yyyy'] - 1925;
$heisei = $_GET['yyyy'] - 1988;
print "西暦{$_GET['yyyy']}年は、";
print "明治{$meiji}年、大正{$taisho}年、<BR>¥n"
. "昭和{$showa}年、平成{$heisei}年にあたります。<BR>¥n";
?>
</BODY>
</HTML>
```

内容は一目瞭然、入力された数値から特定の定数（変数と反対で、変化することのない実際の値のこと。即値とも呼ばれる）を引いて、それぞれ各和暦の値として表示しているだけです。= は数学では「左辺と右辺は等しい」という方程式を表しますが、PHPの場合は「左辺値に右辺値を代入する」という意味になります。PHPでは、演算子についてもここで使用している - をはじめ、必要なものはほとんど揃っています。詳細は「3.20　演算子」で解説します。

二重引用符で囲まれた文字列の中に変数名を直接記述する場合、PHPから見ると変数の名前が不明確になることがあるので注意してください。たとえば「明治$meiji年」のように変

*11 PHP 4.1.2までは入力値をそのまま$yyyy というかたちで受け取ることができましたが、PHP 4.2.0からはセキュリティへの配慮のため、デフォルトではこの動作が禁止されました。詳細は第3章で説明します。

*12 事前にコンパイルを行なっておかないと実行できないタイプのプログラミング言語。コンパイルとは、ソースプログラムから各CPU依存の機械語への変換（翻訳）を行ない、機械語の実行ファイルを生成する作業のことです。PHPのようにコンパイルが不要な言語はスクリプト言語と呼ばれます。

数名と次の文字列をくっつけて書いてしまうと、$が変数の始まりですから、PHPのパーサ(構文解析部)は **$meiji年** という名前の変数を探しに行こうとします。しかしそのような変数は定義されていないので評価結果は空文字列になり、結局何も表示されません。ちなみに本書推奨の設定では、未定義変数の参照という意味の英語の警告が表示されます。対応策としては「明治$meiji 年」のように変数のうしろに1個以上の半角空白を置いて変数名の終端を明示するか、この例のように **{}** でくくるか、または後述するprintf()関数のフォーマット指定などを使います。

文字列の中で配列を参照する場合はさらに複雑になります。PHPの配列では **[]** により添字を指定します。細かい説明は第3章に譲りますが、文字列の中では単純に **$変数名['添字']** と書くとパースエラー(文法エラー)となってしまうので、変数名全体を **{}** でくくって **{$変数名['添字']}** と書くようにしてください。**$変数名[添字]**(シングルクォートを忘れている)と書いても現状は動作しているように見えますが、これは将来動作が変更されるかもしれないのでお勧めできません。

ではtest4.htmlを動かして、筆者の生まれた年である **1959** を入力し、送信ボタンを押してみましょう。ブラウザのURL入力エリアのところで **?yyyy=1959** が自動的に付加されているのを確認してください(w3mではcキーを押すと現在のURLを表示します)。画面には以下のように表示されます。

```
西暦1959年は、明治93年、大正48年、
昭和34年、平成-29年にあたります。
```

通常、元号の変わり目は1年の途中にあるので、本来ならば年月日まで入力させてきちんと判定しないといけないのですが、この対応はあとの宿題に取っておくことにします。

2.11 URLで直接引数を指定する

test4.phpはtest4.htmlから呼び出されることを前提に記述されていますが、FORMタグでGETメソッドを使用する場合は、直接引数を指定して呼び出すこともできます。コマンドラインから

```
w3m http://localhost/~hotta/test4.php?yyyy=1959
```

と呼び出してみてください。上記とまったく同じ出力が行なわれることがわかります。

このURLに埋め込む引数は「クエリー文字列」と呼ばれています。クエリー文字列は、受け取り側のスクリプトのURLに続いて **?引数1=値&引数2=値** という形式で渡すことができます。

2.12 スクリプトをひとつにまとめる

test4.htmlとtest4.phpのように、入力と出力を各々別のファイルにするのは一見分かりやすくてよさそうですが、システムが複雑になってくるとファイルの数が増えて収拾がつかなくなることもあります。なるべく一連の処理はひとつのファイルにまとめたいものです。かといって、単純にスクリプトをくっつけただけではうまくいきません。通常は入力と出力によって処理を分ける必要があります。ここでは $GET['yyyy'] に値が入っているかどうかによって処理を分けることにしました。また、最初や最後の決まり文句（<HTML>タグなど）の出力など、重複する処理はひとつにまとめるようにしてみました（test41.php）。

リスト1-8　test41.php

```
<HTML>
<BODY>
<?php if (empty($_GET['yyyy'])): ?>
<FORM action="test41.php">
西暦<INPUT TYPE=text NAME=yyyy>年
<INPUT TYPE=submit>
</FORM>
<?php else:
$meiji = $_GET['yyyy'] - 1866;
$taisho = $_GET['yyyy'] - 1911;
$showa = $_GET['yyyy'] - 1925;
$heisei = $_GET['yyyy'] - 1988;
print "西暦{$_GET['yyyy']}年は明治{$meiji}年、大正{$taisho}年、<BR>¥n"
. "昭和{$showa}年、平成{$heisei}年となります。<BR>¥n";
endif;
?>
</BODY>
</HTML>
```

何らかの条件にしたがった処理の分岐にはIF文を使います。IF文の括弧の中には何らかの「式」が入ります。式（評価式）とは、その部分を評価することによって何らかの値が得られるようなプログラムの一部分のことをいいます（詳しくは「3.3　ステートメント」で解説します）。

IFに与えられた条件が真、すなわち $_GET['yyyy'] がセットされていない場合は<FORM>～</FORM>までが実行され、そうでなければ（$_GET['yyyy'] がセットされていれば）ELSEに制御が移り、ENDIFまでの文が実行されます。IFの条件にかかわらず、最初の<HTML><BODY>と最後の</BODY></HTML>は無条件に出力されます。

ここでは、$_GET['yyyy'] がセットされているかどうかの判定のためにempty()という組

み込み関数を使っています。関数とは、0個以上の引数をとり（つまり引数はなくてもよい）、何らかの値を返すように作られている一連の処理に対して、ほかから呼び出すための名前をつけたものです。関数は変数とは別の名前空間を持っているので、変数$abcと関数abc()にはまったく関連性はありません。

PHPには便利な組み込み関数が多数用意されています（詳細は第3部の関数リファレンスを参照）。empty()もそのうちのひとつで、「与えられた変数がセットされていれば真を返す」という働きを持っています。今まで使ってきたprintも（本当は関数ではありませんが）関数のように使うことができ、print(引数) という書式でも使えます。このあたりがPHPの柔軟で、かついいかげんともいえる点です。

IF文も IF(...) という形式を持っているので一見関数のようですが、これは関数ではなく、実行の流れを制御するための文（ステートメント）です。IF(条件) のうしろはセミコロン（;）ではなくてコロン（:）です。IF() の行だけでは文として成り立たないので、IF()の行には ; はつきません。IF() 文の終りは ENDIF; になります。このように、関数ではないPHPの要素（予約語）には、IF文をはじめとする各種制御構造などがありますが、数はあまり多くありません（詳細は「3.22 制御構造」で解説します）。

test41.phpでは、IF() の行末でPHPモードを解除してHTMLモードに戻していますが、IF() の条件を満足しない場合は、HTMLモードの部分（<FORM>～</FORM>）も評価（実行、出力）されません。つまりHTMLモードの場合も、PHPの制御構造の管理下にあるということができます。

2.13 処理ロジックを分割する

test41.phpでは、いったん結果を得たあとでもう一度異なった値を入力したいと思っても、現状はブラウザの「戻る」ボタン（w3mではB）で戻るしかありません。そこで、結果を表示しつつ、同時に入力も行えるようにしたいと思います。また、このままではでたらめな値を入力されてもそのまま計算を行なってしまうので、入力のエラーチェック機能も追加したいところです。このように、機能やユーザの使い勝手を追求すればするほど、一般的にプログラムというものは肥大化の一途をたどっていきます。

チェック条件がどんどん複雑になり、またそれらに応じた処理も長くなってくると、スクリプトがだんだん読みにくくなってきて、文法エラーになったり、思ったとおり動いてくれなくなったりします。プログラム中に含まれる、自分の意図とは異なった振る舞いを示すエラーのことをバグ（bug、虫）と呼び、このバグをつぶす（プログラムのエラーを取り除く）作業のことをデバッグ（debug）といいます。プログラミングとは、すなわちデバッグ作業なのです。しかし、ただずるずるといたずらに長くなってしまったプログラムコードをデバッグするのは非効率的です。シンプルで読みやすく美しいコードはバグの混入の可能性（やバグの数）を少なくし、開発の生産性を高めます。

さてtest41.phpの改訂案ですが、以下のように考えてみました。

```
入力エリアを（入力値があればそれも合わせて）表示する;
if （入力値があり、それが妥当であれば）:
    元号に直す計算をする;
    計算結果を表示する;
endif;
```

このように、人間が読める形式で処理の流れを記述することを、疑似コードによるプログラミングと呼びます。複雑なプログラムを書こうとする場合、いきなりコーディングを始めないで、まずこういったことを行なってみるのもプログラムの見通しをすっきりさせるために役立ちます。

では、これをPHPの文法に直してみましょう。

リスト1-9 test5.php

```
<?php
display_input_area();                              // 入力エリアの表示
if (!empty($_GET['yyyy']) && input_is_valid()):
    $result = calc_gengou();                       // 元号の計算
    display_result($result);                       // 結果の表示
endif;
?>
```

どうでしょうか？ だいぶすっきりしてきたと思いませんか？

PHPでは //から行末まではコメントとみなされ、実行には影響を与えません。このように日本語でコメントを入れておけば、多少なりともプログラムが見やすくなります。

display_input_area()という、いかにもそのまんまといった名前の関数を使っていますが、これはPHPで用意されたものではなく、私達が自由に名前をつけて使うことのできる「ユーザ定義関数」と呼ばれるものです。「入力エリアの表示」といった一連のまとまった処理をひとつの関数としてまとめるようにすると全体の流れを追いやすくなるので、ぜひこのように書くことをお勧めします。また、わかりきったコメントを書くより、ストレートに処理の内容を表す関数名を考えるほうが、プログラムの可読性を上げるために有用です。

2行目に出てくるinput_is_valid()も「入力が妥当であれば真を返す」関数として実装しようと考えています。もっとも初回の表示時など、何も入力されていない状態で入力チェックをするのは無駄ですから、$_GET['yyyy'] がセットされている場合のみ入力チェックを行なうようにしました。&& は「かつ」を表す論理演算子です。式は && を使っていくつも連結して評価することができます。「または」を表すのは || です。

2.14 疑わしきは検証せよ

ここで if(式A && 式B) と書いたとき、式Aを評価してそれが偽であれば、式Bを評価するまでもなく全体が偽となるのは、人間が見れば自明です。コンピュータに判断させる場合にも、本当にそう動いてくれれば無駄な処理をする必要がなくなり、実行速度も上がってよいことばかりなのですが、PHPはそう動いてくれるでしょうか？

プログラミングのコツは「信じない、思いこまない」ことです。プログラミングの途中で疑問がわいてきたら逐一検証してみるのが、最初は寄り道になっても結局は上達への早道です。せっかくなので、ユーザ定義関数を使って、これを検証するためのコードを書いてみましょう（**リスト1-10**）。説明の都合上、行番号をつけています。

リスト1-10 test6.php

```
 1 <?php
 2 function func1()
 3 {
 4     print "func1()¥n";
 5     return FALSE;
 6 }
 7 function func2()
 8 {
 9     print "func2()¥n";
10     return TRUE;
11 }
12 if (func1() && func2()):
13     print "TRUE¥n";
14 else:
15     print "FALSE¥n";
16 endif;
17 ?>
```

新たなキーワードfunctionとreturnが出てきました。functionはその名のとおり、ユーザ定義関数を作ります。関数名は、PHPで定義されている組み込み関数名および予約語と衝突しなければ何でもかまいません。関数の命名規則は「3.17　関数」で解説します。

ユーザ定義関数は0個以上の引数を受け取ることができますが、上記の例では引数なしで定義しています。また、関数はreturn命令によりひとつの値を戻り値として呼び出し元に返すことができます。明示的にreturnで戻らない関数の場合、戻り値はFALSEとなります。TRUEおよびFALSEはPHPの予約語で、それぞれ真と偽を表します。"TRUE" および "FALSE" と二重引用符で囲ってしまうと単なる文字列となってしまうので、コーディングの際は気をつけましょう。

図 1-11 test6.phpの実行結果

```
hotta@star ~/public_html/1-2$ php -q test6.php
func1()
FALSE
```

ユーザ定義関数は、定義されるだけでは実行に何の影響も与えません。test6.phpで一番始めに実行されるのは、12行目のIFの行です。IFの()内の評価式が複数ある場合は、左側から順に評価されます。最初の評価式func1()は関数なので、ユーザ定義関数への呼び出しが発生し、func1()に制御が移ります。

func1()では単に自関数名の表示を行なったあと、呼び出し元に対してFALSE（偽）を返しています。ここでifに制御が戻りますが、&&で結合された評価式のうちひとつが偽と判定されたので、そのあと後続をいくら評価しようが全体としては偽になります。そこでPHPはIF文の評価を打ちきり、ifの条件が成り立たなかった場合の処理である偽の表示を行ない、プログラムを終了します。この動きは、func2()に入れば、かならず行なわれるはずのfunc2()の表示が行なわれていないことから明らかです。

ここで疑い深い筆者は、ユーザ定義関数で定義した名前がすでにPHPの組み込み関数として用意されている関数名とだぶってしまったらいったいどうなるんだろうという疑問が起こりました。疑問が起こったら早速検証です。

リスト 1-11 test7.php

```
<?php
function strlen($text)
{
    print $text;
}
$foo = "ABC";
print "strlen(¥$foo)=" . strlen($foo);
?>
```

strlen(文字列) は、指定された文字列のバイト数を返すための組み込み変数です。たとえばstrlen("ABC")は3を返します。test7.phpではこれをユーザ定義変数として使っています。

ここで、結果を表示するprint文で**¥$foo**という表記を行なっています[*13]。実は、ここは単に **strlen($foo)**という文字列定数を表示したいのですが、このとおり書いてしまうとprint文の実行時に**$foo**が変数名として先に評価されてしまい、その中身が展開されて表示されてしまうからです。ここではそれを避けるために**$**の前にバックスラッシュ（＼。日本語キーボードでは**¥**：エンマークで刻印されている）をつけて、本来変数を意味する$をエスケープして（打ち消して）います。このように、特殊文字の直前に置かれ、後続文字の意味を打ち消すための記述を「エスケープシーケンス」と呼びます。同様に、二重引用符（"）自体

*13
fooは、ソフトウェアの世界で「何とか」という総称的な意味を表すための俗語です。これ以外にbarやbazなどがあります。日本語で書く場合にはfuga、hoge（ふが、ほげ）などと書くこともあります。

を表したい場合も¥''とする必要があります。要するに、PHPの文法上特殊文字として使われている文字を通常の文字として使用したい場合は、¥でエスケープしなければなりません。

strlen()の引数として$fooを渡しているので、ユーザ定義関数が呼ばれればfoo、本来のPHPのstrlen()が呼ばれれば3と表示されるはずです。では試してみましょう。

図1-12 test7.phpの実行結果

```
Fatal error: Cannot redeclare strlen() in test7.php on line 2
```

エラーメッセージが表示されてしまいました。これを訳すと、「致命的エラー：test7.phpの2行目で、strlen()を再定義することができません」となります。このように、PHPでは内部関数とユーザ定義関数の名前の重複など文法的に許されない表記を行なうと、パーサが行番号とともにエラー内容を通知してくれるので、安心してコーディングを行なうことができます。

2.15 ソースを分割する

さて本題に戻って、元号計算プログラムの中身を作っていきましょう。test5.phpを再掲します。

リスト1-12 test5.php

```
<?php
display_input_area();                          // 入力エリアの表示
if (!empty($_GET['yyyy']) && input_is_valid()):
    $result = calc_gengou();                   // 元号の計算
    display_result($result);                   // 結果の表示
endif;
?>
```

次にユーザ定義関数の中身を書いていくわけですが、最終的にはプログラムが若干長くなりそうです。ここで、ソースファイル（スクリプトを格納するファイルそのもの）を分割することを考えます。display_input_area()を**リスト1-13**のようにしてみました。HTMLの表示にはヒアドキュメントを使っています。

リスト1-13 display.inc

```
<?php
//
// 入力画面の表示
//
function display_input_area()
{
    print   <<<__EOD__
<FORM action="{$_SERVER['PHP_SELF']}">
西暦<INPUT TYPE=text NAME=yyyy SIZE=5 VALUE="{$_GET['yyyy']}">年
<INPUT TYPE=submit VALUE="送信">
</FORM>
__EOD__;
}
?>
```

PHPに限らず、一定以上の規模のプログラムを1本のソースプログラムで記述しようとすると読みにくくなりがちです。またプログラムが複雑になってくると、あちこちに重複した処理や共有したい処理、定数などが現れます。共有したいもの（定数、ユーザ定義関数の定義など）をあちこちのソースプログラムにコピーしてまわると、ちょっとした変更が発生したりしてもすべてのソースを開いて変更を加えないといけなくなり、収拾がつかなくなります。

このような場合、共通処理をひとつまたは複数のソースファイルに分けて格納し、その機能を呼び出したい各プログラムが、共通処理が書かれているソースファイルを「インクルードする（取り込む）」ことで、あたかもその部分に共通処理が書かれているように扱うことができます。重複した処理は、ユーザ定義関数としてひとつにまとめることにしましょう。多少の違いは引数で判断して振る舞いを変えることにより、関数内部で違いを吸収してやるようにします。

上記の例ではdisplay_input_area()を別ファイルに追い出してみました。その中身ですが、<INPUT>タグのVALUE属性で、入力された$_GET['$yyyy']の値をデフォルト値として入力エリアの中に表示しています。

ACTIONタグの属性を{$_SERVER['PHP_SELF']}としていますが、この配列変数は「自分自身のスクリプト名」を保持している予約変数です。さらに、このようにインクルードされたスクリプトから参照しても、この値はURLで指定された、インクルードしている側のメインスクリプト名が入ります。test5.phpから呼び出されるのであれば、この変数の値はtest5.phpとなります。

test5.phpのほうにはプログラムの先頭に

```
require_once("display.inc"); // または include_once("...")
```

という1行を追加してやります。

するとパーサは、構文解析に先立って（include_once()の場合は実行時に）require_once()文の行をdisplay.incの中身で置き換えます。ここではインクルードされるほうのファイルの拡張子を.incとすることにしますが、実際には何でもかまいません。しかし .htmlなど、もともとApacheで関連づけが行われているものは避けたほうがよいでしょう。インクルードされるファイルは、これらが単独で呼び出されることは想定していない場合が多いでしょうから、セキュリティのために、可能であればインクルードファイルが単独で呼び出されることのないように、Apacheの設定を変更しておいたほうがよいでしょう[*14]。

*14
.inc の拡張子を持つファイルへのアクセスを禁止するには、httpd.conf の中で以下のように設定します。

```
<Files ~ "^¥.inc">
Order allow,deny
Deny from all
</Files>
```

詳細は Apache のマニュアルを参照してください。

2.16 入力値のチェック

ユーザからの入力を受けつけて処理を行なうにあたり、入力チェックは必須です。人間の振る舞いは、コンピュータシステムの中で一番信頼がおけないもののひとつです。ではinput_is_valid()を実装してみましょう。これも別ソースにしてみました。

リスト1-14 input.inc

```
<?php
//
//  1000～2999のあいだであれば真を返す
//
function input_is_valid()
{
    if (ereg("^[12][0-9]{3}$", $_GET['yyyy'])):
        return TRUE;
    endif;
    return FALSE;
}
?>
```

この関数では、$_GET['yyyy']が1000～2999の範囲にあれば真、そうでなければ偽を返します。ここでは新しい組み込み関数ereg()を使っています。これは「第2引数として与えられた文字列が第1引数として与えられた正規表現（コラム参照）にマッチしていれば真を返す」働きを持っています。ここでは**^[12][0-9]{3}$**という正規表現を使っています。これは、「1または2という文字（[12]）で始まり（^）、その直後に0から9までのいずれかの文字が3個続き（[0-9]{3}）、それで文字列が終わる（$）もの」を示します。これを一般的なことばで書けば、1000～2999という西暦の範囲を表すことになります。

Column

正規表現

正規表現とはテキスト処理においてよく使われるもので、文字列をパターン化して総称的にうまく表現できるように考えられた記号の並びのことです。PHPではPOSIX1003.2で定義されたPOSIX拡張正規表現を使用します。よく使うものとして、下記のようなパターンがあります。

● 正規表現のパターン例

`^`	文字列の始まり
`$`	文字列の終り
`?`	直前の文字の`0`個または`1`個の繰り返し
`*`	直前の文字の`0`個以上の繰り返し
`+`	直前の文字の`1`個以上の繰り返し
`{n}`	直前の文字の`n`回の繰り返し
`{m,n}`	直前の文字の`m`～`n`回の繰り返し
`[abc]`	`a`、`b`、`c`いずれかの1文字
`[a-z]`	小文字1文字
`[0-9A-Za-z]`	英数字
`[^0-9]`	数字以外
`|`	`or`指定
`(...)`	正規表現のグループ化

「man 7 regex」manページで、正規表現に関する詳しい解説を読むことができます。正規表現はそれだけで1冊の本が書けるくらい奥深いものですが、慣れると文字列のパターンマッチングに関する記述の効率が飛躍的に高まりますから、基礎的な部分だけでもぜひマスターしておきたいものです。Perl の正規表現に慣れている人は、Perl互換の正規表現(preg_XXX()関数)を使うこともできます。さらに日本語文字列を正規表現で処理する場合は、マルチバイト対応のmb_ereg_XXX()を使ってください。

2.17 複数の値を使う

次にcalc_gengou()を実装してみましょう。

当初の例では、各々の和歴の値を保持するのに$meiji、$taishoといった個別の変数を定義していました。ここではもう少しスマートに、配列を使ってみましょう。calc_gengou()では、入力値に対して四つの計算を行ない、ひとつのreturn文で四つの値を返します。

リスト1-15 calc.inc

```
<?php
//
// 元号の計算
//
function calc_gengou()
{
    $result = array();
    $result['明治'] = $_GET['yyyy'] - 1866;
    $result['大正'] = $_GET['yyyy'] - 1911;
    $result['昭和'] = $_GET['yyyy'] - 1925;
    $result['平成'] = $_GET['yyyy'] - 1988;
    return $result;
}
?>
```

2.18 書式つき出力

最後にdisplay_result()です。

リスト1-16 result.inc

```
<?php
//
// 結果の表示
//
function display_result($wareki)
{
    printf("西暦%d年は明治%d年、大正%d年、<BR>¥n"
         .  "昭和%d年、平成%d年となります。<BR>¥n",
        $_GET['yyyy'], $wareki['明治'], $wareki['大正'],
        $wareki['昭和'], $wareki['平成']);
}
?>
```

display_result()関数の中ではprintf()関数が使われています。これはprintと同じように変数や定数の値を出力するためのものですが、出力のフォーマット指定を柔軟に行なえます。第1引数はフォーマット文字列で、この中に埋め込まれている%dという表現が、二番目以降の引数の値に順番に置き換わります。ここで出力桁数を指定することもできるので、出力の体裁を整えるのによく使われます。詳細は第3部の関数リファレンスを参照してください。

さらに、display_result()では引数として和暦の配列を受けていますが、呼び出し元では

$result という変数名で渡しています。関数コール時に渡す引数を実引数、受け取り側で用意する変数を仮引数といい、これらは名前が異なっていてもかまいません。

では、test5.php に対し、これらのインクルードファイルを取り込むような変更を加えた test8.php を作成し、早速実行してみましょう。未定義変数の警告が出ないように、若干手を加えています。

リスト1-17 test8.php

```
<?php
//
//   西暦を和暦に変換する
//
include_once("display.inc");
include_once("input.inc");
include_once("calc.inc");
include_once("result.inc");
if (empty($_GET['yyyy'])):
    $_GET['yyyy'] = "";
endif;
display_input_area();                                    // 入力エリアの表示
if (!empty($_GET['yyyy']) && input_is_valid()):          // 妥当性チェック
    $result = calc_gengou();                             // 元号の計算
    display_result($result);                             // 結果の表示
endif;
?>
```

いかがでしたか？ ここまでで、とりあえずPHPのスクリプトプログラミングの基礎レベルについては理解してもらえたのではないでしょうか。まだまだこのままでは実用的なサンプルとはいいがたいので、第4章では年月日まで考慮して元号の判定を行なうように、機能追加したスクリプトを作成します。

Hypertext Preprocessor

Chapter-3 PHPの文法を知ろう

前章では主に初心者を対象として、PHPプログラミングの初歩について解説しました。その際説明を簡単にするために、細かい文法説明は省略しました。この章では、PHPの文法について体系的に整理してみます。前提として、読者にはHTMLについての知識があることを想定しています。説明の都合上新しい組み込み関数が現れることがありますが、各関数の仕様については第3部の関数リファレンスを参照してください。また本文で触れられていない点について、付属CD-ROMに収録してあるPHPマニュアルに詳しく説明されていることもあるので適宜参照してください。

3.1 HTML埋め込み型言語

PHPスクリプトは、HTML文書の中に埋め込むことができます。

リスト1-18 ex01.php

```
<HTML><BODY>
<?php echo("ここはPHPコードの部分です。¥n"); ?>
</BODY></HTML>
```

また、ファイル全体をPHPスクリプトにし、必要に応じてHTMLの出力を行なうこともできます。

リスト1-19 ex02.php

```
<?php
echo "<HTML><BODY>¥n";
echo("このスクリプト全体がPHPコードです。¥n");
echo "</BODY></HTML>¥n";
?>
```

後者の変形として、ヒアドキュメントが使えます。

リスト1-20 ex03.php

```
<?php
echo <<<__EOD__
<HTML><BODY>
ヒアドキュメントも使えます。
</BODY></HTML>
__EOD__;
?>
```

どれでも読みやすいと思うやりかたでコーディングすればよいでしょう。

3.2 開始と終了

HTML中にスクリプトを埋め込む方法には、以下の4種類があります。これらの4種類を混在させることも可能です。

● <?php ～ ?> で囲む

リスト1-21 ex04.php

```
<?php echo("ここはPHPコードの部分です。¥n"); ?>
```

一番普通の書き方です。第1部でもこの書き方を使っています。

文の要素間にあるホワイトスペース（空白、タブおよび改行）は無視されるので、この例は

リスト1-22 ex05.php

```
<?php
    echo    ("ここはPHPコードの部分です。¥n");
?>
```

とも書けます。echoと(の間に空白があっても大丈夫です。いろいろためしてみてください。思いがけない発見をすることがあるかもしれません。

● <script language="php"> ～ </script> で囲む

リスト1-23 ex06.php

```
<script language="php">
    echo("ここはPHPコードの部分です。");
</script>
```

● <% ～ %> で囲む

これはMicrosoftのASP（Active Server Pages）と同様の方式ですが、これを有効にするにはphp.iniファイルのasp_tagsディレクティブを有効にする必要があります。デフォルトでは無効になっています。

リスト1-24 ex07.php

```
<?% echo("ここはPHPコードの部分です。¥n"); %>
```

● <? ～ ?> で囲む

リスト1-25 ex08.php

```
<? echo("ここはPHPコードの部分です。¥n"); ?>
```

これは従来の形式ですが、現在ではphp.iniファイルのshort_open_tagディレクティブをOn にしないと使えません。デフォルトでは無効になっており、推奨されません。

3.3 ステートメント（文）

ステートメントは処理の単位です。ステートメントはひとつ以上の「式」から構成され、セミコロン（;）で終わります。各ステートメントは大文字でも小文字でも同様に評価されます。

リスト1-26 ex09.php

```
<?php
ECHO("これはOK");
Echo("これもOK");
echo("これでもOK");
?>
```

例外として、スクリプトの閉じタグ ?> には暗黙に ; の意味を含みます。つまり ?> の直前のステートメントの ; は省略が可能です。

3.4 コメント(注釈)

ステートメントの外側に、C言語またはJava形式のコメントを書くことができます。

リスト1-27 ex10.php

```
<?php
echo("処理1");   /* C言語形式のコメント */
echo("処理2");   // Java形式のコメント
?>
```

3.5 組み込み関数

前節までの説明の中でecho(...)という書式を使っていますが、この**名前(引数)**形式による各種機能の呼び出し方法を関数といいます。PHPでは多数の組み込み関数が用意されており、プログラマはこれらを組み合わせて複雑な処理を行なうことができるようになっています。関数によってはコンパイル時に特殊なオプションをつけないと、デフォルトでは使えないものもあります。第1部で使われている組み込み関数は、どれも標準的に使用できるものです。

3.6 式と型

PHPの文法においてもっとも重要な要素は「式」です。式とはスクリプトの一部分であり、それを評価することによって何らかの値が得られる(これを「値を持つ」と表現します)ものといえます。また、すべての式は「型」を持ちます。型(タイプ)とは、その文脈(プログラムの流れ)において現れた式をどのように扱うかという決まりごとです。

PHPでは以下の型をサポートします。

- ブール(boolean)
- 整数(integer)
- 倍精度実数(double)
- 文字列(string)
- 配列(array)
- オブジェクト(object)
- リソース(resource)

3.7 変数

変数とは、値を保持する記憶領域に名前（シンボル名）をつけたものです。変数の場合はシンボル名の前にドル記号（$）をつけて、関数名などのほかのシンボルと区別します。変数名はステートメントと違って大文字・小文字を区別するので、$abcと$ABCはまったくの別物となります。

$a=5は$aという変数に5という整数値を代入することを表しますが、同時に5という値を持つ式でもあります。また$b=($a=5)といった書き方もできます。これは $b=$a=5 と同値です。

変数にはスカラーと非スカラーという二つのタイプがあります。スカラーとは単独の値を取る変数（単純変数）で、非スカラーとは複数の値を取る変数（配列またはオブジェクト）です。

システムにより予約された定義済み変数もあります。詳細は「5.4　CGI」を参照してください。

3.8 シンボルの命名規則

シンボル名は、PHPスクリプトにおける変数や関数などの識別子です。有効なシンボル名は文字またはアンダースコアから始まり、任意の数の文字、数字、アンダースコアが続きます。正規表現によれば、これは

```
'[a-zA-Z_¥x7f-¥xff][a-zA-Z0-9_¥x7f-¥xff]*'
```

のように表現することができます。ここでいう文字とは、a-z、A-Z、127から255まで（0x7f-0xff）のASCII文字を意味します。変数名に日本語を使うことは推奨はされませんが、LinuxにおいてEUCコードでスクリプトが書かれていれば一応使えます。

3.9 型の相互変換

PHPでは変数の定義をする際、明示的な型宣言を必要としません。変数の型は、その変数が使用される際の文脈により自動的に決定されます。たとえば、変数$varに文字列を代入した場合には$varは文字列変数に、整数値を代入すれば整数になります。

PHPの自動型変換の例として、加算演算子＋が挙げられます。式の中で＋を使用すると、オペランド（演算子の作用対象となるもの）のいずれかに倍精度実数として評価できるものがあれば、すべてのオペランドは倍精度実数として評価され、結果も倍精度実数になります。オペランドのいずれも倍精度実数でない場合、オペランドは整数として評価され、結果も整数になります。この自動型変換は、オペランドの型自体を変更するものではないということ

に注意してください。変わるのは、オペランドがどのように評価されるのかということに過ぎません。

以下の例では、変数に順に値を代入するとともに、gettype()関数で変数の型がどう変化していくのかを観察しています。何を足すかによって、ダイナミックに型が変わっていくのがわかります。

リスト1-28 ex11.php

```
<?php
$foo = "0";                          // $foo は文字列"0" です
echo "¥$foo=$foo " . gettype($foo) . "¥n";
$foo++;                              // $foo は整数 1 です
echo "¥$foo=$foo " . gettype($foo) . "¥n";
$foo += 1;                           // $foo は整数 2 です
echo "¥$foo=$foo " . gettype($foo) . "¥n";
$foo = $foo + 1.3;                   // $foo は倍精度実数 3.3 です
echo "¥$foo=$foo " . gettype($foo) . "¥n";
$foo = 5 + "10 Little Piggies";      // $foo は整数 15 です
echo "¥$foo=$foo " . gettype($foo) . "¥n";
?>
```

図1-13 ex11.phpの実行結果

```
hotta@star ~/public_html/1-3$ php -q ex11.php
$foo=0 string
$foo=1 integer
$foo=2 integer
$foo=3.3 double
$foo=15 integer
```

3.10 型キャスト

変数を一時的に、強制的に別の型として評価したい場合があります。強制的に行なう型変換のことをキャストと呼びます。PHPの型キャストはC言語のそれとよく似ています。つまり、変換しようとする型の名前を括弧に入れて、キャストする変数の前に挿入します。

リスト1-29 ex12.php

```
<?php
$foo = 10;               // $foo は整数です
$bar = (double) $foo;    // $bar は倍精度実数です
?>
```

キャストの書式を以下に示します。

(int)、**(integer)** …… 整数へのキャスト
(real)、**(double)**、**(float)** …… 倍精度実数へのキャスト
(string) …… 文字列へのキャスト
(array) …… 配列へのキャスト
(object) …… オブジェクトへのキャスト

3.11 文字列から数値への変換

ある文字列が数値として評価されるとき、結果の値と型は次のように定義されます。

・PHPは、与えられた文字列の最初の部分が有効な数値データから始まる場合には、一連の文字列を数値として評価しようとします。有効な数値データとは、符号（オプション）のあとに（オプションとして小数点を含む）ひとつ以上の数値があり、そのあとに指数表現（オプション）があるものです。指数表現はeまたはEのあとにひとつ以上の数字があるものです。有効な数値で始まらない文字列は、0と評価されます。
・0でない数値として評価された場合、文字列の有効部分に小数点（.）、e、Eのいずれかを含む場合には倍精度実数として評価されます。そのほかの場合は整数として評価されます。

リスト1-30 ex13.php

```
<?php
$foo = 1 + "10.5";           // $foo は倍精度実数 (11.5)
$foo = 1 + "-1.3e3";         // $foo は倍精度実数 (-1299)
$foo = 1 + "bob-1.3e3";      // $foo は整数 (1)
$foo = 1 + "bob3";           // $foo は整数 (1)
$foo = 1 + "10 Small Pigs"; // $foo は整数 (11)
?>
```

この変換規則に関する詳細は、UNIXのstrtod（3）manページを参照してください。

3.12 配列

配列とは、ひとつの変数名で複数の値を保持するためのしくみです。たとえば$aという配列がある場合、配列内部の各々の要素は、$a[0]、$a[1]などのように、格納されている順番を

表す「添字（そえじ）」をつけて表されます。PHPの場合、さらに「連想配列」というメカニズムが用意されており、添字として任意の文字列が使えるようになっています。実際、数値で表される添字も文字列で要素を特定するための「キー」もPHPの内部的には同等なので、ここでの説明ではキーで統一します。

配列の初期化には、値を連続的に代入するか、またはarray()関数のいずれかを用います。キーを指定せずに値を連続的に代入すると、PHPは自動的に数値のキーを生成し、配列に対して連続的に配列要素が追加されます。その要素は配列の最後の要素として追加されます。

リスト1-31 ex14.php

```
<?php
$fruits[] = "りんご";      // $fruits[0] = "りんご"
$fruits[] = "みかん";      // $fruits[1] = "みかん"
$fruits[3] = "なし";       // $fruits[3] = "なし"
                           // $fruits[2]は未定義のまま
?>
```

配列要素のキーは、1からではなく0から始まるので注意してください。

array()は、変数のリストを配列にして返す関数です。上記の例をarray()関数を使って書くと以下のようになります。

リスト1-32 ex15.php

```
<?php
$fruits = array("りんご", "みかん", "", "なし");
?>
```

連想配列による配列の初期化にもarray()関数を使用したほうが便利です。たとえば、果物の種類とそれぞれの単価を以下のように表すことができます。

リスト1-33 ex16.php

```
<?php
$prices = array(
   "りんご" => 120,       //  $prices["りんご"] = 120
   "みかん" =>  80,       //  $prices["みかん"] =  80
   "なし"   => 170);      //  $prices["なし"]   = 170
?>
```

このように、連想配列とはすなわちキーと値のペアです。上記の例では、果物の名前というキーがわかればそれに対応する値（値段）が得られます。ここで、配列の内部では要素の格納順序は不定であることに注意してください。登録順でさえありません。このため、配列を何らかの意味のある順番に参照したい場合は、明示的にソート（整列、並べ替え）を行なう必要があります。配列をソートするには、値の型に応じた各種のソート用の組み込み関数が用意されています。またcount()関数を用いて配列の要素数をカウントしたり、next()とprev()関数で配列の中を移動することができます。さらに、配列の中での一般的な移動法としてeach()関数があります。詳細については第3部の関数リファレンスを参照してください。

文字列の中で配列参照を行なう場合は、配列変数全体を大括弧 {} で囲みます。

リスト1-34 ex17.php

```
<?php
$a['bar'] = "Bob";
echo "¥$a['bar']={$a['bar']}";  //  $a['bar']=Bob
?>
```

3.13 多次元配列

多次元配列の指定は簡単です。配列のキーのあとに、さらにキーをつけるだけです。

リスト1-35 ex18.php

```
<?php
$a = 1;                          //  単純変数
$b[0] = 2;                       //  一次元配列
$c[0][1] = 3;                    //  二次元配列
$c[0]['foo'] = 4;
$d[0]['foo']['bar'] = 4;         //  三次元配列
?>
```

多次元配列を初期化する際、array()の引数としてarray()をネスト（入れ子に）して使用することができます。

リスト1-36 ex19.php

```
<?
    $a = array(
            "リンゴ"  => array(
                "色"  => "赤",
                "味"  => "甘い",
                "形"  => "丸い"
          ),
           "オレンジ"  => array(
                "色"  => "オレンジ色",
                "味"  => "すっぱい",
                "形"  => "丸い"
          ),
             "バナナ"  => array(
                "色"  => "黄色",
                "味"  => "ペースト様",
                "形"  => "バナナの形"
          )
    );

    echo $a[リンゴ][味];     # "甘い" を出力します
?>
```

3.14 可変変数

変数名を可変にできると便利なことがあります。可変変数を使うと、変数名を動的にセットして使用することができます。通常の変数は

```
$a = "hello";
```

のような代入文により値をセットします。可変変数は、ドル記号を二つ使用することにより、変数の値を別の変数の名前として扱います。

```
$$a = "world";
```

この時点で二つの変数 $a と $hello が定義されています。これらの値はそれぞれhelloとworldです。その状態において

```
echo "$a ${$a}";
```

の出力は

```
echo "$a $hello";
```

とまったく同じになります。すなわち、両方ともに「hello world」を出力します。

可変変数を配列で使用するときには、曖昧さの問題を解決する必要があります。つまり$$a[1]と書いた場合、$a[1]を変数名として使用したいのか、それとも$$aを変数名とし、[1] をその変数のキーとしたいのかを、パーサ（構文解析ルーチン）に教えてやる必要があるのです。この曖昧さを解決するには大括弧{}を使って、前者では${$a[1]}、後者では${$a}[1]という構文を使います。

3.15 参照による代入

PHPでは、変数に対して参照による代入が行なえます。この場合、新しい変数は元の変数を参照するだけです。言い換えると、元の変数のエイリアス（別名）を作る、または元の変数を指すことになります。参照による代入の場合は値のコピーが行なわれないため、値による代入よりは高速になります。

参照により代入を行なうには、代入する変数（ソース変数）の頭にアンパサンド（&）をつけます。たとえば、次の簡単なコードは「My name is Bob」を二度出力します。

リスト1-37 ex20.php

```
<?php
$foo = 'Bob';               // 値'Bob'を$fooに代入する
$bar = &$foo;               // $fooを$barにより参照
$bar = "My name is $bar";   // $barを変更...
echo $foo;                  // $fooも変更される
echo $bar;
?>
```

注意すべき重要な点として、名前を持つ変数のみが参照により代入できるということがあります。

リスト1-38 ex21.php

```
<?php
$foo = 25;
$bar = &$foo;       // これは有効な代入です。
$bar = &24;         // 文法エラーです。名前のない式を参照しています。
?>
```

3.16 定数

定数は変数と異なり、文字どおり変化することのない値であり、即値とも呼ばれます。定数には数値定数と文字列定数があり、それらの扱いはPerlとほぼ同じです。数値定数は1や

-1.1、**1.0e32** といった算術表記で表されるものです。文字列定数は、**"ABC"** や **"定数"** といった、数字以外の文字から始まる文字の並びです。C言語などとは異なり、文字列が引用符でくくられていなくても正常に動作しますが、そのような使い方は推奨されません。また、文字列が半角の数字から始まる場合、および文字列が半角の空白を含む場合は、引用符でくくってやらないと文法エラーになります。

"文字列 $abc" のように、二重引用符でくくられた文字列の中に変数が含まれる場合、まず変数の展開を行なってから評価が行なわれます。単一引用符でくくられた文字列では変数展開は行なわれません。ただし単一引用符でくくった文字列をさらに二重引用符でくくった場合は、変数展開が行なわれます。

二重引用符でくくられた文字列の中に、単なる文字としての二重引用符やドル記号を記述したい場合、直前に**¥**を置いて、次の特殊文字の意味をうち消して（エスケープして）やらなければなりません。このように、ある文字が本来持っている意味をうち消してやるための記述のことをエスケープシーケンスと呼びます。また、**¥＋特定の文字** により、特殊な意味を表すエスケープシーケンスもあります。以下に一覧を示します。

表1-2 エスケープシーケンス

記述	意味
¥n	改行（ラインフィード）
¥r	復改（キャリッジリターン）
¥t	水平タブ
¥¥	バックスラッシュ
¥$	ドル記号
¥"	二重引用符
¥[0-7]{1,3}	**8**進数表記の**1**文字。**8**進数は **¥666** のようにして表す
¥x[0-9A-Fa-f]{1,2}	**16**進数表記の**1**文字。**16**進数は **¥xFF** のようにして表す

定数をシンボルとして表したものを定数シンボルといいます。たとえば**3.141592**を**PI**というシンボルで表すなど、主にプログラムの可読性を高めるために使われます。定数シンボル名は、先頭に**$**がつかないこと以外は変数シンボルと同じ命名規則にしたがいますが、慣習的に大文字で記述します。定数シンボルの定義を追加するにはdefine()関数を使います。**リスト1-39**に例を示します。

あいまいな記述を排除して可読性を高めるために、文字列定数（および文字列を使った配列キー）はかならず引用符でくくるように習慣づけておくべきでしょう。ただしphp.iniでerror_reporting=E_ALLに設定している場合、ex22.phpの2行目と8行目で、「未定義の定数を参照しようとした」という意味の警告が発せられます。

PHPで予約されている定義済み定数シンボルは非常にたくさんあります。一覧表は、PHPオンラインマニュアルの「付録G　予約語の一覧」を参照してください。定数シンボルはヒアドキュメントの内部では使えませんので注意してください。

リスト1-39 ex22.php

```
 1 <?php
 2 echo CONSTANT;                          //  "CONSTANT" を出力（文字列定数）
 3 define("CONSTANT", "Hello world.");
 4 echo CONSTANT;                          //  "Hello world." を出力（定数シンボル）
 5 $CONSTANT = "Other world.";
 6 echo $CONSTANT;                         //  "Other world." を出力（文字列変数）
 7 echo CONSTANT;                          //  "Hello world." を出力（定数シンボル）
 8 echo CONSTANt;                          //  "CONSTANt" を出力（文字列定数）
 9 CONSTANT = "Other world.";              //  文法エラー（定数には代入できない）
10 ?>
```

3.17 関数

関数とは、あるひとまとまりの手続きに名前をつけたものであり、ひとつのスカラー値またはひとつの非スカラー値を返すことができます。関数はあらゆる型の値を返すことができ、明示的に値を返さない場合はFALSEが返されます。関数を呼び出し元から見れば、その戻り値を値とする式であるといえます。

通常、関数は単に値を返すだけではなく、なにかの計算やまとまった処理を行ないます。PHPでは膨大な数の組み込み関数を用意していますが、以下のようなフォーマットを使って自分でユーザ定義関数を作ることもできます。

リスト1-40 ex23.php

```
<?php
function foo($arg1, $arg2)
{
    return $arg1 + $arg2;
}
?>
```

関数名の命名規則は定数シンボルと同じです。定数シンボルとの違いは、うしろに (がついているかどうかで見分けます。関数の本体（ボディ）は {}で囲みます。

上記の例では、関数foo()は二つの引数を取り、それらの和を返す数値型のユーザ定義関数です。関数に限らず式の型はすべて実行時に決定され、型を明示的に宣言するための文法は用意されていません。上記の関数foo()についても、戻り値の型は引数に何を渡されるかによって、整数または倍精度実数となります。

関数の中では、ほかの関数やクラス定義（後述）を含む、PHPのあらゆる有効なコードを使用することができます。

関数から複数の値を返すことはできませんが、配列を返すことにより同等のことを行なえます。

リスト1-41 ex24.php

```
<?php
/*  配列を返す関数  */
function foo() {
   return array( 0, 1, 2 );
}
list($zero, $one, $two) = foo();
?>
```

この例では変数$zero、$one、$twoに、それぞれ0、1、2が入ります。list()は、単一の操作で複数の変数に値を入れるための言語構造（厳密には関数ではありません）です。

3.18 引数

引数により関数へ情報を渡すことができます。関数側で用意した、引数を受け取るための変数を仮引数（argument）、呼び出し側で実際に指定する変数や定数を実引数（parameter）と呼びます。引数は、カンマで区切られた変数や定数のリストです。PHPでは値渡し（デフォルト）、参照渡しおよびデフォルト引数をサポートしています。さらに、受け側でfunc_get_args()関数を使えば、可変個数の引数を使えます。

リスト1-42 ex25.php

```
<?php
/*  引数として配列を取る関数    */
function takes_array($input) {
    echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
takes_array(array(1, 2));   //  "1 + 2 = 3" を表示する
?>
```

● 引数の参照渡し

デフォルトでは、関数の引数は値で渡されます。関数側で呼び出した側の引数（実引数）の内容を書き換えることができるようにするには、アンパサンド（&）を仮引数の前に付加することで、その引数を参照渡しとして定義します。もちろん実引数は変数でなければなりません。定数を渡そうとするとエラーになります。

リスト1-43 ex26.php

```
<?php
/*  引数として $bar へのポインタを取る関数  */
function foo(&$bar) {
    $bar .= '猫である';
}
$str = '我輩は';
foo($str);
echo $str;       // '我輩は猫である' を出力します
?>
```

実引数に対してアンパサンドを付加して参照渡しにすることは、最近のリリースにおいて、デフォルトでは禁止になりました。これを有効にしたい場合は、php.iniのallow_call_time_pass_referenceディレクティブを有効にしてください。

● デフォルト引数

関数に対して期待する個数の引数が渡されない場合のために、デフォルトの値を指定しておくことができます。これは「デフォルト引数」と呼ばれます。

リスト1-44 ex27.php

```
<?php
function weather($result = "晴れ" ) {
    echo "明日の天気は $result です¥n";
}
echo weather("雨");      //  "明日の天気は 雨 です"
echo weather();          //  "明日の天気は 晴れ です"
?>
```

デフォルト値は定数式である必要があり、変数や、後述するクラスのメンバであってはなりません。デフォルト引数の機能を使う場合、すべてのデフォルト値の定義は、デフォルト値を持たないすべての引数の右側にある必要があります。そうしないと期待通りの動作は得られません。次のコードを見てみましょう。これは「第2引数がない」という警告が出ます。

リスト1-45 ex28.php

```
<?php
function cook($type = "卵入り", $what)  {
    return "$type$what";
}
echo cook("お好み焼き");     // 引数が足りないといって怒られます
?>
```

次の例はうまく動作します。

リスト1-46 ex29.php

```
<?php
function cook($what, $type = "卵入り")  {
    return "$type$what";
}
echo cook("お好み焼き");              // "卵入りお好み焼き"
echo cook("お好み焼き","イカ入り");    // "イカ入りお好み焼き"
?>
```

3.19 変数のスコープ（有効範囲）

変数のスコープは、その変数が定義されたコンテキスト（文脈）です。ユーザ定義関数の外で定義されるPHP変数は大域（グローバル）スコープになります。ユーザ定義関数の内部で定義される変数は閉域（ローカル）スコープに制限されており、その関数の内部だけで有効になります。以下に例を示します。

リスト1-47 ex30.php

```
<?php
$str="TEST"; // グローバルスコープ
Function Test() {
    if (!empty($str))   return $str; // ローカルスコープ変数の参照
}
echo Test(); // 何も表示されない
?>
```

このスクリプトは何も表示しません。これは、Test()関数内部でローカル変数の$strを参照しているにもかかわらず、この関数内部のスコープでは$strに値が代入されていない（すなわち未定義である）からです。

PHPではC言語と異なり、特に宣言しないかぎり、関数内で参照される変数は自動的にローカル変数であるとみなされるので、予期せぬ変数の書き換えが起こりにくく、比較的安全です。関数内でグローバル変数を使用する場合は、関数の内部でglobal命令により明示的にグローバル変数として宣言する必要があります。**リスト1-48**に例を示します。

リスト1-48 ex31.php

```
<?php
$a=1;
$b=2;
Function Sum() {
    global $a,$b;    //  グローバルスコープの $a,$b を参照
    return $a + $b;
}
echo sum();          //  “3”を出力します。
?>
```

ただし、後述するスーパーグローバル変数($_GET、$_POST、$_REQUEST、$_SESSION、$GLOBALS)については、global宣言をしなくても使えます。

PHPには静的(スタティック)変数も用意されています。

リスト1-49 ex32.php

```
<?php
Function Test() {
    $count=0;

    echo $count;
    $count++;
}
Test();      //  0が表示される
Test();      //  やっぱり0
?>
```

この関数は、コールされるたびに$countを0に初期化してしまうので、毎回0が出力されてしまいます。関数の実行後も$countの値を保持しておきたい場合は、変数を静的に宣言します。

リスト1-50 ex33.php

```
<?php
Function Test() {
    static $count=0;     //  初回のみ初期化を行なうstatic変数
    echo $count;
    $count++;
}
Test();      //  0が表示される
Test();      //  今度は1になる
?>
```

こうすれば、Test()関数が呼ばれるたびに$countの値が増加していきます。

3.20 演算子

PHPではCやPerlから取り込んだ、多くの演算子がサポートされています。

● 算術演算子

表1-3 算術演算子

例	名前	式の値
$a + $b	加算	$aと$bの和
$a - $b	減算	$aと$bの差
$a * $b	乗算	$aと$bの積
$a / $b	除算	$aと$bでの商
$a % $b	剰余	$aを$bで割った余り（整数）
++$a	前置加算	$aに1を足してから評価する
$a++	後置加算	$aを評価してから1を足す
--$a	前置減算	$aから1を引いてから評価する
$a--	後置減算	$aを評価してから1を引く

除算演算子（/）を使用する場合、除数が0（ゼロ）の場合には警告が表示されて画面が乱れますが、ほかの処理系と異なりエラーとはならず、実行が継続されます。

リスト1-51 ex34.php

```
<?php
echo "10 / 0 = ", 10 / 0, "<BR>¥n"; // 普通は異常終了だが？
echo "10 / 1 = ", 10 / 1, "<BR>¥n";
?>
```

図1-14 ex34.phpの実行結果

```
10 / 0 =
Warning: Division by zero in a.php on line 2
10 / 1 = 10
```

表1-4にあるもの以外の算術演算については、関数というかたちで提供されます。詳細は第3部の関数リファレンスや、付属CD-ROMに収録してあるPHPマニュアルを参照してください。

● **ビット演算子**

ビットとは整数を2進数として評価した場合の各桁のことで、各桁は0または1の値を取ります。ビット演算子は特定のビットをオン（1）またはオフ（0）にします。ビット演算は算術演算と違って各桁ごとに演算が閉じており、ある桁の計算結果がほかの桁に影響を与えるといったことはありません。

表1-4　ビット演算子

例	名前	式の値
$a & $b	論理積（and）	$aおよび$bの両方が1であれば1
$a ¦ $b	論理和（or）	$aまたは$bのどちらかが1であれば1
$a ^ $b	排他的論理和（xor）	$aと$bが異なる場合に1
~ $a	否定（not）	$aが1なら0、0なら1
$a << $b	左シフト	$aのビットを左に$bビットずらす （算術的には「2をかける」ことを意味する）
$a >> $b	右シフト	$aのビットを右に$bビットずらす （算術的には「2で割る」ことを意味する）

0xff（16進数のff＝10進数の255）は、下8ビット（1バイト）がすべて1の値です。0xff（＝0x000000ff）とandを取るということは、最下位バイトに属するビット以外をすべて0に（クリア）するということです。これを10進数で考えると、256（2の8乗）で割った余りを算出することと同義になります。また0xffとorを取るのは、下位8ビットをすべて1にすることです。以下の例を見てください。

リスト1-52　ex35.php

```
<?php
    printf("0x1234 & 0xff = 0x%X<BR>¥n", 0x1234 & 0xff);
    printf("0x1234 | 0xff = 0x%X<BR>¥n", 0x1234 | 0xff);
    printf("~ 0xffff = 0x%X<BR>¥n", (~ 0xffff) & 0xffff);
?>
```

図1-15　ex35.phpの実行結果

```
0x1234 & 0xff = 0x34
0x1234 | 0xff = 0x12FF
~ 0xffff = 0x0
```

ここで、printf()は書式つきの出力を行なう組み込み関数です。文字列中の%Xは、対応する引数を整数値として16進数で出力するよう指示しています。詳細は第3部の関数リファレンスを参照してください。

● **文字列演算子**

文字列演算子は一種類、つまり文字列結合演算子（.）しかありません。

リスト1-53 ex36.php

```
<?php
$a = "Hello ";
$b = $a . "World!";
echo $b;              // "Hello World!"
?>
```

● **代入演算子**

いままで何気なく使ってきた=ですが、これは「右辺を左辺に代入する」という演算子です。派生型として、ほかの算術演算子と一緒になった複合演算子があります。以下に一覧を示します。

表1-5 代入演算子

例	名前	式の値
$a = $b	代入	$aに$bの値を代入。$aの型は$bと同じになる
$a += $b	加算代入	$aに$bの値を加算。両辺とも数値とみなされる（以下同様）
$a -= $b	減算代入	$aから$bの値を減算
$a *= $b	乗算代入	$a*$bの値を$aに代入
$a /= $b	除算代入	$a/$bの商を$aに代入
$a %= $b	剰余代入	$a/$bの剰余（整数）を$aに代入
$a &= $b	And 代入	$a&$bの値を$aに代入
$a ¦= $b	Or 代入	$a¦$bの値を$aに代入
$a .= $b	連結代入	$aに$bの値を連結。両辺とも文字列とみなされる
$a =& $b	参照 代入	$aに$bの記憶域のアドレスを代入。$bが$aを指すようになる
$a <<= 1	左シフト	$aの内容を左に1ビットシフト
$a >>= 1	右シフト	$aの内容を右に1ビットシフト

参照による代入の例を以下に示します。

リスト1-54 ex37.php

```
<?php
$a = "ABC¥n";
$b =& $a;
$b = "DEF¥n";
printf("¥$a=%s¥n", $a);          //   “$a=DEF”を出力します
?>
```

● **比較演算子**

CやPerlなどと同様に、PHPにも真偽値があります。PHPではすべての非ゼロの数値はTRUE（真）で、ゼロはFALSE（偽）です。負の値は非ゼロなのでTRUEとみなされます。文字列に関しては、空文字列と文字列"0"はFALSEですが、そのほかの文字列はすべてTRUEです。非スカラー値（配列とオブジェクト）に関しては、要素があればTRUE、なければFALSEとみなされます。

比較演算子による式は、TRUEまたはFALSEいずれかの真偽値をとります。これらの式は、一般的にはIF文のような条件式の内部で使用されます。

リスト1-55 ex38.php

```
<?php
if (10==10.0):  //  この式が成り立つので
    echo "10==10.0 です。<BR>¥n";   //  ここを通る
else:
    echo "10!=10.0 です。<BR>¥n";
endif;
?>
```

● **三項演算子**

```
$first ? $second : $third;
```

最初の部分式$firstの評価結果がTRUE（非ゼロ）の場合、二番目の部分式$secondが評価され、この条件式全体の値となります。そうでない場合、三番目の部分式$thirdが評価され、この式の値となります。直前の例を三項演算子で書くと、以下のようになります。うまく使えばコーディングがすっきりしますが、慣れるまでは読みにくいかもしれません。

リスト1-56 ex39.php

```
<?php
printf("10%s10.0です¥n", (10==10.0) ? "==" : "!=");
                                                //  10==10.0 です
?>
```

● **論理演算子**

論理演算子を含む式は、前述の比較演算子と同様に真（TRUE）または偽（FALSE）いずれかの真偽値をとります。andおよびor演算子が二種類あるのは、演算が行なわれる際の優先順位が異なっているものを提供しているためです（次節の「演算子の優先順位」を参照してください）。

表 1-6　論理演算子

例	名前	式の値
$a and $b	論理積	$aおよび$bの両方がともに真の場合に真
$a or $b	論理和	$aまたは$bのどちらかが真の場合に真
$a xor $b	排他的論理和	$aまたは$bのどちらかが真でかつ両方とも真でない場合に真
! $a	否定	$aが真でない場合に真
$a && $b	論理積	$aおよび$bがともに真の場合に真
$a ¦¦ $b	論理和	$aまたは$bのどちらかが真の場合に真

● 実行演算子

PHPは一種類の実行演算子、バッククォート（`）をサポートします。シングルクォートではないことに注意してください。PHPは、バッククォートの中身をシェルコマンドとして実行し、その出力を返します。すなわち、出力を単にダンプするのではなく、変数に代入することができます。

リスト 1-57　ex40.php

```
<?php
$output = `ls -al`;
echo "$output";
?>
```

図 1-16　ex40.phpの実行結果

（カレントディレクトリにおけるファイル一覧が表示される）

なお、バッククォートでくくられたコマンドが実行できるかどうかは、PHPが何というユーザで動作中であるのか、また実行するコマンドのオーナなど、オペレーティングシステムの環境に依存します（何でも実行できるようだと、セキュリティホールになってしまう可能性があります）。本書で推奨しているインストール方法では、PHPはApacheのDSOモジュールとして動作するので、PHPはApacheと同じapache権限で実行されることになり、その環境から実行されるコマンドも同じapacheユーザで動きます。何も表示されないなど、期待した動作にならない場合には、Apacheのエラーログ[*15]にエラー内容が記載されているので参照してください。

実行演算子と類似の機能を持つ組み込み関数として、system()、passthru()、exec()、popen()、 escapeshellcmd()があります。php.iniのsafe_modeをOnにしている場合は、バッククォートは使用できず、これらの実行関数も制限を受けます。詳細は第3部を参照してください。

*15
本書のインストールにしたがえば、エラーログファイルは/etc/httpd/logs/error_log（RPMからのインストール）または /usr/local/apache/logs/error_log（ソースからのインストール）にあります。

3.21 演算子の優先順位

演算子の優先順位は、二つ以上の演算子を含む式において、式のどの部分から順に評価されるのかを決定するものです。たとえば、式 **1+5*3** の答えは通常の算術演算では16になり、18とはなりません。これは乗算演算子（*****）は加算演算子（**+**）より優先度が高いからです。

表 1-7　演算子の優先順位

優先度	結合時の評価	演算子
⇧ 低い	左	,
	左	or
	左	xor
	左	and
	右	print
	左	= += -= *= /= .= %= &= ¦= ^= ~= <<= >>=
	左	? :
	左	¦¦
	左	&&
	左	¦
	左	^
	左	&
	結合しない	== != ===
	結合しない	< <= > >=
	左	<< >>
⇩ 高い	左	+ - .
	左	* / %
	右	! ~ ++ -- (int) (double) (string) (array) (object) @
	右	[
	結合しない	new

表 1-7 の中における「結合時の評価」とは、**式A 演算子 式B** という式があるとき、その演算子は式Aと式Bのどちらに対する演算であるのかを示します。

3.22 制御構造

PHPでは条件分岐や繰り返し（ループ）、ループからの脱出など、一般のプログラミングに必要な制御構造が用意されています。

● IF～ELSEIF～ELSE～ENDIF

IF文は命令の条件実行を行ないます。実行文が複数ある場合は大括弧{}で囲んで条件ブロックを作り、その中に実行文を記述します。「2.7　疑わしきは検証せよ」で述べたように、条件式の中では式は論理値で評価されます。式がTRUEと評価された場合、PHPは文を実行します。FALSEと評価された場合はこれを無視します。

リスト1-58　ex41.php

```
<?php
$visit_count = 0;
IF ($visit_count==0)
  echo "はじめまして(^O^)¥n";
ELSEIF ($visit_count==1)    {
  echo "おひさしぶりです。¥n";
  echo "二度目ですね(^^)v¥n";
} ELSEIF ($visit_count>1)
  echo "いつもありがとうございますm(__)m¥n";
ELSE
  echo "エラーが発生しました(;_;)";
?>
```

ENDIF構文を使用すると、上記の処理は以下のようにも書けます。評価式のうしろのコロン（:）を忘れないように注意してください。ただしこの構文については、将来は廃止される可能性もあります。これは後述のENDWHILE構文などについても同様です。

リスト1-59　ex42.php

```
<?php
$visit_count = 0;
IF ($visit_count==0):    // コロン（:）をつける（以下同様）
  echo "はじめまして(^O^)¥n";
ELSEIF ($visit_count==1):    {
  echo "おひさしぶりです。¥n";
  echo "二度目ですね(^^)v¥n";
} ELSEIF ($visit_count>1):
  echo "いつもありがとうございますm(__)m¥n";
ELSE:
  echo "エラーが発生しました(;_;)";
ENDIF;                    // セミコロン（;）をつける
?>
```

IF文は無限に入れ子にできます。これにより、プログラムのさまざまな部分で柔軟な条件分岐ができます。

●WHILE～ENDWHILE

WHILE文は、条件式の値が真である間、ブロック中の処理を繰り返し実行します。条件式の値が始めから偽となる場合は、処理は一度も実行されません。

WHILEループは最も簡単なタイプのループです。以下のスクリプトを実行すると永久ループとなり、ブラウザの中止ボタンを押すまで無限に出力が行なわれます。

リスト1-60 ex43.php

```
<?php
WHILE(1)    echo 'これは無限ループです';  // 不用意に実行しないほうが …
?>
```

条件つきループの例です。

リスト1-61 ex44.php

```
<?php
srand();                    //  乱数の初期化
$i = rand();                //  乱数の生成
WHILE($i>0) {
    echo 'この部分が実行されるかどうかは、$iの値次第です。¥n';
}
?>
```

ENDIFと同様に、ENDWHILE構文もあります。

リスト1-62 ex45.php

```
<?php
$i = 0;
WHILE($i<10):    // コロン（:）をつけることに注意
    echo "¥$i は現在 $i です。<BR>¥n";
    $i++;
ENDWHILE;        // セミコロン（;）をつけることに注意
?>
```

●DO～WHILE

DO～WHILEループは、各繰り返しの最後に論理式のチェックを行ないます。このためWHILEループとは異なり、最低1回のループ実行が保証されます。

リスト1-63 ex46.php

```
<?php
$i = 0;
DO {
    echo "ここは最低でも1回は通ります。";
} WHILE ($i==1);
?>
```

●FOR

リスト1-64 ex47.php

```
<?php
FOR ($i=1, $sum=0; $i<=10; $i++)  {
     $sum += $i;
}
echo "¥$sum = $sum";  //  55 が出力される。
?>
```

FOR文は、C言語のforループと同様に動作します。FORは**(式1; 式2; 式3)**と三つの式をとります。式1はループが始まる前に一度だけ評価（実行）され、主にループカウンタ（ループを実行する回数を制御するための変数）を初期化するのに使われます。次にループの終了判定である式2が評価され、条件を満たしていれば{}でくくられたブロック部分が評価されます。そしてブロック部分を1回評価するごとに式3が評価されます。これを利用してループカウンタのインクリメント（1、あるいは定数を足すこと）を行なうことが多いようです。式3の評価のあとで再度式2が評価され、これが真の場合再びループを実行します。式2が偽になるとそのループを抜けます。各式のいずれか、またはすべてを空とすることもできます。式2を空にすると暗黙のうちに真と評価され、無限ループとなります。

FORループはWHILEループに比べて5%程度のオーバーヘッド（処理にかかるコスト）がかかりますが、よりスマートに書けます。

●BREAK

BREAK文は、WHILE、DO～WHILE、FORループ構造およびSWITCH（後述）の内部で、現在のループ構造を脱出するのに使用します。またC言語と異なり、引数としてネストレベルを指定することにより、ネストした（入れ子になった）任意のループからの脱出ができます。

リスト1-65 ex48.php

```
<?php
for ($i=0; $i<10; $i++) {
    for ($j=0; $j<10; $j++) {
        if ($j>1)   break;      // 内側のループを抜ける
        if ($i>2)   break 2;    // 外側のループを抜ける
        echo "¥$i = $i, ¥$j = $j<BR>¥n";
    }
}
?>
```

図1-17 ex48.phpの実行結果

```
$i = 0, $j = 0
$i = 0, $j = 1
$i = 1, $j = 0
$i = 1, $j = 1
$i = 2, $j = 0
$i = 2, $j = 1
```

●CONTINUE

CONTINUE文はループ構造の始めまでジャンプします。

リスト1-66 ex49.php

```
<?php
for ($i=0; $i<10; $i++) {
    if ($i%2)   continue; // 奇数ならスキップ
    echo "$i, ";          // ($i%2)は($i%2 != 0)と同じ意味です
}
?>
```

図1-18 ex49.phpの実行結果

```
0, 2, 4, 6, 8,
```

●SWITCH

SWITCH文は同じ変数を異なる値と比較し、値に応じて異なった処理を実行します。

リスト1-67 ex50.php

```
<?php
$button = "修正";              //   未定義の場合、警告が出るので
switch ($button)    {
    case "登録":
        print "登録処理を行ないます";
        break;
    case "修正":
        print "修正処理を行ないます";
        break;
    case "削除":
        print "削除処理を行ないます";
        break;
    default:
        print "デフォルト処理を行ないます";
}
?>
```

SWITCH文はループの一種であるとみなされます。もしSWITCH文の内部に break 2; と記述すると、その SWITCHから抜け、さらにネストした一番内側のループから抜けます。

● FOREACH

これは配列要素に対する反復処理を効率的に書くために用意されたものです。これには二種類の構文があります。

```
FOREACH(配列表現 as $value) 文
FOREACH(配列表現 as $key => $value) 文
```

最初の形式は、配列表現で指定した配列に関してループ処理を行ないます。各ループにおいて、現在の要素の値が$valueに代入され、内部配列ポインタがひとつ前に進められます。以下に例を示します。

リスト1-68 ex51.php

```
<?php
$fruits = array("りんご", "バナナ", "みかん");
foreach ($fruits as $val) { print("$val¥n"); }
?>
```

二番目の形式も同様ですが、各ループでさらに現在の要素のキーが変数$keyに代入されます。

リスト1-69 ex52.php

```
<?php
$fruits = array("りんご" => "100", "バナナ" => "200", "みかん" => "300");
printf("品名¥t単価¥n");
foreach ($fruits as $key => $val)   {
    print("$key¥t{$val}円¥n");
}
?>
```

¥t はタブコードを表すエスケープシーケンスです。また、$val円という変数名と誤認されないように、{$val}で変数名の範囲を明示しています。

図1-19 ex52.phpの実行結果

```
品名      単価
りんご    100円
バナナ    200円
みかん    300円
```

FOREACHの実行開始時には、内部配列ポインタは配列の先頭要素を指すように自動的にリセットされます。このため、FOREACHループの前にreset()をコールする必要はありません。

●REQUIRE

REQUIRE文は、Cプリプロセッサの#include文の動作と同じく、自分自身を指定したファイルで置き換えます。置き換えはスクリプトの構文解析に先立って行なわれます。REQUIRE()文をループ構造の中に置いて、繰り返しの途中で毎回異なったファイルの内容を読み込むようなことはできません。そのような用途にはINCLUDE文を使用してください。

```
<?php
REQUIRE('header.inc');
?>
```

●INCLUDE

INCLUDE文は指定したファイルを読み込み、評価します。INCLUDE文が処理されるたびにファイルの読み込みが行われます。そのため、ループ構造の内部でINCLUDE文を使用し、異なったファイルを読み込むことができます。

```
<?php
$files = array('first.inc', 'second.inc', 'third.inc');
for ($i = 0; $i < count($files); $i++) {
    include($files[$i]);
}
?>
```

INCLUDEはREQUIREと異なり、その文の処理が行われるたびに（実行時においてのみ）再度評価されます。一方REQUIRE文は、仮に条件が成立しないようなブロックの中にある場合でも、最初に読み込まれたファイルで無条件に置き換えられます。

ファイルが評価された際、パーサ（構文解析ルーチン）はHTMLモードとなっており、PHP開始タグ（<?php）に出合うまでファイルの中身が出力されます。関数リファレンスのreadfile()、virtual()も参照してください。

なお、require()やinclude()される側のファイルの中で再度include()をしたりすると、同じファイルが2回以上読み込まれて再定義エラーになったりする場合があります。これを防ぐために、一度だけしか評価を行わないrequire_once()やinclude_once()が用意されています。

3.23 クラスとオブジェクト

変数と、それに付随する固有の手続きをカプセル化したものをクラスと呼びます。言い換えれば、クラスとは独自の手続きロジックを包含する、拡張された変数のための型宣言です。実際にクラスを使うためには、そのクラスの型を持つ変数（オブジェクト）を生成する必要があります。これにはnew命令を使います。サンプル（**リスト1-70**、**リスト1-71**）を見てください。

クラスの宣言はclass命令で始まります。var命令で、そのクラス内部でのみアクセス可能な変数を定義します。これをそのクラスのメンバ変数と呼びます。メンバ変数は、クラスの外ではクラスの実体（オブジェクト）を通して間接的にしかアクセスできません。これをメンバ変数は外から「隠蔽」されているといいます。すなわち関数内のローカル変数と同様の扱いです。

クラスの内部には、複数のユーザ定義関数（メンバ関数）が定義できます。メンバ関数の中からはメンバ変数に自由にアクセスできますが、その際メンバ変数を引数で渡してもらう必要がありません。この意味では、メンバ変数はそのクラス内部ではグローバル変数のような性格を持っているといえます。メンバ関数からメンバ変数にアクセスするには、**$this**という特別の変数を通して **$this-> メンバ変数名**という書式でアクセスします。メンバ変数名自体には**$**記号はつけません。**$this**と書くのをうっかり忘れてしまっても、文法エラーとはならないので注意してください。この場合は「メンバ変数と同名の、メンバ関数内固有のローカル変数」にアクセスすることになり、望む結果が得られません。なお、クラス内部から（自分自身を含む）メンバ関数を呼び出すのにも、**$this-> メンバ関数名**という書式を使います。

リスト1-70 ex53.php

```
<?php
class foo {                             //  クラスの宣言
    var $bar;                           //  メンバ変数

    function    foo()       {           //  コンストラクタ
        $this->bar = 10;                //  メンバ変数の初期化
        $this->display("foo()");
    }

    function    add()       {           //  メンバ関数
        ++$this->bar;
        $this->display("add()");
    }

    function change($val=0) {
        $this->bar = $val;
        $this->display("change()");
    }

    function    display($func)    {
        printf("%s: 現在の bar の値は %d です。¥n", $func, $this->bar);
    }
}
?>
```

リスト1-71 ex54.php

```
<?php
require('ex53.php');
$baz = new foo;                         //  オブジェクトの生成
$baz->add();
$baz->add();
$baz->change(5);
$baz->change();                         //  デフォルト引数を使う
?>
```

図1-20 ex53.phpの実行結果

```
foo(): 現在の bar の値は 10 です。
add(): 現在の bar の値は 11 です。
add(): 現在の bar の値は 12 です。
change(): 現在の bar の値は 5 です。
change(): 現在の bar の値は 0 です。
```

メンバ関数の中でも、特にそのクラスと同じ名前を持つ関数を「コンストラクタ(constructor ― 構築子)」と呼びます。コンストラクタは、そのクラスのオブジェクトをnewで生成する際に自動的に暗黙に呼ばれる関数で、初期化の処理を記述しておきます(なくてもかまいません)。オブジェクトが消滅する際に呼び出されるデストラクタ(destructor ― 消滅子)は、PHPではサポートされていません。

リスト3-53の例では、foo()、add()、change()でそれぞれメンバ変数$varの値を変化させています。change()メンバ関数では、呼び出し側で引数が指定されなかった場合のデフォルト引数を定義しています。呼び出し側ではnewでそのクラスのオブジェクトを定義したあと、**オブジェクト名 -> メンバ関数名()** でメンバ関数を呼び出しています。

クラスには「継承」という概念があります。これは、いったん汎用的な手続きを定義したクラスを宣言しておき、さらにそのクラスから「派生」した、独自の手続きが追加されたクラスを宣言するというものです。派生クラスでは、その親(基底)クラスで定義された内容を再定義することなく、そのまま使用することができます。さらに、親クラスで定義されたメンバ関数と同じ名前の関数を再定義(オーバライド)することによって、一部のメンバ関数に関する振る舞いだけを変えることができます。上記のfoo()クラスを読み込んだ状態で、さらにそれを継承したクラスfoo2()を定義してみましょう。

リスト1-72 ex55.php

```
<?php
require('ex53.php');              //  基底クラスの定義
class   foo2    extends foo {
    function    foo2()  {         //  コンストラクタ
        $this->foo();             //  基底クラスのコンストラクタ呼び出し
    }

    function    change($val=3)  {
        $this->bar = $val * 2;    //  二倍してから格納
        $this->display("change()");
    }
}

$baz = new foo2;                  //  オブジェクトの生成
$baz->add();                      //  基底クラスのメンバ関数の呼出し
$baz->add();
$baz->change(5);                  //  派生クラスのメンバ関数の呼出し
$baz->change();                   //  デフォルト引数を使う
?>
```

図 1-21　ex55.php の実行結果

```
foo(): 現在の bar の値は 10 です。
add(): 現在の bar の値は 11 です。
add(): 現在の bar の値は 12 です。
change(): 現在の bar の値は 10 です。
change(): 現在の bar の値は 6 です。
```

派生クラスを定義する場合、extends 命令で基底クラスの名前を指定します。複数の基底クラスからの多重継承はサポートされていません。また、PHP では派生クラスのコンストラクタで基底クラスのコンストラクタを自動的に呼び出してはくれないので、必要があれば明示的に呼び出しておきます。

この例題では、add()メンバ関数については foo クラスと同じ振る舞いでかまわないので foo2 クラスの中では特に定義していませんが、呼出し元からは foo クラスの場合と同じように呼び出せます。change()メンバ関数については、デフォルト引数の値を変えたいのと、指定された値の 2 倍で変数を更新するように新たに定義しました。新しい change()のほうが呼ばれていることを確認してください。

3.24 スーパーグローバル変数

PHP 4.2.0 より php.ini の register_globals がデフォルトで Off になり、それまでのように、フォーム変数について気軽に **＄変数名** でアクセスすることができなくなりました。それ以外にも大きな変更がありましたが、まとめて「スーパーグローバル変数」として紹介します。なお、これは PHP 4.1.0 で登場した概念です。

これらの連想配列は、通常の変数と異なり、スクリプト中のいかなる場所においても global 宣言をすることなく参照できるため、スーパーグローバル変数と呼ばれています。スーパーグローバル変数には以下のものがあります。

表 1-8　スーパーグローバル変数

変数名	従来の名前	用途
$_SERVER	$HTTP_SERVER_VARS	実行環境情報などを保持する
$_ENV	$HTTP_ENV_VARS	環境変数
$_POST	$HTTP_POST_VARS	HTTP POST メソッドで与えられた変数
$_GET	$HTTP_GET_VARS	HTTP GET メソッドで与えられた変数
$_COOKIE	$HTTP_COOKIE_VARS	HTTP クッキーを通して与えられた値
$_FILES	$HTTP_POST_FILES	ファイルアップロード時にセットされる
$_REQUEST	(なし)	GET/POST/COOKIE/FILES の内容を保持する
$_SESSION	(なし)	セッション管理対象の変数名とその値を保持する

● **$_SERVER**

従来、環境周りの情報を取得するために使用していた $PHP_SELF（スクリプト名）、$HTTP_USER_AGENT（クライアントのソフトウェア識別文字列）、$REMOTE_ADDR などがすべて $_SERVER[]としてまとめられました。今後は $_SERVER['PHP_SELF']、$_SERVER['HTTP_USER_AGENT'] などとしてアクセスすることになります。ほかにどのようなメンバがあるのかについては、後述する「5.4.4　phpinfo()」や、付属CD-ROMに収録してあるPHPマニュアルの「定義済みの変数」の章を参照してください。

コマンドライン版（/usr/bin/php）の場合は、PHP_SELFなど$_SERVERの一部がセットされなくなります。

● **$_ENV**

PHP実行時の環境変数を保持しています。これらの値はgetenv()を使って個別に取得できるものです。PHPがDSOで組み込まれている場合は、サーバの実行時の環境変数となります。

● **$_POST、$_GET**

HTMLフォームのMETHODでPOSTを指定した場合、スクリプト側は**$_POST['変数名']**で入力を受け取ります。METHODにGETを指定もしくは何も指定しない場合、およびURLに**?変数名＝値**形式で渡された引数については、**$_GET['変数名']**で受け取ります。

● **$_COOKIE**

クライアントからクッキーとして送り込まれた値を保持します。

● **$_FILES**

ファイルのアップロード機能（後述）によりクライアントから送り込まれたファイルの情報を保持します。

上記の連想配列すべてについて、同じキーが重複して存在する場合、どの配列の値が有効になるかは、評価する順序で決まります。デフォルトは EGPCS（それぞれの変数の頭文字）となっており、たとえば、$_ENV['abc'] の値は $_GET['abc'] の値で上書きされます。この評価順序はphp.iniのvariables_orderにより変更できます。

● **$_REQUEST**

$_GET、$_POST、$COOKIE、$FILESにあるキーとその値をすべて保持します。

● **$_SESSION**

これはスクリプト内部で自由に使うことができる、セッション管理対象となる連想配列です。この配列に入れた値は、スクリプトの終了時にsession.save_handlerで指定された方法（デフォルトはファイル）で自動的に保存されます。session_start()が呼ばれると（session.auto_start=1の場合はスクリプトの起動時に自動的に）、そのファイルが読み込まれ、$_SESSIONに値がセットされます。セッションに関するディレクティブや関数は多数あります。詳細についてはPHPマニュアルを参照してください。使い方は第4章で紹介しています。

Hypertext Preprocessor

Chapter-4 PHPを使いこなそう

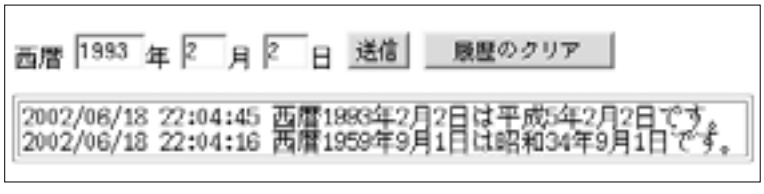
4.1 実用的なサンプル

本章では、第2章で紹介したサンプルに機能を追加し、多少なりとも実用的なサンプルにしてみます。第2章のサンプルでは西暦の年を入力して、それが明治/大正/昭和/平成の各々何年にあたるかを表示するというものでした。この章では、以下のような肉づけを行なってみましょう。

・年月日の入力
・厳密な入力チェック。文字種や数値の範囲チェック、および閏年判定を含む、正確な妥当性チェック
・正確な和暦変換。各和暦の境界を日単位でチェックすることにより、元号の変わり目の日も正確に判定
・実行履歴の表示。過去に表示された判定結果を逐次表示

まずは**図1-22**の実行画面を見てください。

図1-22 test9.phpの実行画面



http://localhost/~hotta/1-4/test9.php にアクセスすると年月日の入力画面が現れます。適切な入力を行なって送信ボタンを押すと、入力した値がそのまま入力エリアに表示されるとともに、それを和暦に直した値が、それまでの入力の履歴とともに下の枠の中に表示されます。この履歴は履歴のクリアボタンを押すと消去されます。不適切な入力を行なうと、赤でエラーメッセージが表示されます。では、その実現方法を見ていきましょう。

リスト1-73 test9.php

```
<?php
//-----------------
// 入力画面の表示
//-----------------
function display_input_area()
{
    require("expand.inc");              //  $yyyy,$mm,$ddの展開
    print   <<<EOD
<FORM METHOD=POST ACTION="{$_SERVER['PHP_SELF']}">
西暦
<INPUT TYPE="text" NAME="yyyy" SIZE="5" VALUE="$yyyy">年
<INPUT TYPE="text" NAME="mm" SIZE="3" VALUE="$mm">月
<INPUT TYPE="text" NAME="dd" SIZE="3" VALUE="$dd">日
<INPUT TYPE="submit" NAME="submit" VALUE="送信">
<INPUT TYPE="submit" NAME="clear" VALUE="履歴のクリア">
</FORM>
EOD;
}
//--------------------------
// 入力値のチェック
//--------------------------
function date_is_valid()
{
    require("expand.inc");              //  $yyyy,$mm,$ddの展開
    if (! ereg("^[0-9]{4}$", $yyyy)) return FALSE;
    if (! ereg("^(0?[1-9]|1[0-2])$", $mm)) return FALSE;
    if (! ereg("^(0?[1-9]|1[0-9]|2[0-9]|3[01])$", $dd)) return FALSE;
    if ($dd==31) {                      //  大の月
        if ($mm==2 || $mm==4 || $mm==6 || $mm==9 || $mm==11) {
            return FALSE;
        }
        return TRUE;
    }
    if ($dd==30) {
        if ($mm==2) {
            return FALSE;
        }
        return TRUE;
    }
    if ($mm==2 && $dd==29) {
        if ($yyyy % 4 == 0) {           //  4で割れれば閏年
            if ($yyyy % 100 == 0) {     //  しかし100で割れれば平年
                if ($yyyy % 400 == 0) { //  しかし400で割れればやっぱり…
                    return TRUE;
```

```
45                  }
46                  return FALSE;
47              }
48              return TRUE;
49          }
50          return FALSE;
51      }
52      return TRUE;
53  }
54  //---------------------------
55  // 元号の計算
56  // 返り値：計算結果の文字列
57  //---------------------------
58  function calc_wareki()
59  {
60      require("expand.inc");                //  $yyyy,$mm,$ddの展開
61      $border = array(
62          array("開始日"=>18680908, "終了日"=>19120730, "元号"=>"明治"),
63          array("開始日"=>19120730, "終了日"=>19261225, "元号"=>"大正"),
64          array("開始日"=>19261225, "終了日"=>19890107, "元号"=>"昭和"),
65          array("開始日"=>19890107, "終了日"=>20091231, "元号"=>"平成")
66      );
67      $target = sprintf("%04d%02d%02d", $yyyy, $mm, $dd);
68      if ($target < $border[0]['開始日']) return "計算対象外";
69      for ($i=0; !empty($border[$i]); $i++)   {
70          if ($border[$i]['開始日'] <= $target &&
71              $target <= $border[$i]['終了日']) {
72              $wareki = $yyyy - substr($border[$i]['開始日'], 0, 4) + 1;
73              break;
74          }
75      }
76      if ($i > 3) return "計算対象外";
77      return sprintf("%s%s年%d月%d日",
78          $border[$i]['元号'], ($wareki == "1") ? "元" : $wareki, $mm, $dd);
79  }
80  //-----------------
81  //  メイン
82  //-----------------
83  session_start(); // セッションを開始／復帰
84  require("expand.inc");                          //  $yyyy,$mm,$ddの展開
85  display_input_area();                           // 入力エリアの表示
86  if (!empty($_POST['clear']))        {           //  「履歴のクリア」が押された
87      if (!empty($_SESSION['hist']))  {
88          unset($_SESSION['hist']);               // 履歴を破棄
89      }
```

```
90      exit;
91  }
92  if (date_is_valid()) {                          // 入力のチェック
93      $wareki = calc_wareki();                    // 元号の計算
94      $result = sprintf("西暦%d年%d月%d日は%sです。<br>¥n",
95          $yyyy, $mm, $dd, $wareki);
96  } else {
97      if (!empty($yyyy) || !empty($mm) || !empty($dd)) {
98          print("<font color=red>日付の入力が誤っています</font>\n");
99      }
100 }
101 if (!empty($result))    {                       // 履歴に追加＆表示
102     $hist = !empty($_SESSION['hist']) ? $_SESSION['hist'] : "";
103     $_SESSION['hist'] = date("Y/m/d H:i:s ") . $result . $hist;
104     print("<table border>¥n<tr><td>¥n{$_SESSION['hist']}</table>¥n");
105 }
106 ?>
```

リスト1-74　expand.inc

```
<?php
    $yyyy = !empty($_POST['yyyy']) ? $_POST['yyyy'] : "";
    $mm = !empty($_POST['mm']) ? $_POST['mm'] : "";
    $dd = !empty($_POST['dd']) ? $_POST['dd'] : "";
?>
```

4.2 入力画面の表示

入力画面を表示する部分はdisplay_input_area()という関数にしました。特に難しいことはしておらず、単なるHTMLドキュメントを作って表示しているだけです。ヒアドキュメントの中で変数を使っているので、変数の中味によって生成されるHTMLの成果物が異なります。

●7行目

require文でスクリプトexpand.inc（**リスト1-74**）を読み込んでいます。これ以外の関数の先頭でも、可読性を高めるために同じことをしています。**$_POST['変数名']**には、HTMLのフォームにおいて<INPUT>タグで指定した入力値が入ってきます。$_POSTの詳細については「3.24　スーパーグローバル変数」を参照してください。

ここでは三項演算子を使い、**$_POST['変数名']**が空でなければ**$変数名**にその値を、そうでなければ（空であれば）"" を入れています。関数display_input_area()の中で

expand.incをrequire()しているので、こうして生成された$yyyy、$mm、$ddは関数内部のローカル変数となり、global宣言なしでもそのまま使えるようになります。

●8行目

ヒアドキュメントの始まりです。

●9行目

ヒアドキュメントの中は""で囲まれているのと同様の扱いになり、配列変数を使う場合はその前後を{}でくくらないと文法エラーになります。$_SERVER['PHP_SELF'] は実行しているスクリプト名（URLで指定された名前）を保持しています。この例では

```
<FORM METHOD=POST ACTION="/~hotta/1-4/test9.php">
```

などと書くのと同義ですが、常にこのように書くよう習慣づけておけば、スクリプト名の変更に柔軟に対応できます。

●11～13行目

VALUE= の値は、expand.incによりそれぞれPOSTされた値に置き換わります。これで「前回入力された値をデフォルト値として表示する」機能を実現しています。

4.3 入力値（日付）のチェック

関数date_is_valid()は $_POST[]から年月日を受け取り、その日付が有効な場合は真、そうでなければ偽を返します。

●25～27行目

入力値が数字だけから構成されている文字列かどうかを調べています。25行目の正規表現は「4桁の数字の並び」、26～27行目はそれぞれ「1～12」「1～31」かどうかの判定です（正規表現のパターンについては第2章のコラムで解説しています）。このように、明らかな入力ミスは最初から除外しておけば、それ以降の複雑なチェックを行なう必要がなくなり、処理速度の向上が図れ、間違いも少なくなります。

●28～33行目

日として31が入力された場合のチェックです。これは大の月しかあり得ませんから、小の月ならエラーとしています。大の月なら正常なので真を返して関数を抜けています。小の月の判定をereg()と正規表現で書くとどうなるか、考えてみてください。

●34～39行目

日として30が入力された場合のチェックです。ここは小の月の場合ですから、2月以外が正常となります。

●40～51行目

閏年のチェックです。これは2月29日の場合にのみチェックするようにして、それ以外の場合に余計な計算を行なわないようにしています。考え方としては以下のとおりです。

・4で割り切れる年は閏年
・例外的に、100で割り切れる年は平年
・さらに例外的に、400で割り切れる年は閏年

●52行目

31日および30日以外でかつ2月29日以外（つまり各月の27日以前、または2月28日）なら、ここまでたどりついてTRUE（妥当な入力）となります。

たかが日付チェックといいながら、きちんとやろうとすると結構なステップ数になります。

4.4 和暦への変換

和暦への変換ルーチンも、だいぶ本格的なものになってきました。汎用性には欠けますが、$_POSTから年月日を受け取って、**XX（元号名）yy年mm月dd日**という文字列を返すようなインタフェースにしてみました。サポートする年の範囲を超えた場合は「計算対象外」を返します。

●61～66行目

変数$borderは、明治～平成までの各元号の開始日と終了日、および元号名を保持する二次元配列です。array()を入れ子で使って初期化しています。平成についてはまだ終了日はありませんが、とりあえず2009年12月31日までを対象とし、それ以降の日付および明治元年以前は「計算対象外」ということで除外しています。

●67行目

入力された年月日を8桁の文字列に変換しています。$borderで年月日を別々に持たずに**yyyymmdd**と8桁の文字列で保持するようにしたので、if()の条件判定が開始終了で各々1回ずつで済んでいます。

●68行目

**$border[0][開始日]**（すなわち明治元年の最初の日）と比較し、それより以前の場合は計算対象外としてreturnします。$borderの一番目のキーは、array()で定義された順に0から3までの番号が振られています。

●69～75行目

$border[0]～$border[3]それぞれの開始日および終了日の間に入力日が入っているかを判

定し、それで入力日が属する元号を決定します。広辞苑によれば、各元号の開始/終了日はその前後の元号と重複するようなので、先にマッチしたほうの元号（すなわち時期的に前にあるもの）が出力されます。

●70～71行目

開始日から終了日の範囲内に入っているかどうかを判定しています。

●71行目

この元号に該当すれば、一時変数$warekiに和暦の年を格納しています。substr()は文字列の一部を取り出す関数です。ここでは**$border[$i][開始日]**の0バイト目から4文字（すなわち年の部分）を取り出して、入力値から開始年を引いて1を足すことにより、その元号における和暦を算出しています。

●76行目

明治～平成のどれにもマッチしなかった（未来の）日付は計算対象外とします。

●77～78行目

sprintf()で文字列を生成し、それをそのまま返しています。年が1の場合は「元年」となるようにしています。

4.5 メインルーチン

基本的な流れは第2章の**リスト1-17**と変わりません。ここでは履歴を表示するという機能が追加されているので、これについて説明します。

4.5.1 セッションの考え方

このスクリプトは和暦を計算します。その元になる年月日は毎回ユーザが入力します。では、「履歴の表示」をするための履歴データは、いったいどこから持ってくればよいのでしょうか？

計算結果は毎回ブラウザの画面に表示されますが、これはクライアントのメモリ（あるいはキャッシュファイル）上にあるものであり、そのままではサーバ側で利用することはできません。これらを毎回履歴としてサーバ側で利用するためには、これらのデータをどこかに保存しておく必要があります。

Webページへのアクセスは毎回完結した処理です。しかし上記のように、以前に使われた情報を何らかの方法で記憶することで一連の処理を関連づけてやれば、見かけ上連続した処理を実現することができます。このように、論理的に連続した一連の処理のことをセッションといい、これを実現することを「セッション管理」と呼びます。PHPではこのセッション管理機能を標準で持っており、比較的簡単に使うことができます。

4.5.2 PHPのセッション管理機能を使う

PHPにおけるセッション管理の実現方法は複雑ですが、使い方はいたって簡単です。ではtest9.phpのメインルーチンを見てください。

●83行目

session_start()で、PHPに対してセッションを開始することを宣言しています。セッション管理特有のおまじないは、これで終わりです。/etc/php.iniで**session.auto_start=1**と設定されている場合は、スクリプトの開始時にセッションが自動的に開始されるため、83行目さえも不要です。あとは**$_SESSION['要素名']**という変数に値を代入するだけで、これがセッション変数として保持されます。

●85行目

入力画面部分の表示ルーチンを呼んでいます。

●86～91行目

「履歴のクリア」ボタンが押されたら履歴をクリアします。履歴は $_SESSION['hist'] に入っています。クリアボタンが押されると、$_POST['clear'] にVALUEで指定（15行目）した値が入ります。unsetは変数を削除するステートメントです。クリアしたら、あとは何もする必要がないので終了しています（厳密には、HTMLの開始や終了タグをつけなければなりませんが、ここでは手抜きしています）。

クリアが押されたらいきなり履歴を消してしまってもよいのですが、もし消すべき履歴がまだない（$_SESSION['hist'] が存在しない）場合は、設定によっては[*16]、unset()で未定義変数への参照という警告が出てしまいます。これを防ぐために、前もって87行目で削除対象の変数が存在するかどうかを判断しています。empty()は変数が空（未定義）かどうかを評価する関数ですが、評価対象の変数が存在しなくても警告が出ないような仕様になっています。

●92～100行目

入力された日付をチェックし、妥当であれば和暦に変換します。無効な入力であればエラーメッセージを表示しています。

●101～105行目

和暦に変換した値があれば、セッション変数に登録します。一時変数 $histにいったんセッション変数を代入しているのは、やはり次の103行目で警告が出るのを避けるためです。$_SESSION['hist'] に代入した最新履歴を、HTMLの表で囲んで表示します。

リスト1-73の説明は、これにてひとまず終了です。あとはPHPマニュアルを探検していただいて、誌面の都合で紹介できなかったいろいろなテクニックを習得してください。これ以降は、Webコンピューティングを行なうにあたり一般知識として知っておいていただきたい、バックグラウンドの技術について紹介していきます。

*16
php.ini でerror_reporting=E_ALLにしている場合。この設定は、不要なバグの混入を防ぐためにお勧めできます。なお、phpスクリプトでは式の前に@を置くことでエラー表示を抑止することができますが、潜在バグの温床となりかねないのであまりお勧めできません。

4.6 セッション管理手法のいろいろ

ここでは一般的なセッション管理手法について説明します。PHPのセッション管理機構を使う分にはブラックボックスとなっている部分ですが、前提知識として知っておいたほうがよいでしょう。

Webシステムで使用されているHTTPプロトコル（次章で解説）には、そもそもセッションの概念がありません。このため、Webを利用して一連の処理を行なう場合、プログラマがアプリケーションレベルでセッション管理のしくみを入れてやる必要があります。その手法としては、以下の三つが考えられます。

①HTTPクッキーを使う。
②URLの引数として指定する。
③hidden属性を使って持ち回る。

いずれの手法においても、セッションの情報（もしくは、サーバがセッション情報を取り出すためのキー）は、クライアントが持ち回ることになります。実際問題として、クライアントの種類をある程度絞り込むことができれば①、そうでなければ用途に応じて②と③を、臨機応変に使い分けるというところが定石となります。これらの手法は混在して使用することもできますし、実際、PHPのセッション管理はこれら3つを併用して実現されています。ではひとつずつ見ていきましょう。

4.6.1 HTTPクッキーを使う

変数の値をクライアント側にファイルとして保持させる方法です。クッキーは（PHPではsetcookie()関数を使って）HTTPヘッダとして送られるため、デバッグ文を含むすべてのデータを出力する前に、クッキーで設定するべきデータを確定しておく必要があります。なぜなら、実データを送ったあとでHTTPヘッダを送るのは、HTTPプロトコル上許されないからです。これについては、PHPで標準で用意されている出力のバッファリング機能が助けになるでしょう。そのほかのCGI言語などでは、内部変数に出力を溜め込んでおいて、プログラムの最後に（クッキーを含む）HTTPヘッダを出力後、バッファの内容を出力するなどの工夫が必要です。

なお、クッキーはブラウザによってはサポートされていなかったり、ブラウザの設定で無効にすることもできるので、使用できるクライアント環境が限定される場合があります。

4.6.2 URLの引数として指定する

たとえばhttp://host/a.php?ID=hoge&PASSWD=secret&PAGE=1などと、セッション情報をアンカーのURLに埋め込んでしまう方法です。

この方法はお手軽ですが、URLが長くなって見づらく、またURLの長さの制限に引っかかりそうで、あまりにも長い変数を覚えさせるにはやや不安があります。しかしクライアントの実装に依存しない方法なので、これもよく使われています。この方式を使ったコンテンツは、URLを保存しておくことによって後日、再利用（再度のコンテンツ呼び出し）ができます。逆にいえば、悪意の第三者にURL文字列が漏洩すると、なりすまし（セッションハイジャック）が行なわれる恐れもあります。

4.6.3 hidden属性を使う

フォームの<INPUT TYPE=hidden …>を使って変数を持ちまわる方法です。これもブラウザに依存しませんし、URLは変わらないので見た目はすっきりします。表示されたコンテンツ自体に情報が埋め込まれているため、いったんブラウザを終了してしまえば、ブラウザで「名前をつけて保存」でもしておかないかぎり、それらの変数を復元することはできません。

4.7 PHPのセッション管理の実現方法

php.iniのセッション関連のオプションは多数ありますが、代表的なものについてのみ説明しておきます。ここではphp.iniが**図1-23**のようになっているとします。これらの中には、.を _ に変えたものを関数名として、値を動的に値を変更できるものもあります。

図1-23 php.iniにおけるセッション関連のオプション（一部）

```
session.save_handler = files     … ①
session.save_path = /tmp         … ②
session.use_cookies = 1          … ③
session.name = PHPSESSID         … ④
session.auto_start = 0           … ⑤
session.cookie_lifetime = 0      … ⑥
session.gc_probability = 1       … ⑦
session.gc_maxlifetime = 1440    … ⑧
session.cache_limiter = nocache … ⑨
session.cache_expire = 180       … ⑩
```

PHPでは、セッションをセッション名とセッションIDというペアで管理します。セッション名はセッション情報を保持するためのキーとなる文字列です。その名前は④で決まり、デフォルトではPHPSESSIDとなっています。

セッションIDは、セッションが新たに開始されるごとにPHPによって動的に割り当てられる、セッション情報の識別子です。セッション維持の手法は①で定義され、デフォルトはファイルに保存されます。このファイルは②で指定されたディレクトリに置かれます。ちょっと実験してみましょう。**図1-24**をご覧ください。w3mでHTTPヘッダ（後述）を表示しています。

図1-24 初回のリクエスト送信に対するHTTPヘッダ

```
hotta@star ~$ ls /tmp
hotta@star ~$ w3m -dump_head http://localhost/~hotta/1-4/test9.php
Received cookie: PHPSESSID=a037816df06324210f0d414e430c34e4
HTTP/1.1 200 OK
Date: Fri, 21 Jun 2002 13:17:07 GMT
Server: Apache/1.3.26 (Unix)  (Vine/Linux) mod_ssl/2.8.9 OpenSSL/0.9.6b PHP/4.2.2
X-Powered-By: PHP/4.2.2
Set-Cookie: PHPSESSID=a037816df06324210f0d414e430c34e4; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Connection: close
Content-Type: text/html

hotta@star ~$ ls /tmp
sess_a037816df06324210f0d414e430c34e4
```

これでわかるように、セッション機能が有効となっているコンテンツにアクセスすると、サーバは②のディレクトリに**sess_セッションID**という名前のファイルを生成し、この中にセッション変数（$_SESSION）の各要素の名前とその値を保存します。そのあとサーバはSet-Cookie: というHTTPヘッダにより**セッションキー＝セッションID**を表す文字列を送っています。

このコンテンツにはフォーム（**図1-25**）が含まれており、ユーザはこれに対し何らかの入力を行なって、<ACTION>タグで示すコンテンツを取得するためのリクエストを送信します。この際ブラウザは、このフォームの中に、直前に受信したセッションIDを PHPSESSIDという変数名で挿入します。

サーバはこの変数の値からファイル名を生成し、②のディレクトリからセッション情報（前回の変数の値）を取得します。③が指定されていない場合は、PHPはHTTPクッキーの代わりに前述の隠しフィールドでセッションIDを通知します。初回のリクエスト時には、クライアントがクッキーを使用できる場合でもサーバにそれがわからないので、サーバはクッキーを送ると同時に、隠しフィールドを生成して返します（**図4-5**の3行目のhidden）。

図1-25 初回のリクエスト送信に対するコンテンツ

```
hotta@star ~$ w3m -dump_source http://localhost/~hotta/1-4/test9.php
Received cookie: PHPSESSID=16ef3b34d89dc5b6427804c56f5759c5
<FORM METHOD="POST" ACTION="/~hotta/1-4/test9.php"><input type="hidden"
name="PHPSESSID" value="16ef3b34d89dc5b6427804c56f5759c5" />
西暦
<INPUT TYPE="text" NAME="yyyy" SIZE="5" VALUE="">年
<INPUT TYPE="text" NAME="mm" SIZE="3" VALUE="">月
<INPUT TYPE="text" NAME="dd" SIZE="3" VALUE="">日
<INPUT TYPE="submit" NAME="submit" VALUE="送信">
<INPUT TYPE="submit" NAME="clear" VALUE="履歴のクリア">
```

⑤がセットしてあれば、session_start()を書かなくてもすべてのPHPコンテンツについて自動的にセッション処理が開始されます。ほとんどのページでセッションを使う場合は、指定しておいてもよいでしょう。

⑥の指定により、ブラウザを終了してもセッションを継続するようにもできます。図4-3の指定ではブラウザをいったん終了するとセッションは破棄されます。さて、破棄されたセッションの実体(情報を持つファイル)はもはやゴミなので、いつかは消さなくてはなりません。これを制御するのが⑦と⑧です。**図1-23**の例では、ゴミになったセッションファイルは、その1440秒後以降にリクエストがあった際に、1%の確率で消去されます。

送信した入力内容をブラウザにキャッシュ(覚え)させて「戻る」ボタンで呼び出せるようにしたい場合は⑨を変更します。**図1-23**の指定では、いっさいキャッシュされません。キャッシュさせるようにした場合、⑩のキャッシュの有効期限が意味を持ちます。詳細はsession_cache_limiter()関数を参照してください。

Hypertext Preprocessor

Chapter-5 PHPを支える技術

この章では、PHPを使ってより実用的なアプリケーションを作成する際に、知っておくべき事柄について説明します。さらに第2部への足がかりとして、PHPとデータベースの連携に関する概念について説明します。

5.1 フォーム

フォームとは、ブラウザからWebサーバへデータを送るための手段を提供するためのHTMLのタグです。フォームに関連する構文を以下に示します。なお、[…]は省略可能を、{…¦…}はいずれか選択することを示します。ブラウザによってはサポートされていないタグがあるかもしれません。タグをどのように扱うかは すべてブラウザ次第です。各ブラウザは、自分に理解できないタグは単に無視します。

各コンポーネントについて順に説明します。実行結果はブラウザによって表示が異なる場合があります。以下のスナップショットではMozilla 1.0を使っています。

5.1.1 フォームの開始

```
<FORM [METHOD=post|get]
      ACTION={"URL"|"mailto:メールアドレス"}>
```

このタグから次に見つかった</FORM>までがひとつのフォーム（一括入力の単位）としてブラウザに認識されます。ブラウザからの入力が行なわれると、このフォームで入力された値が**変数＝値**のリストとしてサーバ側に送信されます。

METHODは変数をサーバに渡すための方法を指定します。省略するとgetとなります。getは変数リストをURLの末尾に**?変数名1＝値1&変数名2＝値2&…**の形式で追加する方式です。postは変数リストを標準入力経由で渡す方式です。

ACTIONは変数リストをどこに渡すのかを指定します。通常はURLとしてCGIプログラム名やPHPスクリプト名を指定します。**mailto:メールアドレス**を指定すると、クライアント側のメーラ（メール送受信用ソフト）が起動します。mailto: によるメール送信はクライアントの機能として行なわれるので、Webサーバ側は関与しません。

図1-26 フォーム関連構文

```
<FORM [METHOD={POST|GET}] ACTION={"URL"|"mailto:メールアドレス"}>
<!-- 単一行テキスト-->
<INPUT TYPE={TEXT|PASSWORD} NAME=変数名 [SIZE=入力欄の文字数]
[MAXLENGTH=入力可能最大文字数] [VALUE="デフォルト文字列"] >
<!-- 複数行テキスト-->
<TEXTAREA NAME=変数名 [ROWS=行数] [COLS=文字数]
[WRAP={hard(physical)|soft(virtual)|off}]>
<!-- ラジオボタン-->
<INPUT TYPE=radio NAME=変数名 VALUE=文字列 [CHECKED]>
<!-- チェックボックス-->
<INPUT TYPE=checkbox NAME=変数名 [VALUE=文字列] [CHECKED]>
<!-- 隠しフィールド-->
<INPUT TYPE=hidden NAME=変数名 VALUE=文字列>
<!-- プルダウンメニュー-->
<SELECT NAME=変数名>
<OPTION VALUE=識別1 [SELECTED]>候補文字列1
....
<OPTION VALUE=識別n [SELECTED]>候補文字列n
</SELECT>
<!-- リストボックス-->
<SELECT SIZE=行数 NAME=変数名>
<OPTION VALUE=識別1 [SELECTED]>候補文字列1
....
<OPTION VALUE=識別n [SELECTED]>候補文字列n
</SELECT>
<!-- リセットボタン-->
<INPUT TYPE=reset [VALUE=ボタン名]>
<!-- 送信ボタン-->
<INPUT TYPE=submit [NAME=変数名] [VALUE=ボタン名]>
</FORM>
```

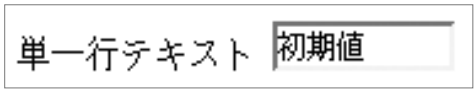
5.1.2 単一行テキスト

```
<INPUT TYPE={text|password} NAME=変数名
       [SIZE=入力欄の文字数]
       [MAXLENGTH=入力可能最大文字数]
       [VALUE="デフォルト文字列"] >
```

単一行フィールド（**図1-27**）で文字列を入力します。TYPEは通常textを指定します。passwordを指定すると、入力した文字列は画面にエコーバックされず、アスタリスク（*****）で表示されます。SIZEで入力欄の画面上の桁数を指定しますが、SIZE桁数を超えた入力を行なうと自動的に横スクロールします。入力する桁数を制限したい場合はMAXLENGTHを指定します。VALUEはデフォルトで入力フィールドに表示しておきたい文字列です。

```
<FORM>
単一行テキスト
<INPUT TYPE=text NAME=field1 SIZE=10 VALUE="初期値">
</FORM>
```

図1-27 実行結果



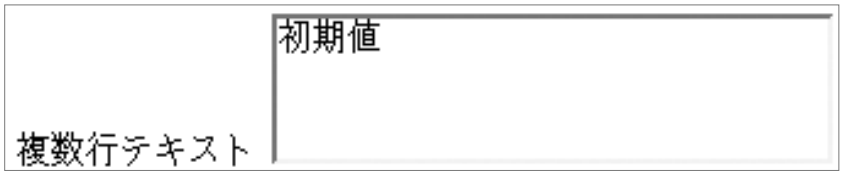
5.1.3 複数行テキスト

```
<TEXTAREA    NAME=変数名
     [ROWS=行数]
     [COLS=文字数]
     [WRAP={hard(physical)|soft(virtual)|off}]>
```

ボックス型のフィールド(図1-28)を表示し、複数行の入力を受けつけます。ボックスの縦横の長さはそれぞれROWSとCOLSで指定します。入力中の行は、画面上ではユーザが明示的に改行しないかぎりは改行されませんが、WRAP指定を行なうと、入力フィールドの大きさに合わせて自動改行を行ないます。hardまたはphysicalを指定すると改行がサーバへの送信データにも反映されますが、softまたはvirtualでは改行は見かけだけのものになり、送信データへは影響を与えません。offは省略時のデフォルトで、自動改行を行ないません。

```
<form>
複数行テキスト
<TEXTAREA NAME=field2 ROWS=3 COLS=30>
初期値
</textarea>
</form>
```

図1-28 実行結果



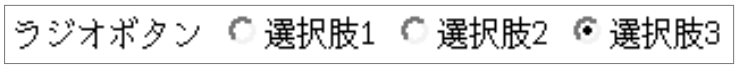
5.1.4 ラジオボタン

```
<INPUT TYPE=radio NAME=変数名
VALUE="送信する文字列" [CHECKED]>表示する文字列
```

選択可能なラジオボタン（図1-29）を表示します。典型的な使い方としては、同一変数名で複数のradio属性を指定することにより複数のボタンを表示して、どれかひとつを選択させることです。チェックされたボタンに設定されたVALUEで指定された文字列が、その変数の値としてサーバに送られます。CHECKED属性をつけたものがデフォルトでチェックされて表示されます。

```
<FORM>
ラジオボタン
<INPUT TYPE=radio NAME=radio1 VALUE="VAL1">選択肢1
<INPUT TYPE=radio NAME=radio1 VALUE="VAL2">選択肢2
<INPUT TYPE=radio NAME=radio1 VALUE="VAL3" CHECKED>選択肢3
</FORM>
```

図1-29 実行結果



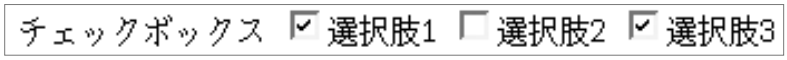
5.1.5 チェックボックス

```
<INPUT TYPE=checkbox NAME=変数名 [VALUE=文字列] [CHECKED]>
```

選択可能なチェックボックス（図1-30）を表示します。ラジオボタンと異なるのは複数選択ができることです。複数の選択を可能にするためには、各エントリに異なった変数名をつける必要があります。今度は変数名でフィールドを識別できるので、VALUEはあまり意味がなくなります。CHECKEDは複数指定することができます。

```
<FORM>
チェックボックス
<INPUT TYPE=checkbox NAME=check[] VALUE="1" CHECKED>選択肢1
<INPUT TYPE=checkbox NAME=check[] VALUE="2">選択肢2
<INPUT TYPE=checkbox NAME=check[] VALUE="3"  CHECKED>選択肢3
</FORM>
```

図1-30 実行結果



PHPの文法における特徴で、**$変数名[]**という式に対して代入を行なうと、自動的に配列に要素が追加されます。このしくみを利用して、checkboxの変数名を配列にすることができます。ひとつのチェックボックスのすべての要素に同じ**NAME=変数名[]**という名前をつけ、VALUEに異なった値を指定しておくことにより、大量の選択肢がある場合でも効率的な処理を行なうことができます。

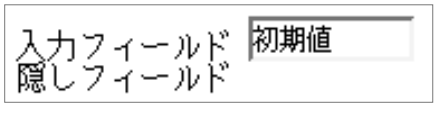
5.1.6 隠しフィールド

```
<INPUT TYPE=hidden NAME=変数名 [VALUE=文字列]>
```

変数＝値のペアをサーバに返します。このフィールドはユーザの目に触れることはなく、また変更されることもありません。プログラマが次のURLに何らかの値を渡したい場合に使用します。PHPのセッション機能が間接的に使っています。

```
<FORM>
入力フィールド
<INPUT TYPE=text NAME=field1 SIZE=10 VALUE="初期値"><BR>
隠しフィールド
<INPUT TYPE=hidden NAME=field2 VALUE="秘密の値">
</FORM>
```

図1-31 実行結果



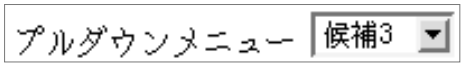
5.1.7 プルダウンメニュー

```
<SELECT NAME=変数名>
<OPTION VALUE=識別1 [SELECTED]>候補文字列1
……
<OPTION VALUE=識別n [SELECTED]>候補文字列n
</SELECT>
```

プルダウンメニュー（**図1-32**）を表示します。選択された候補のVALUEがその変数の値となります。デフォルト値を設定するにはSELECTEDオプションを使います。

```
<FORM>
プルダウンメニュー
<SELECT NAME=pmenu>
<OPTION VALUE=SEL1>候補1
<OPTION VALUE=SEL2>候補2
<OPTION VALUE=SEL3 SELECTED>候補3
</SELECT>
</FORM>
```

図1-32 実行結果



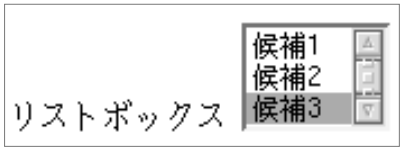
5.1.8 リストボックス

```
<SELECT SIZE=要素数(行数) NAME=変数名>
       <OPTION VALUE=識別1 [SELECTED]>候補文字列1
       ....
       <OPTION VALUE=識別n [SELECTED]>候補文字列n
       </SELECT>
```

リストボックス（図1-33）を表示します。選択された候補のVALUEがその変数の値となります。デフォルト値を設定するにはSELECTEDオプションを使います。

```
<FORM>
リストボックス
<SELECT SIZE=3 NAME=lbox>
<OPTION VALUE=SEL1>候補1
<OPTION VALUE=SEL2>候補2
<OPTION VALUE=SEL3 SELECTED>候補3
</SELECT>
</FORM>
```

図1-33 実行結果



5.1.9 リセットボタン

```
<INPUT TYPE=reset [VALUE=ボタン名]>
```

リセットボタン（図1-34）を表示します。リセットボタンはそのフォームに入力中のデータをすべて破棄（クリア）するためのもので、サーバの動作には影響を与えません。VALUEを指定しない場合のボタンの表示は、ブラウザによって異なります。

```
<FORM>
リセットボタン
<INPUT TYPE=reset>
</FORM>
```

図1-34 実行結果

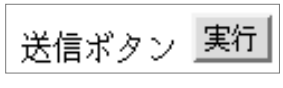
5.1.10 送信ボタン

```
<INPUT TYPE=submit [NAME=変数名] [VALUE=ボタン名]>
```

送信ボタン（図1-35）を表示します。送信ボタンはそのフォームに入力されたデータをまとめてサーバ側に送信するものです。ボタンを複数配置する場合は、どのボタンが押されたのかをNAMEにより識別します。VALUEを指定しない場合のボタンの表示は、ブラウザによって異なります。

```
<FORM>
送信ボタン
<INPUT TYPE=submit>
</FORM>
```

図1-35 実行結果



5.1.11 ファイルのアップロード

```
<FORM ENCTYPE="multipart/form-data" ACTION="URL" METHOD=post>
<INPUT TYPE=hidden NAME=MAX_FILE_SIZE VALUE=受信可能バイト数>
送信ファイル名
<INPUT TYPE=file NAME=変数名 [SIZE=入力フィールド文字数]>
</FORM>
```

このタグの組み合わせによりファイルのアップロードを行ないます。URLはアップロードされたファイルを処理するためのPHPスクリプト名です。隠しフィールドMAX_FILE_SIZEは、サーバが受信することができる最大のファイルサイズのバイト数です。この宣言は、ファイル名入力フィールドの前に置く必要があります。VALUEを超えるサイズのファイルをアップロードしようとすると、PHPに拒否されます。

<INPUT>タグのfileタイプにおけるNAMEオプションは、アップロードされたファイルを識別するための変数名を指定します。この名前は、アップロードされるファイルの（クライアント側における）オリジナルのファイル名とはまったく関係がありません。アップロードに成功すると、PHPスクリプト内部で以下の変数が参照できるようになります。「変数名」はINPUT=fileのNAME属性で指定した名前です。

●$_FILES['変数名']['tmp_name']

アップロードされたファイルの一時的な名前。サーバマシンにアップロードされたファイルは、一時的にサーバ内部の一時ディレクトリに保存されますが、この変数にはそのパス名が格納されます。

このファイルは、スクリプトの実行終了時にPHPによって自動的に削除されてしまうので、スクリプト内部で適切に移動やコピーなどの処理を行なってやる必要があります。一時ファイルを保存するディレクトリ名は、環境変数 TMPDIR の設定およびphp.iniのupload_tmp_dirオプションによって変更できます。

●$_FILES['変数名']['name']

クライアントマシンにおける、ファイルの元の名前。クライアントがWindowsの場合は日本語ファイル名はShift-JISコードで入ってくるので、サーバ上に保存する際は適切に変換してやらなければならない場合があります。

●$_FILES['変数名']['size']

アップロードされたファイルのサイズ（バイト数）。

●$_FILES['変数名']['type']

この情報がブラウザで指定された場合、ファイルのMIME型（例："image/gif"）。ただしブラウザはこの情報を提供するとはかぎりませんし、ブラウザにより返す値もまちまちのようです。

●$_FILES['変数名']['error']

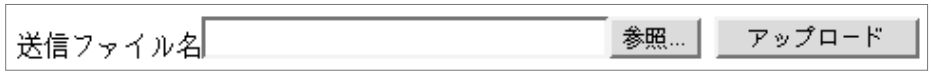
アップロードに失敗した場合のエラー文字列。

以下のスクリプトは、ファイルのアップロード用のフォームを表示します。

リスト1-75 1-29.php

```
<?php
$path="/tmp/test.out";                          //  保存ファイル名
if (!empty($_FILES['uploaded']['name']))    {
    echo <<<EOD
¥$_FILES['uploaded']['tmp_name']={$_FILES['uploaded']['tmp_name']}<BR>
¥$_FILES['uploaded']['name']={$_FILES['uploaded']['name']}<BR>
¥$_FILES['uploaded']['size']={$_FILES['uploaded']['size']}<BR>
¥$_FILES['uploaded']['type']={$_FILES['uploaded']['type']}<BR>
EOD;
    if (move_uploaded_file(
        $_FILES['uploaded']['tmp_name'], $path) == FALSE)  {
        printf("ファイルの移動に失敗しました：%s<BR>¥n",
            $_FILES['uploaded']['error']);
    }
    system("/bin/ls -l /tmp/test*");
} else  {
    echo <<<EOD
<FORM ENCTYPE="multipart/form-data"
ACTION="{$_SERVER['PHP_SELF']}" METHOD=post>
<INPUT TYPE=hidden name=MAX_FILE_SIZE value=10000000>
送信ファイル名<INPUT NAME=uploaded TYPE=file SIZE=30>
<INPUT TYPE=submit VALUE=アップロード>
</form>
EOD;
}
?>
```

図1-36 実行結果



リスト1-29を呼び出すと、まず図1-36の画面になります。「参照」ボタンを押すと、クライアントブラウザ側で用意されているファイル選択ダイアログが表示されます。図1-37はブラウザのダイアログでファイルを選択したところです。選択したファイルは「アップロード」ボタンを押すことによりサーバ側に送信されます。

図1-37 ファイル選択ダイアログ

図1-38 実行結果

```
$_FILES['uploaded']['tmp_name']=/tmp/phppCYShh
$_FILES['uploaded']['name']=sss.tar.gz
$_FILES['uploaded']['size']=122132
$_FILES['uploaded']['type']=application/x-gzip
-rw------- 1 apache apache 122132 Jun 22 09:48 /tmp/test.out
```

move_uploaded_file()関数により、一時的なファイルをアプリケーションで決めたファイル名にリネームして保存しています。正しく保存されたかどうかを/bin/lsコマンドで確認しています。

5.2 URL

これまでにも何度か出てきていますが、URLとはネットワークリソースのアドレスのことです。Webにかぎっていえば、特定のコンテンツの場所を示すための表記方法のことを指します。URLは以下のパーツに分解することができます。

＜プロトコル名＞:＜プロトコル固有の形式＞

URLで使用されるプロトコル名のうちよく使われるものを**表1-32**に示します。

表1-9　URLで使用するプロトコル

プロトコル名	説明
http://	HTTP（Hyper Text Transfer Protocol）
https://	SSL上で暗号化されたHTTP
ftp://	FTP（File Transfer Protocol）
file://	ホスト固有のファイル名

HTTPなど大文字で書かれているものはプロトコルなどの略語であることを表し、httpなど小文字で書かれているものは、ブラウザでURLを入力する際にキーインするべき文字列を表しています。

プロトコル固有の形式は、プロトコルによって異なります。HTTPにおけるURLでは以下の形式が使われています。**[...]** は省略可能を意味します。

```
[[http://]server[:port]/][dir/]file[.ext][?var=val[&var=val]...]
```

server …… サーバ名
port …… ポート番号
dir …… ディレクトリ名
file …… ファイル名
ext …… 拡張子
var …… 変数名
val …… 値

URLに**?変数名＝値**オプションをつけてURLを呼び出した場合、それはGETメソッドによるパラメータ渡しとなります。渡された変数は、PHPスクリプト側で**$_GET['変数名']**とすることにより参照できます。**?**以降の部分のことを「検索文字列」とか「クエリコンポーネント」などと呼ぶこともあります。

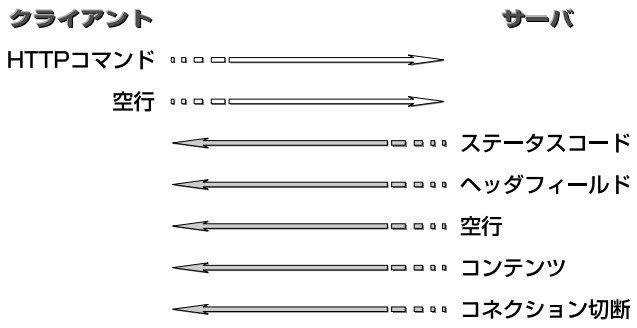
5.3 HTTP

クライアント（通常はブラウザ）とWebサーバ間で情報をやりとりする際には、HTTPというプロトコルが使用されます。HTTPを理解すると、いろいろな応用が利くようになります。

5.3.1 HTTPの流れ

図1-33にHTTPの流れを示します。まずクライアントは一個以上のHTTPのコマンドおよびひとつの空行をサーバへ送ります。サーバはそれに対してステータスコードと呼ばれる行を返し、それに続いてヘッダフィールドと呼ばれる複数の行を返します。そして空行を送ることによりメタ情報（管理情報）の終わりを示し、それに続いて実際のコンテンツを返します。

図1-39 HTTPの流れ



5.3.2 HTTPコマンド

もっともよく使われるコマンドはGETコマンドです。これはクライアントがサーバに対して特定のコンテンツを要求するためのもので、文法は次のような形式です。

```
GET <URL> HTTP/バージョン番号
```

GETコマンドを使ってCGIなどに引数を渡す場合、URLのあとに**?変数名=値**というクエリコンポーネントをつけて渡します。

HTTPのコマンドのうちWebサーバの処理系で実装する必要最小限のものは、GETのほかにはHEADとPOSTだけです。HEADはファイルのメタ情報（最終更新時刻など）だけを要求し、コンテンツ本体を要求しないコマンドです。POSTはクライアントからサーバにデータを転送するために使用するコマンドです。フォームにおけるmethodのタイプとは意味が異なるので注意してください。

5.3.3 ステータスコード

ステータスコードはコマンドの実行結果を表すものです。このコードとそれに続くヘッダフィールドを合わせて、コンテンツのメタ情報（メッセージヘッダ）と呼んでいます。ステータスコードは**表1-10**に示すカテゴリに分類されます。

表1-11に主なステータスコードを示します。これらをうまく利用すれば、コンテンツの移動を知らせたり認証を行なったりと、通常のHTMLだけでは不可能な、さまざまなことができるようになります。

表1-10 ステータスコードのカテゴリ

カテゴリ	コード範囲	説明
情報	100～199	アプリケーション固有メッセージ
成功	200～299	要求が正常に処理された。 クライアントは要求を処理するために、さらにアクションを起こす必要がある
リダイレクト	300～399	これに対しては、エンドユーザは意識せず、クライアントソフトウェアが自動的に実行することが多い
クライアントエラー	400～499	クライアント側の問題
サーバエラー	500～599	サーバ側の問題

表1-11 主なステータスコード

ステータスコード	説　明
200 OK	エラーなし。要求は正常に処理された
201 Created	POST要求が正常に実行された
204 No Content	要求は正常に処理されたが、クライアントに返すべきデータはない
300 Multiple Choices	要求されたリソースは複数のロケーションから利用できる。サーバの優先選択肢は、応答の中のLocationフィールドに含まれる
301 Moved Permanently	要求されたURLは恒久的に移動された。移動先は応答の中のLocationフィールドで指定する。今後、このリソースへの要求は新しいURLを指定すること
302 Moved Temporarily	要求されたURLは一時的に移動されている。移動先は応答の中のLocationフィールドで指定する。今後、このリソースへの要求は、元のURLを指定すること
304 Not Modified	条件つきGET要求がなされたが、そのコンテンツはIf-Modified-Sinceフィールド内の日付以降、更新されていない
400 Bad Request	要求が認識されなかった
401 Unauthorized	これがanonymous要求である場合には認証が行なわれなければならない。認証済みの要求であった場合には認証が拒否されたことを示す
403 Forbidden	権限不足により要求が実行できない
404 Not Found	URLで指定されたコンテンツが見つからない
500 Internal Server Error	サーバ内部のエラー。cgiが実行できない、など
503 Server Unavailable	一時的にサービスできない状態。通常はサーバの過負荷または保守中を示す

ステータスコードおよび後述するヘッダフィールドについては、PHPにもheaders()という組み込み関数が用意されており、これで自由に作成・送信することができます。「4.8.1 HTTPクッキーを使う」で紹介したSetCookie()関数も、Set-Cookie: というヘッダフィールドを送信する関数です。

5.3.4 ヘッダフィールド

ヘッダフィールドは電子メールのヘッダと同様の**キーワード: 値**という構造を持ち、いくつでも指定することができます。よく使われるヘッダフィールドを**表1-12**に示します。

表1-12 主なヘッダフィールド

ヘッダフィールド	説　明
Content-Length	送信するメッセージ本体のサイズ（バイト数）
Content-Type	メッセージ本体の文書タイプ。HTMLの場合は「text/html」となる
Date	メッセージが発行された日付と時刻
Expires	データの有効期限。 クライアントはこれをキャッシュの保存期限とする
If-Modified-Since	GETコマンドでリソースを要求する際に指定するオプションの日付と時刻。これで「304 Not Modified」ステータスを返してもらうことにより、更新されていないコンテンツのむだな再ロードを防ぐ
Last-Modified	最終更新時刻
Location	自動リダイレクト（ステータス 300～399）の場合に使われる新しいURL
Server	HTTP サーバの名前とバージョン情報
User-Agent	HTTP クライアントの名前とバージョン番号
WWW-Authenticate	BASIC 認証で使用される

5.3.5 HTTPの実際

では、実際にHTTPを通してindex.htmlが呼び出されるところを見てみましょう（**図1-40**）。index.htmlの中身は「TEST」と書いてあるだけです。下線部は入力を表しています。第4章ではHTTPヘッダなどを見るのにw3mを使いましたが、ここでは一番原始的な、telnetで接続してみましょう。説明の都合上、行番号をつけています。

● 1行目

telnetコマンドには、第2引数（またはオプション）としてポート番号を与えています。これで対応するポートをlisten（接続待ち）しているサーバに接続することができます。この例では80番ポート（http）を指定しているので、80番ポートで接続を待っているWebサーバ（httpd）に接続することになります。同様の方法で、SMTPやPOPといったTCPの上位プロトコルの解析を行なうことができます。

● 2～4行目

telnetクライアントが表示している、サーバへの接続状況です。ここでWebサーバに接続され、HTTPコマンドの入力待ちになります。

● 5行目

キーボードからGETコマンドを入力しています。一番目の引数は受信したいファイル名へのパスですが、/は特別で、Webサーバで規定されたドキュメントのルートディレクトリ

(Apacheの設定ファイルであるhttpd.confのDocumentoRootで指定)を表します。また、要求するファイル名をここでは明示的に指定していませんが、この場合はデフォルトのファイル名(httpd.confのDirectoryIndexで指定)であるindex.htmlが選択されます。

●6行目

空行を送って、サーバにコマンドの終了を知らせます。

●7行目

サーバが返したステータスです。

●8～15行目

サーバが返したヘッダフィールドです。

●16行目

メタ情報(ステータス+ヘッダフィールド)の終了を表す空行。

●17行目

コンテンツ本体。

●18行目

コンテンツを送り終え、サーバがコネクションを切断したことを示します。

図1-40 HTTPの実際

```
 1 hotta@star ~$ telnet localhost 80[Enter]
 2 Trying 127.0.0.1...
 3 Connected to localhost.
 4 Escape character is '^]'.
 5 GET / HTTP/1.0[Enter]
 6 [Enter]
 7 HTTP/1.1 200 OK
 8 Date: Sat, 22 Jun 2002 15:02:35 GMT
 9 Server: Apache/1.3.26 (Unix)  (Vine/Linux) mod_ssl/2.8.9 OpenSSL/0.9.6b PHP/4.2.2
10 Last-Modified: Sat, 22 Jun 2002 15:01:06 GMT
11 ETag: "540cf-5-3d149132"
12 Accept-Ranges: bytes
13 Content-Length: 5
14 Connection: close
15 Content-Type: text/html
16
17 TEST
18 Connection closed by foreign host.
```

5.4 CGI — *Common Gateway Interface*

第1章で、CGIの説明として「Webサーバが、URLで指定されたファイル（CGIプログラム）を外部プログラムとして起動する形式」としていましたが、実は元々CGIとはWebサーバと外部プログラム間のインタフェース（接続規約）の名前です。CGIにより、HTTPサーバと非HTTPプログラムの「ゲートウェイ」接続を実現するわけです。しかし、CGIプログラムやCGIスクリプトのことを単にCGIという場合もあります。ここではインタフェースとしてのCGIについて解説します。

PHPもCGI相当の機能を使って実装されていますが、プロトコルがうまく隠蔽されており、通常はその存在を意識することはありません。CGIスクリプトとして最もポピュラーなのはPerlですが、Perl CGIを使用する場合はある程度CGIやHTTPを意識する必要があります。もっとも、PerlではCGIを専用に扱うライブラリなどが非常に充実しているので、慣れればC言語でコーディングするよりはるかに楽です。それでも、PHPはPerlよりもさらに初心者にとってとっつきやすい文法体系になっています。

CGIではコマンド行、パラメータ（環境変数）、入出力などのHTTPサーバとCGIプログラム間のインタフェースを規定します。以下、C言語で書かれたCGIプログラムについて説明します。Perl CGIの場合はPerlの処理系（インタプリタ）内部の話になります。PHPのコマンドライン版をCGIとして使用する場合にも当てはまります。

5.4.1 コマンド行

CGIプログラムをブラウザから起動する場合は、URLとしてCGIプログラム名を指定します。URLの末尾に**?変数名＝値**形式の引数を付加することができます。コマンド行は、標準のargc/argv引数を介してCGIプログラムに渡されます。

5.4.2 環境変数

そのほかのデータは環境変数を介してCGIプログラムに渡されます。CGI環境変数を**表1-13**に示します。

たとえば、HTTPヘッダのUser-Agentを表す環境変数HTTP_USER_AGENTにはブラウザの名前が入るので、CGI内部でそれを見て処理を変えることにより、ブラウザの種類（i-modeであれば携帯電話の型番）やバージョンに依存する細かい動作の違いを吸収するといったテクニックが使えます。PHPでは $_SERVER['USER_AGENT'] のようにして取得できます。

表1-13 CGI環境変数

環境変数	説　明
AUTH_TYPE	HTTP認証メカニズム
CONTENT_LENGTH	HTTPのContent-Lengthヘッダと同じもの
CONTENT_TYPE	HTTPのContent-Typeヘッダと同じもの
GATEWAY_INTERFACE	サーバが準拠するCGI仕様のバージョン。通常はCGI/1.1
HTTP_*	HTTP ヘッダの前に接頭辞HTTP_をつけると、HTTPヘッダを取得できる
PATH_INFO	URLから抽出したCGIプログラムへのパス
PATH_TRANSLATED	サーバOS固有表現に変換されたCGIプログラムへのパス
QUERY_STRING	コード化されたURLのクエリ部分（？以降）
REMOTE_ADDR	クライアントのIPアドレス
REMOTE_HOST	クライアントのFQDN
REMOTE_IDENT	クライアントの名前（利用可能な場合）
REMOTE_USER	クライアントのユーザ名（認証を使用する場合）
REQUEST_METHOD	要求されたHTTPコマンド
SCRIPT_NAME	CGIプログラムの名前
SERVER_NAME	サーバの名前
SERVER_PORT	要求を受信したポート番号
SERVER_PROTOCOL	プロトコル名とバージョン（通常はHTTP/x.x）
SERVER_SOFTWARE	サーバソフトウェアの名前とバージョン

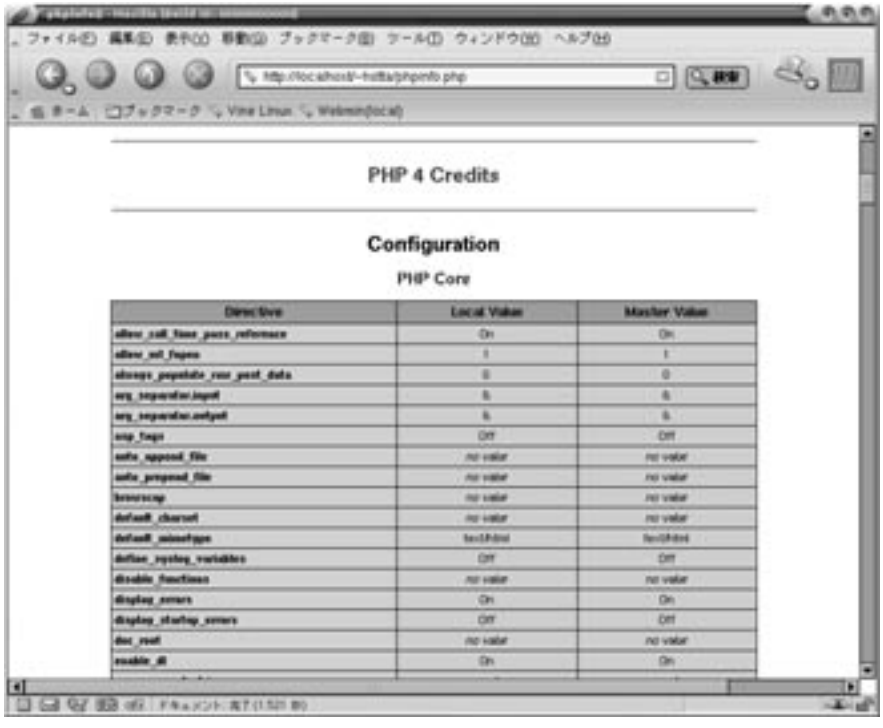
5.4.3 入出力

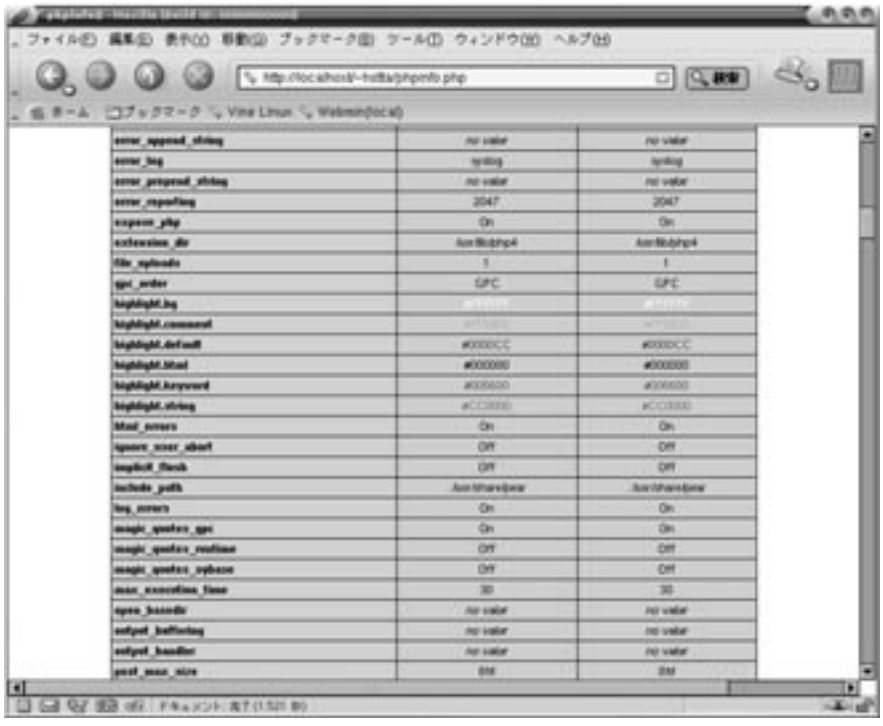
クライアントからの入力は標準入力（stdin）から読み取られます。入力データのサイズは前もって環境変数 CONTENT_LENGTH にセットされています。クライアントへの出力は標準出力（stdout）に書き込むことによって送信されます。

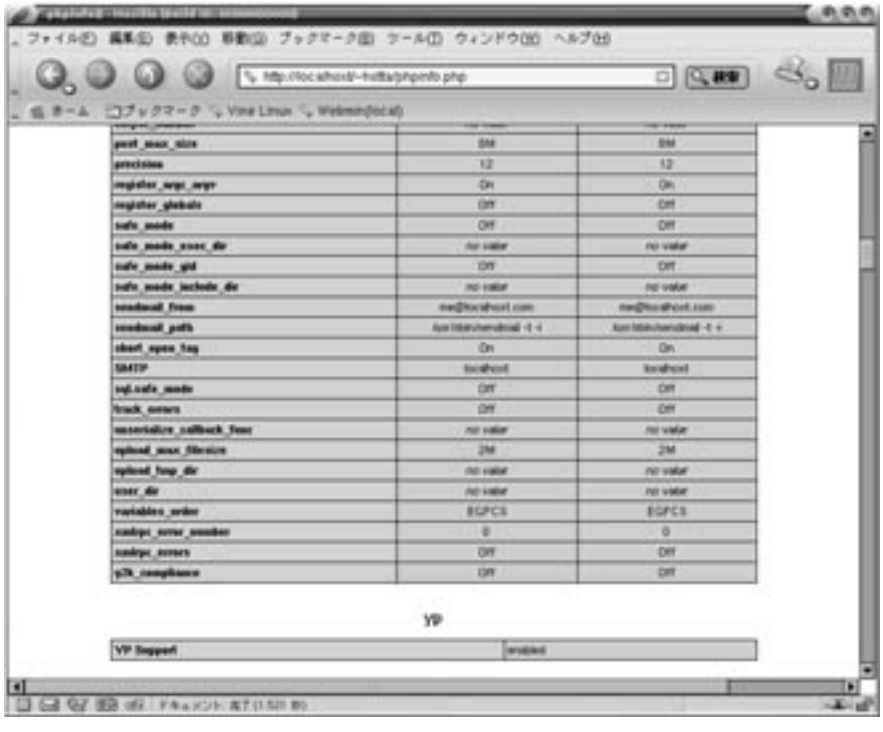
5.4.4 phpinfo()

ここで、組み込み関数phpinfo()を呼び出すだけのスクリプトを実行してみます。これを呼び出すと、**図1-41**のような大量の出力が行なわれます。この中のApache Environmentで各種CGI環境変数を見ることができます。それ以外にも、興味深い情報がたくさん得られることでしょう。

図1-41 phpinfo()の出力







YP

YP Support	enabled

xslt

XSLT support	enabled

xml

XML Support	active
XML Namespace Support	active
EXPAT Version	expat_1.95.2

wddx

WDDX Support	enabled
WDDX Session Serializer	enabled

standard

Regex Library	Bundled library enabled
Dynamic Library Support	enabled
Path to sendmail	/usr/sbin/sendmail -t -i

Directive	Local Value	Master Value
assert.active	1	1
assert.bail	0	0

Dynamic Library Support	enabled
Path to sendmail	/usr/sbin/sendmail -t -i

Directive	Local Value	Master Value
assert.active	1	1
assert.bail	0	0
assert.callback	no value	no value
assert.quiet_eval	0	0
assert.warning	1	1
safe_mode_allowed_env_vars	PHP_	PHP_
safe_mode_protected_env_vars	LD_LIBRARY_PATH	LD_LIBRARY_PATH
url_rewriter.tags	a=href,area=href,frame=src,input=src,form=fakeentry	a=href,area=href,frame=src,input=src,form=fakeentry

sockets

Sockets Support	enabled

session

Session Support	enabled

Directive	Local Value	Master Value
session.auto_start	Off	Off
session.cache_expire	180	180
session.cache_limiter	nocache	nocache
session.cookie_domain	no value	no value
session.cookie_lifetime	0	0


session
posix


posix
pcre
openssl
mbstring

gmp

gmp support	enabled

gettext

GetText Support	enabled

gd

GD Support	enabled
GD Version	1.8.2 or higher
FreeType Support	enabled
FreeType Linkage	with TTF library
JPG Support	enabled
PNG Support	enabled
WBMP Support	enabled

ftp

FTP support	enabled

domxml

DOM/XML	enabled
libxml Version	2.4.13

domxml

DOM/XML	enabled
libxml Version	2.4.13
HTML Support	enabled
XPath Support	enabled
XPointer Support	enabled
DOM/XSLT	enabled
libxslt Version	1.0.10

dba

DBA support	enabled
Supported handlers	gdbm

ctype

ctype functions	enabled (experimental)

bz2

BZip2 Support	Enabled
BZip2 Version	1.0.1, 23-June-2000

zlib

ZLib Support	enabled

zlib

ZLib Support	enabled
'zlib' fopen wrapper	enabled
Compiled Version	1.1.4
Linked Version	1.1.4

Directive	Local Value	Master Value
zlib.output_compression	Off	Off

pgsql

PostgreSQL Support	enabled
Active Persistent Links	0
Active Links	0

Directive	Local Value	Master Value
pgsql.allow_persistent	On	On
pgsql.auto_reset_persistent	Off	Off
pgsql.max_links	Unlimited	Unlimited
pgsql.max_persistent	Unlimited	Unlimited

apache

APACHE_INCLUDE	
APACHE_TARGET	
Apache Version	Apache/1.3.24
Apache Release	10324100

apache

APACHE_INCLUDE	
APACHE_TARGET	
Apache Version	Apache/1.3.24
Apache Release	10324100
Apache API Version	19980320
Hostname:Port	[redacted]
User/Group	[redacted]
Max Requests	Per Child: 100 - Keep Alive: on - Max Per Connection: 100
Timeouts	Connection: 300 - Keep-Alive: 15
Server Root	/etc/httpd
Loaded Modules	mod_ssl, mod_php4, mod_setenvif, mod_so, mod_headers, mod_expires, mod_auth_db, mod_auth_anon, mod_auth, mod_access, mod_rewrite, mod_alias, mod_userdir, mod_actions, mod_imap, mod_asis, mod_cgi, mod_dir, mod_autoindex, mod_include, mod_info, mod_status, mod_negotiation, mod_mime, mod_log_referer, mod_log_agent, mod_log_config, mod_env, mod_vhost_alias, http_core

Directive	Local Value	Master Value
child_terminate	0	0
engine	1	1
last_modified	0	0
shellback	0	0

Apache Environment

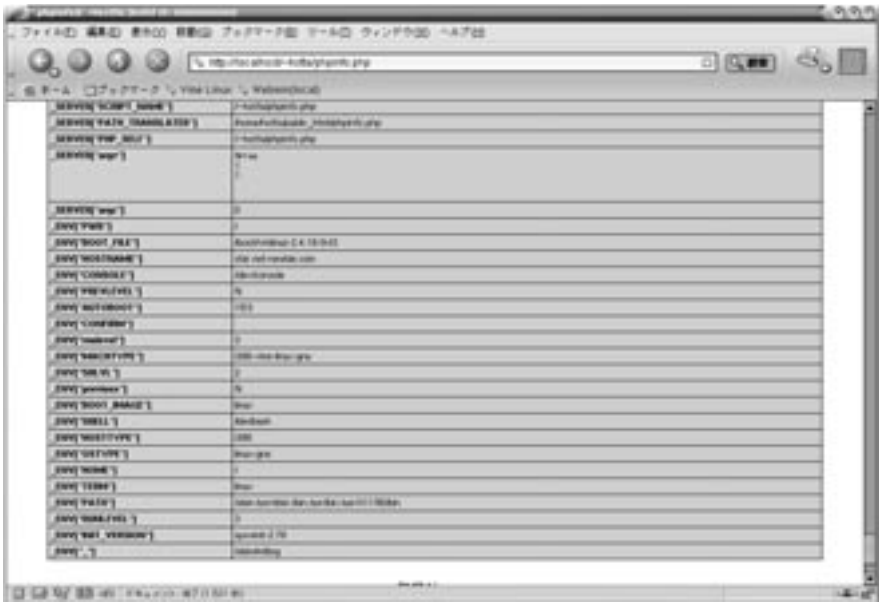
Variable	Value
DOCUMENT_ROOT	[redacted]


Apache Environment


HTTP Headers Information
Additional Modules


Environment
Variable
Value
PHP Variables
Variable
Value


PHP Variables
Variable
Value




PHP License

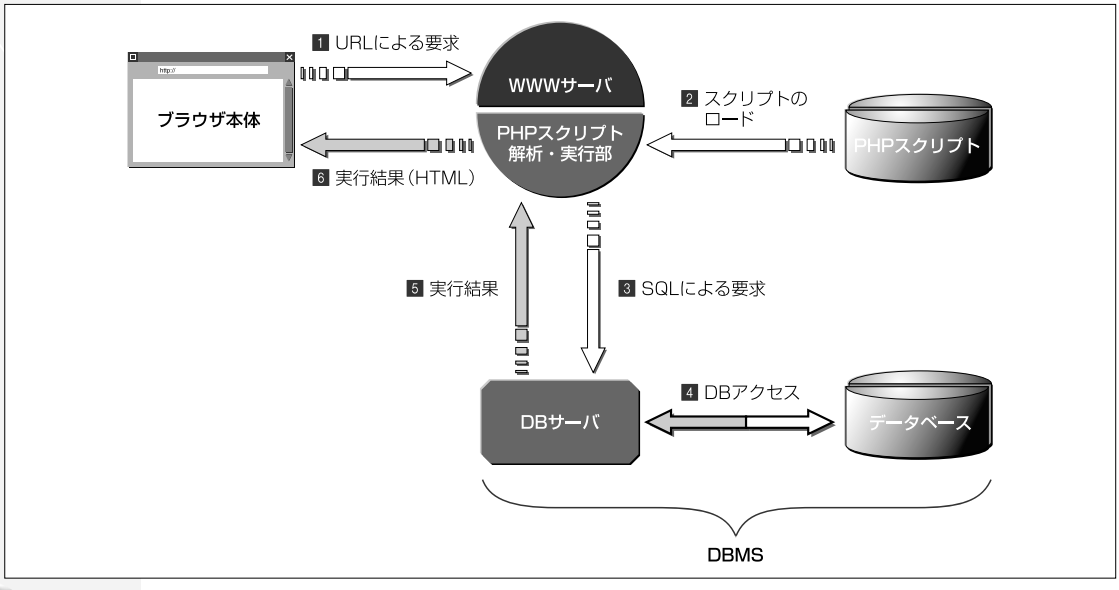
5.5 データベースとの連携

PHPには多くのDBMS（データベース・マネージメント・システム）との連携機能が組み込まれています。現在PHPがサポートするDBMSを以下に示します。

Adabas D	Ingres	Oracle（OCI7およびOCI8）
dBase	InterBase	Ovrimos
Empress	FrontBase	PostgreSQL
FilePro（読み込みのみ）	mSQL	Solid
Hyperwave	Direct MS-SQL	Sybase
IBM DB2	MySQL	Velocis
Informix	ODBC	Unix dbm

DBMSにアクセスするためには、SQL（Structured Query Language ― 構造化問い合わせ言語）を使うのが標準になっています。SQLはデータを定義するためのDDL（Data Definition Language ― データ定義言語）であり、かつデータを操作するためのDML（Data Manipulation Language ― データ操作言語）でもあります。PHP組み込みのDBMSアクセス関数でも、SQL言語を使ったDBMSへのアクセスを記述することができます。**図1-42**にDBMSとの連携についての概念図を示します。ここでDBMSサーバはWWWサーバと同一のマシン上で動作していてもかまいませんし、負荷を分散するために異なったマシン上で動作させていてもかまいません。

図1-42　PHPとDBMSとの連携



本書ではDBMSの代表的なものとしてPostgreSQLを取り上げ、第2部で実践的な解説を行なっています。ただDBMSを使うには、前提とする知識もそれなりに必要です。ここでは初心者のために、第2部を読みこなすための予備知識について解説しておきましょう。

5.5.1 データベースとは

データベースとは、いろいろなデータの固まり、あるいはそれを入れるための入れ物です。ただし、ここでいう狭義のデータベースはリレーショナル（関係）データベース（RDBMS）と呼ばれるもので、表形式でデータを管理します。Excelなどの表計算ソフトを使ったことがある人にはおなじみの、横にいくつかの項目があって、縦には同じ構造を持った複数のデータがマス目の中に並んでいる構造です。

5.5.2 DBMSの一般的な構成

表計算では、マス目のひとつひとつはセルと呼ばれます。表計算ソフトではカーソルが表示され、縦横自由にカーソル（＝操作対象のセル）を動かすことができます。しかし、PostgreSQLをはじめとするDBMSの場合、各データの操作という基本的な部分のみの機能だけを提供しており、実際に取り出したデータを表示するなどのいわゆるユーザインタフェースの部分は外出しにするのが普通です。つまり、実際にディスク上にあるデータを取り出したり更新したりという作業はデータベースサーバ（バックエンド）が行ない、GUIなど人間が見たり入力したりする部分（フロントエンド）は別のプログラムが担当します。たとえばODBC（Open DataBase Connectivity）というミドルウェアを使えば、ExcelがPostgreSQLやOracleといったDBMSのフロントエンドとなることができます。またJavaアプリケーションから接続したい場合はJDBC（Java DataBase Connectivity）というミドルウェアを使います。

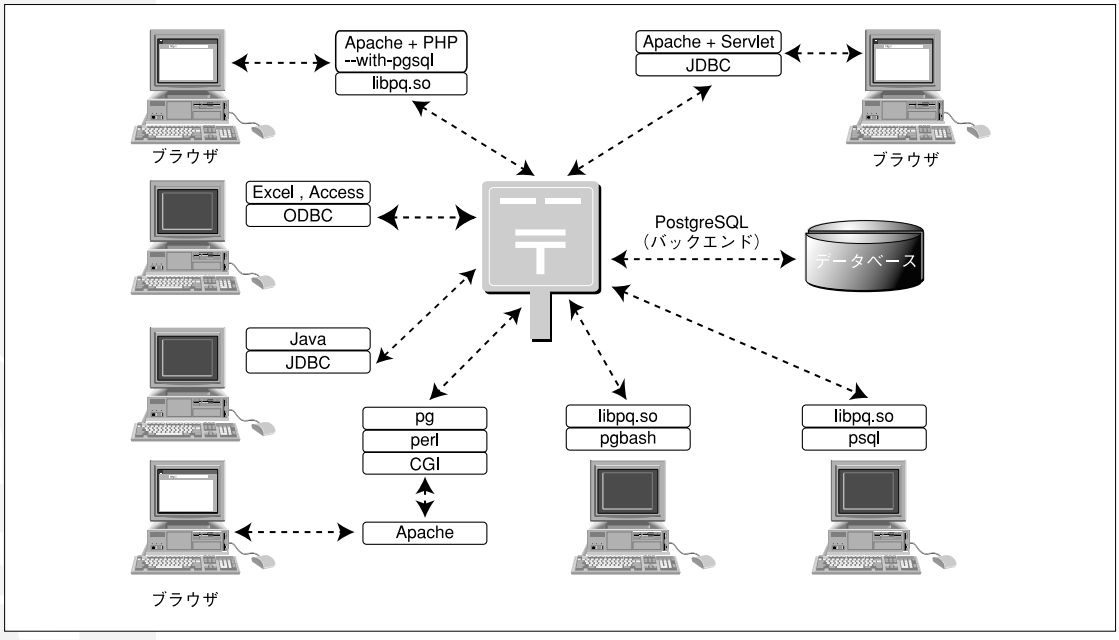
DBMSはクライアント・サーバ構成になっているので、フロントエンド（クライアント）はバックエンドに対してデータを操作することを「依頼する」ことしかできないことに注意してください。PHPとPostgreSQLとを連携させる場合であれば、PostgreSQLは要求がくるのをじっと待っているバックエンド、PHP（Apache）はPostgreSQLに対して要求を出すフロントエンドという役割分担となります。**図1-43**にその概念図を示します。

この図を見てもわかる通り、PHPもPostgreSQLから見ると、数あるクライアントのうちのひとつに過ぎません。PostgreSQLでは各種プログラミング言語からアクセスできるように、いろいろな言語に対するAPI[*17]が規定されています。一般的にデータベースにアクセスするプログラムは、ユーザの目に触れるフロントエンド部と、実際にDBMSに要求を出すAPI部がリンクされたかたちで作成されています。PHPの場合は、libpq.soというライブラリがPostgreSQLに対するAPIの役割を果たします。

PostgreSQLのパッケージにも、psqlというコマンドラインベースの会話型フロントエン

*17
Application Program Interfaceの略。各種言語から特定の機能を呼び出すための規約。複雑なプロトコルや内部仕様を隠蔽（抽象化）し、プログラマが簡単にいろいろな機能を使えるようにするために用意される、プログラムの層。

図1-43 PostgreSQLの各種API



ドプログラムが含まれています。PostgreSQLをインストールすると、psqlのほかにも各種保守用コマンド（シェルスクリプト）がインストールされますが、その実体はpsqlに対して処理を依頼する、psqlのフロントエンドという場合がほとんどです。psqlの場合もAPIとしてlibpq.soを使用しています。

5.5.3 データベースにアクセスする

各クライアントはPostgreSQLサーバに対して各種の要求を出すことができます。データベースに関する要求は、データの追加、修正、削除、呼び出しなど多岐に渡りますが、これらをSQL命令として発行します。

SQLでは、各種のデータ構造を表（テーブル）という概念で表します。ひとつの表は複数の行（レコード、タプル）で構成され、行は複数の列（カラム、フィールド）で構成されます。各項目はそれぞれいろいろな型（タイプ）や長さを持っています。各行のフォーマットはすべて同一ですが、可変長の項目であればその長さは各々異なります。このような表がひとつ以上格納されている、物理的なディスク上の領域をデータベース（あるいはデータベースクラスタ）と呼びます。

SQLの代表的な文法を以下に示します。詳細は第2部に説明があるので、ここでは概念的な必要最小限の説明にとどめます。まずdbtestというデータベースを生成し、その中にidとnameという二つの項目を持つテーブルtbltestを生成するものとします。

まず、管理用SQLです。ここはPostgreSQLの管理者（慣習的にpostgresというユーザ名にします）権限で行ないます。これらは直接、間接的に、psqlコマンドから発行されるのが普通です。

● アクセス用のユーザの生成

```
CREATE USER apache;
CREATE USER hotta;
```

apacheは、Vine LinuxにおいてApacheが起動する際のデフォルトのユーザ名です[*18]。PHPをブラウザを通して実行する場合はApacheの権限で動きますから、PostgreSQLにアクセスする際もユーザapacheとして認証が行なわれます。また、PHPをコマンドラインで動かす場合はログインユーザ（ここでは hotta とします。適宜自分の環境に合ったユーザ名に読み替えてください）の権限で動きます。

● データベースの生成

```
CREATE DATABASE dbtest;
```

データベースを生成すると、その中にいくつものテーブルを作成することができます。PostgreSQLに接続する際は、接続するデータベースを指定する必要があります。

● テーブルの生成

```
CREATE TABLE tbltest (id int, name text);
```

ここではidという整数の項目とnameという文字列の2項目を持つ、tbltestというテーブルを生成しています。

● テーブルへのアクセス権の設定

```
GRANT ALL ON tbltest TO apache,hotta;
```

アクセス権の設定は各テーブルごとに行ないます。

次にテーブルのレコードを操作するためのSQLです。これらはPHPから呼び出されるのが普通です。その際はapache権限で実行されます。

*18
ただしソースからインストールする場合、デフォルトのユーザ名は 'nobody' となります。その場合、以下も同様に読み替えてください。

● テーブルへのレコードの追加（挿入）

```
INSERT INTO tbltest (id, name) VALUES ( 1, 'name1');
```

● テーブルにあるレコードの置き換え（更新）

```
UPDATE tbltest SET (id = 2, name = 'name2');
```

● テーブルにあるレコードの削除

```
DELETE FROM tbltest;
```

● テーブルにあるレコードの参照

```
SELECT * FROM tbltest;
```

行末の；は、PHPから呼び出す場合は不要ですが、psqlから実行する場合には必要です。

INSERT/DELETEはレコードを増減させますが、テーブルの項目自体が頻繁に増えたり減ったりするようであればそれは設計がまずい証拠であり、そのような状態では安定したシステムの運用は望めません。テーブルの項目は、設計時点で十分に吟味する必要があります。テーブル作成後は、項目の変更はできない（テーブルの削除、再作成となる）と思っておくくらいのほうが無難でしょう。

UPDATE/DELETE/SELECTでは、上記のように何も指定しなければ、そのテーブルに含まれるすべてのレコードを対象とします。たとえば本当に上記のようなDELETE命令を発行すると、そのテーブル内のレコードはすべて失われてしまいます。また、全レコードをSELECTするということは、クライアント側まで全レコードを転送して、一時的ではあるにせよクライアント側のメモリ上に置いておくということですから、レコード数が非常に多い場合はクライアントのマシンのメモリを使い切ってしまうかもしれません。このため実際には「～という条件を満たすレコードのみをXXXXする」という制限をつけることがほとんどです。

PHPでPostgreSQLにアクセスする部分のさわりの例を**リスト1-76**に示します。このコードは説明のためのものですから、実際に動かすための環境についての説明は省略します。なお、すでにデータベースdbtestの中にテーブルtbltestが存在し、ユーザへの適切なアクセス権も与えられているものとします。エラーチェックなども省略しています。

リスト1-76 PostgreSQLにアクセスするサンプル(1-41.php)

```
 1 <?php
 2 $conn = pg_connect("dbname=dbtest");     // サーバへの接続
 3 $sql = "DELETE FROM tbltest";            // 全レコード削除依頼
 4 $result = pg_query($conn, $sql);
 5 for ($i=0; $i<5; $i++)  {
 6     $sql = sprintf("INSERT INTO tbltest VALUES('%d', 'TEST%02d')", $i, $i);
 7     $result = pg_query($conn, $sql);    // レコード追加依頼
 8 }
 9 $sql = "select * FROM tbltest";          // レコード呼び出し依頼
10 $result = pg_query($conn, $sql);
11 for ($i=0; $i<pg_num_rows($result); $i++)   {
12     $rec = pg_fetch_array($result, $i); //  レコードの取り出し
13     printf("%d 行目：id=%d, name=%s\n", $i+1, $rec['id'], $rec['name']);
14 }
15 pg_close($conn);                          // 切断
16 ?>
```

これを実行すると、**図1-44**のような出力が得られます。

図1-44 1-41.phpの実行結果

```
hotta@star ~/public_html/1-5$ php -q 1-41.php
1 行目：id=0, name=TEST00
2 行目：id=1, name=TEST01
3 行目：id=2, name=TEST02
4 行目：id=3, name=TEST03
5 行目：id=4, name=TEST04
```

スクリプト終了後も、テーブルtbltestの中には上記のレコードが保存されています。ではスクリプトの内容について簡単に説明しておきます。

- **2行目**

データベース名を指定してデータベースへの接続を行ないます。今後この接続においては、基本的にはこのデータベースに属するテーブルのみを操作することができます。

- **3～4行目**

レコードを全件削除するSQL文を作成して、サーバに実行を依頼します。

- **5～8行目**

レコードを挿入するSQL文を作成して、サーバに実行を依頼します。これにより5件のレコードが新たに生成されます。

- **9～10行目**

レコードを全件読み込むSQL文を作成して、サーバに実行を依頼します。サーバから受信した実行結果はクライアント側のメモリ上に全件分のレコードを保持する仮想的な配列とし

て保持されますが、この配列はPHP固有のpg_XXX()関数でのみ操作できます。

●11～14行目

受信したレコードを1レコードずつ（ローカルの仮想的な配列から）取り出しては画面に表示します。pg_fetch_array()で1行分の内容を$recに格納し、$rec内の各要素の値を表示しています。ループの終了判定は、該当するレコードの行数としています。

●15行目

PostgreSQLサーバとのコネクションを切断します。

一連の処理の流れを**図1-45**にまとめてみました。第2部を読むにあたって、これを十分に頭に入れておいてください。実際にデバッグする場合は、個々のSQL文の作り方がわからなかったりSQLが意図した振る舞いをしないことがあるかもしれませんが、それはPHPとは無関係です。psql上で何度もSQL文を手でタイプして実行し、十分納得がいくSQL文ができたところでそれをPHPスクリプトに組み込むようにしましょう。

図1-45　DBMS連携における処理の流れ