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前に、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Windows 95/98/NTでも動作するようになりました。

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、企業でも安心して使用できます。IBMによる商用サポートも開始されました。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ブラウザというソフトウェア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。

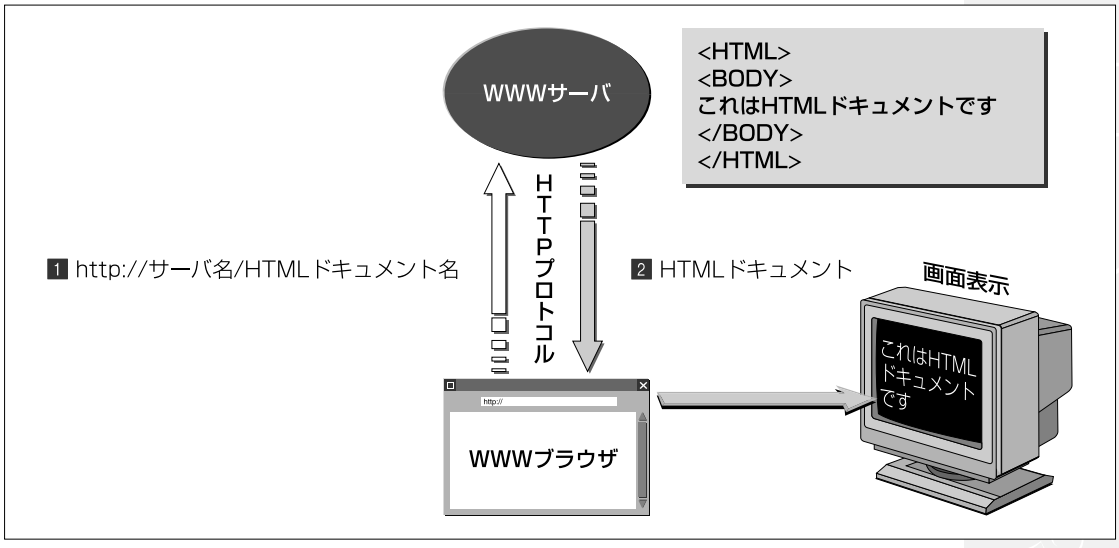
WWWではHTML(HyperText Markup Language)という言語を使用してコンテンツ(文書の内容)を表現します。ブラウザはURL(Uniform Resource Locator)を使って、サーバ上にあるいずれかのコンテンツを要求します。URLとは

http://WWWサーバの名前/ドキュメントへのパス

という形式の表記方法です。

サーバ・クライアント間における情報のやり取りには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ページに対する要求が強くなってきています。

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、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。

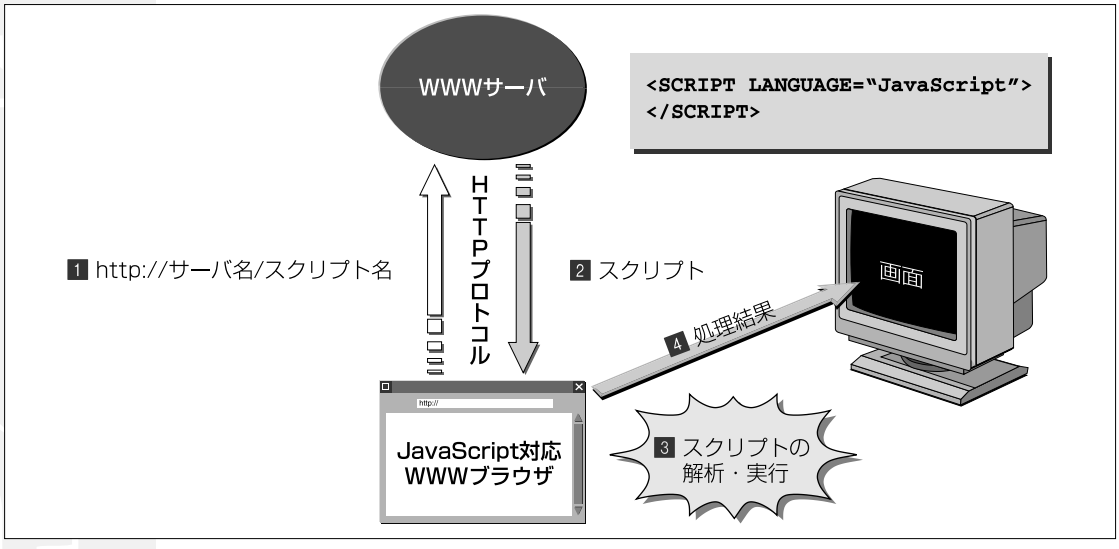
また、出力画面を動的に生成できるようになると、従来は専用の端末やクライアント・ソフトウェアを使って行われていたような、いわゆる事務処理をはじめとする適用業務を、Webサーバを経由してブラウザ上で実行できるようになります。Webサーバはブラウザからの要求を実行し（または要求を別のサーバに送ってその応答を受信し）、結果だけをブラウザに送り返してやればよいわけです。クライアント側にWebブラウザを用意しておくだけで、多くの定型業務がWebベースに移行できる可能性があります。クライアント側にはほとんど費用がかかりませんから、これは大幅なTCO（Total Cost of Ownership）の削減につながります。

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。

図1-2 クライアントサイド・スクリプト

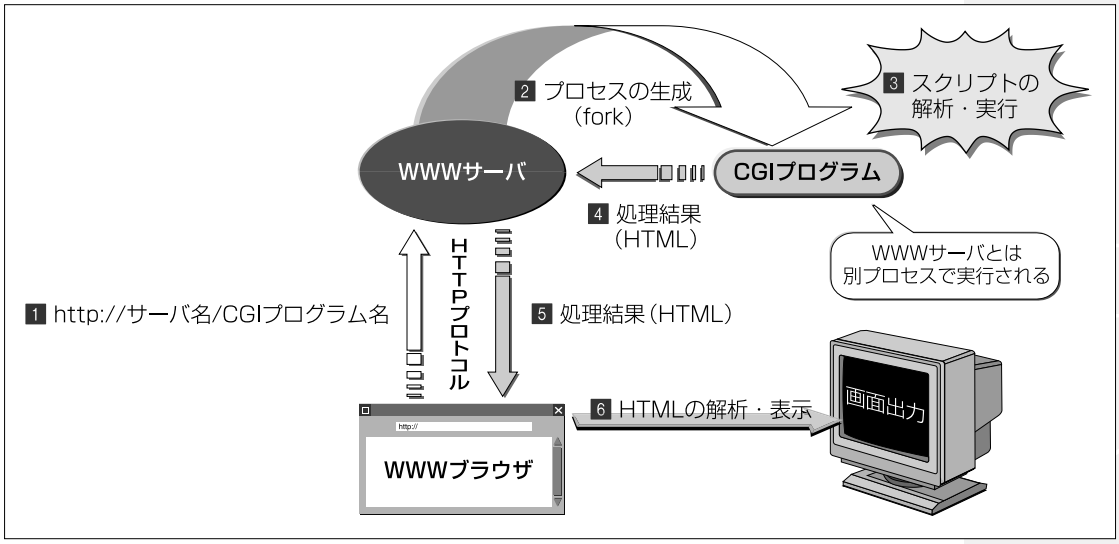


1.2.2 CGI (Common Gateway Interface)

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。

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。

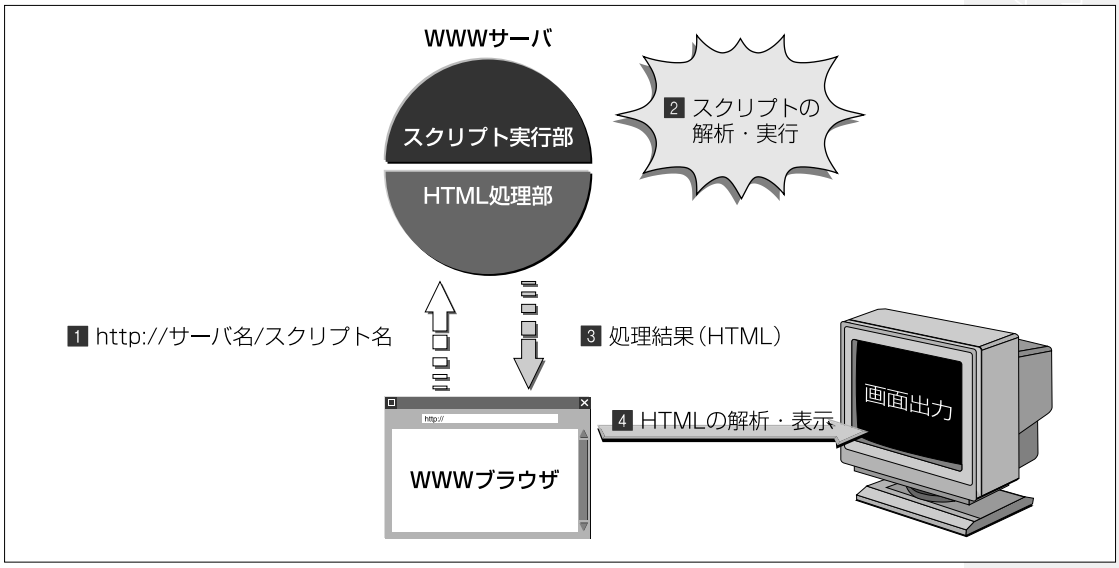
図 1-3 CGI



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。本書の主目的であるPHPもこのタイプです。プロセスを生成しないため、CGIに比べて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、サーバプログラム自体が大きくなる傾向があります。

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。

そしてついに2000年の5月22日に、待望のPHP4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われました。本書の執筆時におけるPHPの最新版はPHP4.0.1pl2です。

PHP3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 H」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

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。

●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に精通してい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ライクな文字列演算子や関数が豊富です。半角英数字に限られますが正規表現[*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の一部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可能です。

●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といったフリー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。

1.3.3 PHP と他言語との比較

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は代表的なクライアントサイド・スクリプトです。これは、動的な画面を生成するための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間、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。

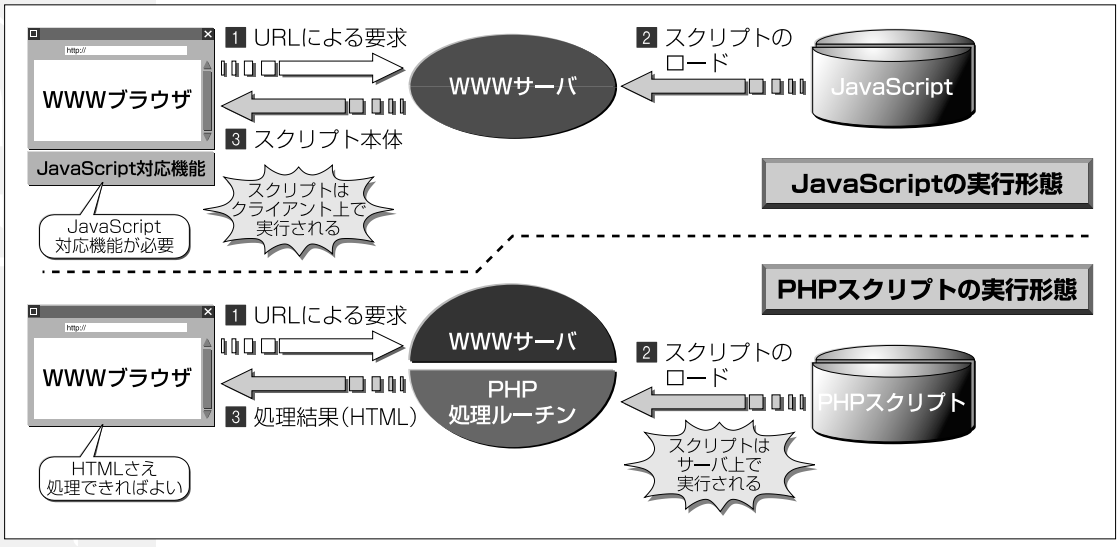
デメリットとしては、ブラウザそのものがその言語をサポートしていなければならないため、ブラウザによっては意図したものが表示されないといった問題が発生することが挙げら

*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。

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、その言語で記述できることであれば何でもできます。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、最悪の場合マシンがダウンしてしまうことも考えられます。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ものが要求されます。

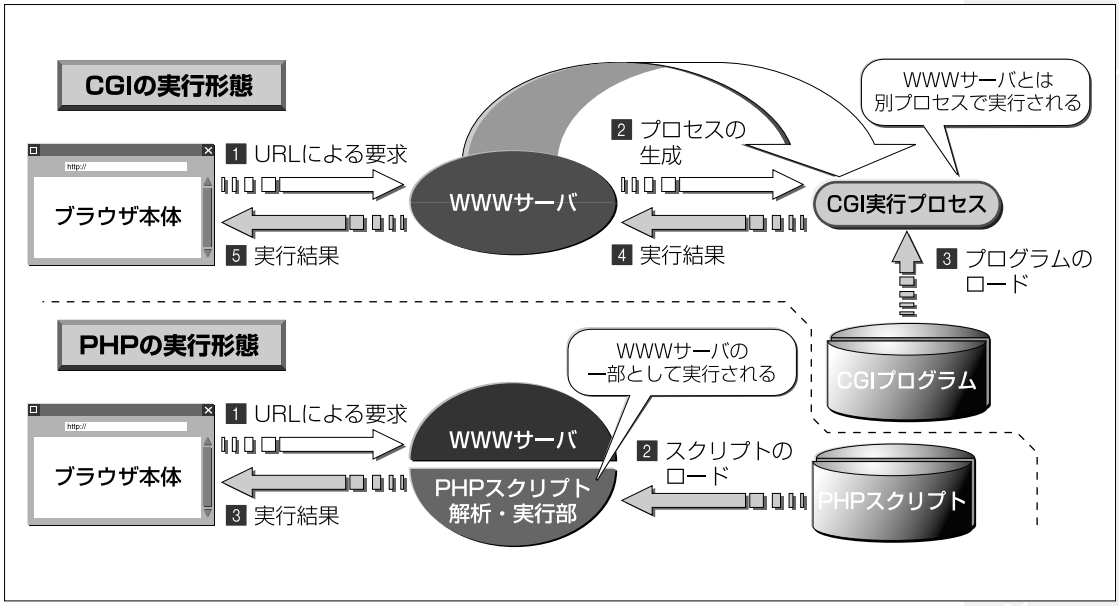
PHPをはじめとするサーバサイド・スクリプトは、CGIに比べればサーバマシンにプロセスの起動／終了に関する負荷がかからないので、軽量であるというメリットがあります。しかし、Webサーバプログラム自体にスクリプトのインタプリタ機能をモジュール（コンパイル済みのオブジェクト）というかたちで組み込む必要があるため、Webサーバ自身の実行イメージのサイズが大きくなります。またスクリプトの実行部で万が一致命的なエラーが発生すると、（理論的には）Webサーバもろとも異常終了してしまうなどのデメリットもあります。

*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。

前者の問題に関しては、ApacheにもDSO（Dynamic Shared Object）機能が組み込まれ、必要なときにモジュールを動的に読み込む（実行時にリンクする）ことで、なるべくメモリを消費しないような工夫がなされています。モジュールを読み込む場合でもプロセスを新たに起動するわけではないので、CGIほどのオーバーヘッドはかかりません。

また、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。

図1-6 CGIとPHP



これらの処理パターンをまとめると、以下のようになります。

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を使って実用的な業務を行うには、何らかのデータベースと組み合わせて使用するのが現実的でしょう。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 ― データベース管理システム）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。本稿執筆時点における国際化対応最新版はphp-3.0.15-i18n.tar.gzです。PHP国際化のページ（http://php.jpnnet.com/）より国際化バージョンの特徴について抜粋してみます。詳細は国際化バージョンに付属するREADME.i18nを参照してください。

国際化対応版における主な特徴

- **PHP3スクリプトファイル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（自動認識も可能）。**
- **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。**
- **POST/GET/COOKIEで渡されたデータを内部コードへ自動変換可能。**
- **内部コードにはSJIS、EUC、UTF-8を指定可能。**
- **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[*3]。**
- **configureの際に「--enable-i18n」を追加するだけで作成できます。**
- **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。**
- **追加PHP3関数。**

PHP4における日本語の状況ですが、LinuxでEUCコードを使う場合およびWindows NTでShift JISコードを使う場合に限っては、特に問題はなさそうです。PHP4を使う場合、正規表現など一部の機能については、そのままでは日本語を使用できません。塚田卓也氏が提供している国際化拡張用コードにより、一部コード変換等の機能が提供されています。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。

*3 この機能が邪魔になることもあります。たとえばmail()関数でファイルを添付すると、この機能に起因する不具合が起こることが報告されています。

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

PHP 4.0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。現時点でApache、ISAPI（Microsoft IIS等）、NSAPI（Netscape Enterprise Web Server、iPlanet、AOL Server等）、Java Servlet（Tomcat）などがWebサーバのネイティブAPIとしてサポートされています。

● セッション管理機能のサポート

PHP 3.0で要望の強かった、HTTPセッション管理機能がサポートされました。この機能は長い間PHPlibでサポートされていたものですが、ついにPHPネイティブに組み込まれることになりました。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（PHP3では**php3.ini**）におけるすべて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）のみです。

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（Incompatibilities between PHP 3.0 and PHP 4.0 ＜http://www.php.net/version4/incompatibilities.php＞からの抜粋）。

- **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（PHP 3.0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限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。PHP 3.0 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、PHP 4.0 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）。
- PHP 3.0 の動的拡張（Windows 上の php3_*.dll）は、PHP 4.0 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

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されていますが、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。本書ではPHPはApacheのDSOモジュールとしてインストールしておき、Webサーバ上にサンプルのコンテンツを置いて、ブラウザからURLを指定して間接的に呼び出すタイプの実行形式を想定しています。スクリプトの実行例も、この環境を前提に作成しています。

PHP自体を単独実行型のコマンドとしてインストールすることもできます。この環境があれば、PHPスクリプトを通常のシェルスクリプトと同等に扱うことができるため、関数単位など短いスクリプトの場合、より手軽にデバッグすることができます。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てしまいます。

PHPの文法はC言語をベースにしており、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。PHP3ではデフォルト(特に指定されない場合の既定値)の拡張子は.php3でしたが、PHP4では.phpに変更されました。もちろん設定次第で別の拡張子に変えることもできます。PHP3の環境でこの章のサンプルを実行する場合には、適宜.phpを.php3に読み替えてください。なお特に記述がない限り、これらのコードはPHP3とPHP4のどちらでも実行できます。

2.1 PHPスクリプティングことはじめ

プログラミング言語を解説するにあたって最初に紹介する例題は、ご多分に洩れず**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 **<?** 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してくれず、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。

出力したい文字列がアルファベットだけから構成される文字列であればそのまま書いてもかまいませんが、そうでなければ文字列全体を単一引用符（**'**）または二重引用符（**"**）でくくって指定しなければなりません。

ここで少し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。こうするとApacheの機能により、ブラウザからは

http://localhost/~hotta/test1.php

といったURLでアクセスすることができます。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。

*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。

図 1-7 test1.phpの実行結果

```
Hello, world!
```

ところで、上記の例は実はあまり適切ではありませんでした。なぜなら **Hello, world!** と出力するだけなら、わざわざPHP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。

これをブラウザから

http://localhost/~hotta/test.txt

としてアクセスすると、上記とまったく同じ出力が得られるはずです。

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を往々にして見かけます[*7]。PHPに限らず、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（PHP4もそのひとつです）。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自体を意識していない）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を見つけられずに“File not found”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上で動作する、LanManager互換ファイルサーバ。UNIXサーバを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。かくいう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://hidemaru.xaxon.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が参照できます。

2.2 変数を使ってみる

今度は単純なHTMLコンテンツではできない芸当として、**Hello,** の次に任意の文字列を表示させてみましょう。表示したい文字列はユーザが指定します。

リスト 1-2 test2.php

```
<?
    print "Hello, " . $hensuu . "¥n";
?>
```

変数とはなんらかの値を受け取ったり保持したりする箱のことで、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、システムの予約語と重複しない範囲でユーザが自由につけることができます。ここでは $hensuu という名前をつけてみました。ピリオド（**.**）は文字列の結合を指示する演算子です。**+** を使うと数字の足し算という意味になるので注意してください。では$hensuu に値を渡してみましょう。**test2.php** に値を渡すためには、ブラウザで

http://localhost/~hotta/test2.php?hensuu=PHP!

とアクセスします。このように、スクリプトや一般のプログラムに対して与えられる、**hensuu=world!** といった付加的な情報のことを、「引数」あるいは「パラメータ」と呼びます。URLの場合は特別に「クエリ文字列」と呼ぶこともあります。

図 1-8 test2.phpの実行結果

```
Hello, PHP!
```

このようにURLの中で **?** が現れると、それ以降は **変数名＝値** という引数の並びとみなされ、PHPスクリプト内部では **$ 変数名** としてアクセスできます。**PHP!** のところをいろいろ変えてみて、指定した値が正しくスクリプトに渡されていることを確認してください。なお複数の値を渡す場合は **?name1=val1&name2=val2...** というように、引数の間を **&** で区切ります。

Column

コマンドライン版PHP

リスト1-2のような1行スクリプトを実行するためだけにブラウザを使うのはなんだかめんどうだとは思いませんか。このような人のために、PHPではコマンドとして動作する実行イメージを作ることができます。これを利用すると、スクリプトの1行目に

```
#!/usr/local/bin/php
```

などと書いてからスクリプトに実行属性を与えれば、PHPスクリプトをコマンドとして使うこともできます。コマンドライン版PHPの作り方は第4部を参照してください。コマンドライン版ではいくつかの起動オプションがあります。

コマンドライン版はデバッグ時にはお手軽でよいのですが、HTMLタグがそのまま出力されてしまうので、出力を適当に読み替えてやることが必要です。

ところでコマンドライン版のPHPを使う場合、スクリプトに変数を渡すためにコマンドラインで

```
php -q test2.php?hensuu=world!
```

などとしてもうまくいきません。これは、PHPが"test2.php?hensuu=world!"という名前のスクリプトファイルを探そうとするからです[*10]。コマンドライン版のPHPに変数を渡すには、シェルとしてbash(sh)を使う場合は

```
export  QUERY_STRING="hensuu=world!"
php -q test2.php
```

tcsh(csh)を使う場合は

```
setenv QUERY_STRING "hensuu=world!"
php -q test2.php
```

としてください。QUERY_STRINGなどの環境変数に関しては「5.4.2　環境変数」を参照してください。

なお、別解として、オプションのargsを使うこともできます。

●コマンドライン版PHPの起動オプション

```
~$ php -?
Usage: php [-q] [-h] [-s] [-v] [-i] [-f <file>] | {<file> [args...]}
  -q              静粛モード。HTTPヘッダを出力しない
  -s              ソースを装飾タグつきで表示する
  -f<file>        <file>をパースする。'-q'が指定されているとみなす
  -v              バージョン番号を表示する
  -c<path>        php.iniファイルがあるディレクトリ
  -a              会話的に実行する
  -d foo[=bar]    php.iniのエントリとしてfooに'bar'という値を割り当てる
  -e              デバッガ／プロファイラ用の拡張情報を生成する
  -z<file>        Zend拡張<file>を読み込む
  -i              PHPの内部情報をHTMLで出力する(phpinfo()と同じ)
  -h              このヘルプを表示する
  args            任意の引数の並び。スクリプト内部では$argc(引数の数)、$argv[](引き数並び)で
                  取り出すことができる
```

*10
正確には、シェルが特殊文字(!)などを別の文字列に置き換えようとするなどコマンドライン入力特有の動きをしますが、詳細は割愛します。

2.3 HTMLに埋め込んで使う

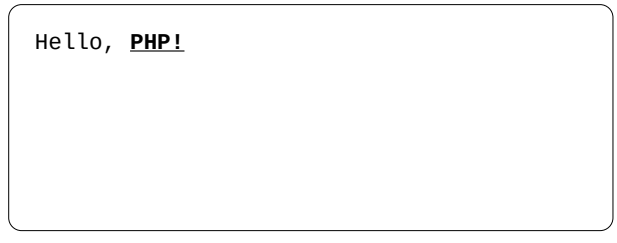
通常WebコンテンツはHTML言語で記述します。PHPはHTMLの中に部分的にスクリプトを埋め込むことができるという点に大きな特徴があります。早速先程の**test2.php**を、少しまともなHTMLファイルにしてみましょう。ついでにユーザが指定した部分だけ太字にし、アンダーラインもつけてみます。

リスト1-3 test3.php

```
<HTML>
<BODY>
Hello, <U><B><? print $hensuu; ?></B></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、**<?** 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われます。**?>** に出会うとまたHTMLモードに戻ります。上記のコードの別解として、次のように書くこともできます。

リスト1-4 test31.php

```
<?
print   "<HTML>¥n";
print   "<BODY>¥n";
print   "Hello, <U><B> $hensuu </B></U>¥n";
print   "</BODY>¥n";
print   "</HTML>¥n";
?>
```

リスト1-5 test32.php

```
<?
print   "<HTML>¥n"
      . "<BODY>¥n"
      . "Hello, <U><B> $hensuu </B></U>¥n"
      . "</BODY>¥n"
      . "</HTML>¥n";
?>
```

リスト1-6 test33.php

```
<?
print "<HTML>¥n<BODY>¥nHello, <U><B> $hensuu </B></U></BODY>¥n</HTML>¥n";
?>
```

どれでも自分が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。普段からHTMLでコンテンツを書いている人なら知っていることだと思いますが、各行についている改行コードは、単にブラウザで「ソースの表示」を行った際の読みやすさのため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、普段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。

なお、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（変数が実行時点の値に置き換えられること）が行われず、意図した結果が得られません。

上記の三つ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。

リスト1-7 test34.php

```
<?
print <<<EOD
<HTML>
<BODY>
Hello, <U><B>$hensuu</B></U>
</BODY>
</HTML>
EOD;
?>
```

test34.phpで使った書式は「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だけの行で文字列の指定が終わります。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（ヒアドキュメントはPHP4で追加された機能です）。

2.4 ほかのスクリプトに値を渡す

次に、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以下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べません（もちろんこの部分をPHPで書いてもかまいません）。

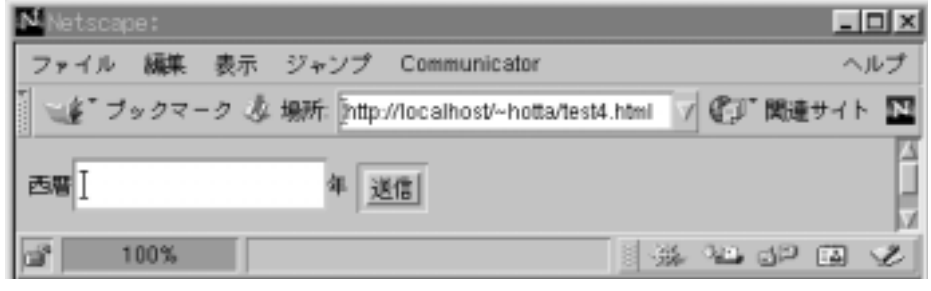
リスト1-8 test4.html

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

これを実行すると、以下のような入力エリアと送信ボタンが表示されます[*11]。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、スクリプトがEUCコードで書かれているか、および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。

*11 これはLinux上のNetscape Communicator4.7の画面です。ブラウザの種類によって、表示されるボタンの形状が異なる場合があります。

図1-10 test4.htmlの実行画面



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、ここでは軽く触れておきます。<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** という変数に格納され、PHPスクリプト側では$yyyyという名前で参照することができます。

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。

次に変数を受ける側の **test4.php** です。起動した時点で$yyyyにはユーザからの入力値が入っているので、そのまま参照することができます。

リスト1-9 test4.php

```
<HTML>
<BODY>
<?
$meiji = $yyyy - 1866;
$taisho = $yyyy - 1911;
$showa = $yyyy - 1925;
$heisei = $yyyy - 1988;
print "西暦 $yyyy 年は、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。即値とも呼ばれます。数値とは限りません）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 **西暦$yyyy年は** というように変数名と次の文字列をくっつけて書いてしまうと、**$** が変数の始まりですから、PHPのパーサ（構文解析部）は **$yyyy年は** 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。対応策としては、**test4.php** のように変数の後ろに空白を置くか、後述するprintf()のフォーマット指定を使います。

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。画面には以下のように表示されます。

図1-11　test4.phpの実行結果

```
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年の途中にあることが多いので、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。

2.5 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コンテンツ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$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**）。

何らかの条件にしたがった処理の分岐にはENDIF構文を使います。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（True：$yyyyの長さが0、すなわち$yyyyに値が入っていない）であれば<FORM>～</FORM>までが実行され、そうでなければ（$yyyyに値が入っていれば）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。

$yyyyに値が入っているかどうかは、$yyyyの長さが0かどうかを調べることによって判断していますが、これにstrlen()という「関数」を利用しています。strlen()のように、$のつかないシンボル文字列に**（引数）**がついたものを関数と呼びます。関数とは、0個以上の引数をとり（つまり引数はなくてもよい）、何らかの値を返すように作られている一連の処理に対して、ほかから呼び出すための名前をつけたものです。関数は変数とは別の名前空間を持っているので、変数$abcと関数abc()にはまったく関連性はありません。

リスト1-10 test41.php

```
<HTML>
<BODY>
<? if (strlen($yyyy)==0): ?>
<FORM action="test41.php">
西暦<INPUT TYPE=text NAME=yyyy>年
<INPUT TYPE=submit>
</FORM>
<? else:
    $meiji = $yyyy - 1866;
    $taisho = $yyyy - 1911;
    $showa = $yyyy - 1925;
    $heisei = $yyyy - 1988;
    print "西暦$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>
```

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）。strlen()もそのうちのひとつで、「与えられた引数を文字列とみなし、その長さを返す」という働きを持っています。今まで使ってきたprintも実は関数として実装されており、**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そうな点です。

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、これは関数ではありません。なぜなら、**IF()**文は値を返さないからです。**IF(条件)**の後ろはセミコロン（**;**）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。

このスクリプトでは、**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、HTMLモードの部分（<FORM>以下の行）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。

2.6 処理ロジックを分割する

test41.phpで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」ボタンなど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。

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、思った通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といいます。プログラミングとは、すなわちデバッグなのです。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1-11　test5.php

```
display_input_area($yyyy);           //  入力エリアの表示
if (strlen($yyyy) > 0 && input_is_valid($yyyy)):
    $result = calc_gengou($yyyy);    //  元号の計算
    display_result($yyyy, $result); //  結果の表示
endif;
```

どうでしょうか？ だいぶんすっきりしてきました。

PHPでは // 以降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。ただし、引用符で囲まれた文字列の中の // はそのまま評価されます。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。また分かり切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」関数として定義しようと考えています。もっとも（起動直後などで）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、**$yyyy**が入力されている（長さが1以上である）場合のみ入力チェックを行うようにしました。**&&** 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。ちなみに「または」を表すのは || です。

2.7 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は本当にそう動いてくれるのでしょうか？

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（**test6.php**）。

新たなキーワード**function**と**return**が出てきました。functionはその名の通り、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および予約語で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としています。また、関数はreturn命令によりひとつ以上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。

リスト 1-12 test6.php

```
function func1()
{
    print "func1()<br>¥n";
    return  False;
}
function func2()
{
    print "func2()<br>¥n";
    return  True;
}
if (func1() && func2()):
    print "真です<br>¥n";
else:
    print "偽です<br>¥n";
endif;
```

⇩

図 1-12 test6.phpの実行結果

```
func1()
偽です
```

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及ぼしません。**test6.php**で一番はじめに実行されるのはIFの行です。IFの()内の評価式が複数あれば、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**func1()
¥n**という表示を行ったあと、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決まっています。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、**偽です**の表示を行い、プログラムを終了します。これはfunc2()に入ればかならず行われるはずの**func2()
¥n**の出力が行われていないことから明らかです。

ここで疑り深い筆者は、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1-13 test7.php

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

strlen()は前述のように、文字列を引数として取り、その長さを返す関数です。**test7.php**では、これをわざと引数を表示するという関数として再定義しました。

ここで、結果を表示するprint文で**\$text**というちょっと変わった表記を行っています。実は、ここは単に**strlen($text)**という文字列を表示したいのですが、この通り書いてしまうとprint文の実行時に$textが先に評価されてしまい、( )の中に$textの中身が展開されて表示されてしまうからです。もっともこの例の場合、この時点で$textの中には値が入っていないので、実際には何も表示されません。ここではそれを避けるために**$**の前にバックスラッシュ(＼、日本語のキーボードでは**¥**で刻印されているものが多い)をつけて、**$**の意味を打ち消して(エスケープして)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** *12 という文字列を渡しているので、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1-13 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。まさにそのまんまです。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うことができます。

*12
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。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。

2.8 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 1-14 test5.php

```
display_input_area($yyyy);          //  入力エリアの表示
if (strlen($yyyy) > 0 && input_is_valid($yyyy)):
    $result = calc_gengou($yyyy);   //  元号の計算
    display_result($yyyy, $result); //  結果の表示
endif;
```

ここでいくつかの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。紙面の都合もあるので、ソースファイルプログラム（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()を以下のようにしてみました。

リスト 1-15 display.inc

```
<?
//  入力画面の表示
function    display_input_area($yyyy)
{
?>
<FORM action="test5.php">
西暦<INPUT TYPE=text NAME=yyyy SIZE=5 VALUE="<? echo $yyyy; ?>">年
<INPUT TYPE=submit VALUE=“送信”>
</FORM>
<?
}
?>
```

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、引数として西暦4桁を取り、<INPUT>タグのVALUE属性で、$yyyyの値をデフォルトで入力エリアの中に表示しています。

また、**test5.php**のほうにはプログラムの先頭に

```
include("display.inc");
```

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文の行をdisplay.incの中身で置き換えます。なお、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かもしれません[*13]。

2.9 入力のチェック

ユーザからの入力を受けつけて処理を行うにあたり、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1-16 input.inc

```
<?
function    input_is_valid($yyyy)
{
    if (ereg("^[12][0-9]{3}$", $yyyy)):
        return  True;
    endif;
    return  False;
}
?>
```

この関数では、入力$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。ここでは新しい関数ereg()を使っています。これは「第二引数として与えられた文字列が第一

*13
.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。

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という文字で始まり、そのあとに0から9までのいずれかの文字が3個続き、それで文字列が終わっているもの」を示します。これを一般的な言葉で書けば、1000～2999という西暦の範囲を表すことになります。

2.10 複数の値を扱う

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。

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

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表現方法で、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。

“man 7 regex”manページでは、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[*14]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ので、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互換の正規表現を使うこともできます。詳細は付属CD-ROMに収録しているPHPマニュアルを参照してください。

●正規表現のパターン例

^	文字列の始まり	{n}	直前の文字のn回の繰り返し
$	文字列の終り	[abc]	a,b,cいずれかの1文字
?	直前の文字の0個または1個の繰り返し	[a-z]	小文字1文字
*	直前の文字の0個以上の繰り返し	[0-9A-Za-z]	英数字
+	直前の文字の1個以上の繰り返し	[^0-9]	数字以外

*14 オライリー・ジャパンより「詳解 正規表現」という書籍が、翻訳版として出版されています。

一般のコンパイラ言語[*15]では、配列は変数名に「添字（そえじ）」（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。

calc_gengou()では、入力値に対して四つの計算を行い、ひとつのreturn文で四つの値を返します。

リスト1-17 calc.inc

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

2.11 書式つき出力

最後にdisplay_result()です。

リスト1-18 result.inc

```
<?
//  結果の表示
function    display_result($seireki, $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",
        $seireki,
        $wareki["明治"], $wareki["大正"],
        $wareki["昭和"], $wareki["平成"]);
}
?>
```

*15 事前にコンパイルを行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。

display_result() は二つの引数 $seireki、$warekiを取りますが、呼び出す側の**test5.php**のほうでは$yyyy、$resultを渡していました。渡す側における引数のことを実引数、受け取る側を仮引数と呼びます。仮引数は引数を受け取るための受け口で、実引数とは別の名前をつけてもかまいません。技術的な話をすると、関数コール時には値の「コピー」が一時的に作られ、それが関数に渡される（値渡し）ので、たとえ関数内部で引数の値を変更したとしても、呼び出し元の変数の値は影響を受けません。関数の中で呼び出し側変数の値を変更したい場合は、後述の「参照渡し」を行う必要があります。

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えます。第一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。

では、これらのインクルードファイルを取り込むように**test5.php**を変更し、早速実行してみましょう。

リスト1-19　変更を加えたtest5.php

```
<?
include("display.inc");
include("input.inc");
include("calc.inc");
include("result.inc");
display_input_area($yyyy);          //  入力エリアの表示
if (strlen($yyyy) > 0 && input_is_valid($yyyy)):
    $result = calc_gengou($yyyy);   //  元号の計算
    display_result($yyyy, $result); //  結果の表示
endif;
?>
```

図1-14　test5.phpの実行結果

どうでしょうか？　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かと思います。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うように機能追加したスクリプトを作成します。

Hypertext Preprocessor

HP

Chapter-3 PHPの文法を知ろう

前章では主に初学の人を対象として、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、CD-ROM付属の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スクリプトでは、HTML文書の中にスクリプトを埋め込むことができます。

```
<HTML>
<BODY>
<? 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の出力を行うこともできます。

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

どちら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。

● **<? ～ ?> で囲む**

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

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

```
<?
    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。

● **<?php ～ ?> で囲む**

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

将来のXMLなどとの相互運用性を考慮して、今後はこちらの書式を使うほうがよいでしょう。ただし、第1部では都合により一番目の書式を使っています。

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

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

● **<% ～ %> で囲む**

これはマイクロソフトの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。

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

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。

```
ECHO("これはOK");
Echo("これもOK");
echo("これでもOK");
```

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。

```
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()** 形式による各種機能の呼び出し方法のことを関数といいます。PHPでは多数の組み込み関数を用意しており、プログラマはこれらを組み合わせて複雑な処理を行うことができるようになっています。なお、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、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)スクリプトの一部分ということができます。また、すべての式は「型」を持ちます。「型」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）（PHP4のみ）
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（PHP4のみ）

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。

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かに過ぎません。

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。

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

⇩ 実行結果

```
$foo=0 string
$foo=1 string
$foo=2 integer
$foo=3.3 double
$foo=15 integer
```

3.10 型キャスト

プログラムを組んでいると、ある型を持つ変数を、強制的に別の型として評価したい場合があります。強制的に行う型変換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。

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

可能なキャストの書式を次に示します。

(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、そのあとに指数値(オプション)があるものです。指数値は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。**

```
$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」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。

```
$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。

```
$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。

```
$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）を行います。配列をソートするには、ソートしたい型に応じて各種のソート用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。

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。

```
$a = 1;                    //  単純変数
$a[0] = 2;                 //  一次元配列
$a[0][1] = 3;              //  二次元配列
$a[0][foo] = 4;
$a[0][foo][bar] = 4;       //  三次元配列
```

PHP3では、文字列の中で直接多次元配列を参照することができませんでした。たとえば、次の例では望む結果を得られません。

```
$a[3][bar] = "Bob";
echo "これは動作しません: $a[3][bar]";
```

PHP3では、上の例は「これは動作しません: Array[bar]」を出力します。これは、変数のうちの**$a[3]**の部分を先に評価してしまい、後ろの**[bar]**は単なる文字列とみなされてしまうからです。この問題を解決するには、文字列結合演算子を使用して以下のようにする必要があります。

```
$a[3][bar] = "Bob";
echo "これは動作します: " . $a[3][bar];
```

さらにPHP4では、(文字列の中で)配列参照を大括弧**{}**で囲むことにより、この問題を回避できるようになりました。

```
$a[3][bar] = "Bob";
echo "これは動作します: {$a[3][bar]}";      // PHP4ならOK
```

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。

```
<?
    $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

PHP3では、変数に値を代入する際、常に右辺から左辺への値のコピーが行われていました。このため、ある変数の値をほかの変数に代入したあとで代入元の変数を変更しても、すでに代入したほかの変数には影響を与えません。この意味で値のコピーは安全ですが、その半面、特に長い文字列の代入の場合、記憶領域や処理速度が犠牲となります。

PHP4では、変数に対して参照による代入が行えるようになりました。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。PHP4では基本的にコピーを行わなくなったため、代入はより高速になります。ただし、速度の向上が体感できるのは、重いループや大きな配列またはオブジェクトを割りつける場合に限られると思われます。

参照により代入を行うには、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。

```
<?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。

```
<?php
$foo = 25;
$bar = &$foo;      // これは有効な代入です。
$bar = &24;        // 無効です。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、文字通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。ただし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、単一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ってから評価が行われます。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****をおいて、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}	この正規表現にマッチする文字シーケンスは、8進数表記の1文字。 PHPでは8進数を“ \666 ”のようにして表す
\x[0-9A-Fa-f]{1,2}	この正規表現にマッチする文字シーケンスは、16進数表記の1文字。 PHPでは16進数を“ \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。定数シンボルの定義を追加するにはdefine()関数を使います。しかしながら、定数シンボルと（引用符で囲まれない）文字列定数は見かけ上は判別できないので、注意して使用することをお勧めします。以下に例を示します。

```
echo CONSTANT;              //  "CONSTANT" を出力します(文字列定数)
define("CONSTANT", "Hello world.");
echo CONSTANT;              //  "Hello world." を出力します(定数シンボル)
$CONSTANT = "Other world.";
echo $CONSTANT;             //  "Other world." を出力します(文字列変数)
echo CONSTANT;              //  "Hello world." を出力します(定数シンボル)
echo CONSTANt;              //  "CONSTANt" を出力します(文字列定数)
CONSTANT = "Other world.";  //  文法エラー
```

プログラマがdefine()で定義した定数のほかに、PHPで予約されている定義済み定数(**表1-3**)があります。

表1-3　定義済み定数

定数	意味
__FILE__	現在処理中のスクリプトファイルの名前 includeあるいはrequireにより読みこまれたファイルで使用された場合、親ファイルではなくインクルードされたファイルとなる
__LINE__	現在のスクリプトファイルの、現在処理中の行の番号 includeあるいはrequireされたファイルで使用された場合、includeされたファイルの中での位置となる
PHP_VERSION	現在使用中のPHPパーサのバージョンを表す文字列 例：'4.0.1pl2'
PHP_OS	PHPパーサを実行中のオペレーティングシステムの名前 例：'Linux'
True	真
False	偽

そのほか、**E_**で始まるエラーレポートのレベル設定用定数があります。詳細は第3部の関数リファレンスのerror_reporting()を参照してください。

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、ひとつ以上の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。

```
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。関数本体は **{ }** 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。なおPHP3では、関数は参照される前に定義されていなければなりませんでした（前方参照ができなかった）が、PHP4では前方参照がサポートされ、関数定義が関数を参照している部分のあとにあってもよくなりました。

関数では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えます。

```
/*  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。

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、呼び出し側で実際に指定する変数や定数を実引数と呼びます。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。可変個数の引数はサポートしていませんが[*16]、配列を渡すことにより同等のことを行えます。

*16 PHP4ではサポートされています。第3部の関数リファレンスのget_variable_*を参照してください。

```
/*  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、その引数を参照渡しとする必要があります。関数の引数を常に参照渡しとしたい場合には、関数定義においてアンパサンド（**&**）を仮引数の前に付加します。もちろん実引数は変数でなければなりません。

```
/*  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
```

関数に対して、ある変数を参照渡しで与えたいが、デフォルトでは参照渡しにしたくない場合、関数コール時に実引数の前にアンパサンドを付加することができます。

```
function foo($bar)   {
    $bar .= "猫である";
}
$str = '我輩は';
foo($str);
echo $str;    // '我輩は' を出力します
foo(&$str);
echo $str;    // '我輩は猫である' を出力します
```

●デフォルト引数

関数の引数定義において、スカラー引数に関しては、引数が指定されない場合に使用するデフォルト値を定義することができます。

```
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。

```
function cook($type = "卵入り", $what)  {
    return "$type$what";
}
echo cook("お好み焼き");    // 単に"お好み焼き"を表示します
```

次の例はうまく動作します。

```
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(次頁)。

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。

```
$str="TEST";          // グローバルスコープ

Function Test() {
    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
```

PHPではC言語と異なり、特に宣言しない限り、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。ただし未定義のローカル変数を参照しても文法エラーにはならない点に注意してください（C言語などのコンパイル型言語では通常エラーとなります）。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。以下に例を示します。

```
$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。
```

関数内部から操作できるグローバル変数の数に制限はありません。
PHPには静的（スタティック）変数も用意されています。

```
Function Test() {
    $count=0;

    echo $count;
    $count++;
}
```

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。

```
Function Test() {
    static $count=0;    //  初回のみ初期化を行うstatic変数
    echo $count;
    $count++;
}
```

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 1-4 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。

```
<?
echo "10 / 0 = ", 10 / 0, "<BR>¥n"; // 普通はエラーだが？
echo "10 / 1 = ", 10 / 1, "<BR>¥n";
?>
```

⇩ 実行結果

```
10 / 0 =
Warning: Division by zero in /home/sim/public_html/a.php3 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、CD-ROMに収録している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1-5　ビット演算子

例	名前	式の値
$a & $b	論理積(and)	$aおよび$bの両方にセットされているビット
$a ¦ $b	論理和(or)	$aまたは$bのどちらかにセットされているビット
$a ^ $b	排他的論理和(xor)	$a または $b のいずれかのみにセットされており、両方にセットされていないビット
~ $a	否定(not)	全ビットを反転したビット
$a << $b	左シフト	$aのビットを左に$bビットずらす(各シフトは「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(各シフトは「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をすべて0に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。

```
<?
    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);
?>
```

⇩ **実行結果**

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

printf()は書式つきの出力を行う組み込み関数です。文字列中の**%X**は、以降の対応する引数が持つ整数値を16進数で出力することを指示しています。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。

```
$a = "Hello ";
$b = $a . "World!"; // $b = "Hello World!"になる
```

●代入演算子

今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1-6　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。これをPHP4で実行すると期待通りの動作を行いますが、PHP3では文法エラーとなります。

```
<?
$a = "ABC¥n";
$b =& $a;
$b = "DEF¥n";
echo "¥$a=".$a."¥n";     //  “ $a=DEF ”を出力します(PHP4のみ)
?>
```

●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。

比較演算子を含む式は、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。

表 1-7 比較演算子

例	名前	式の値
$a == $b	等しい	$aが$bに等しい場合にTrue
$a === $b	等しい	$aが$bに等しく、かつ同じ型である場合にTrue（PHP4 のみ）
$a != $b	等しくない	$aが$bに等しくない場合にTrue
$a < $b	より少ない	$aが$bより少ない場合にTrue
$a > $b	より多い	$aが$bより多い場合にTrue
$a <= $b	より少ないか等しい	$aが$bより少ないか等しい場合にTrue
$a >= $b	より多いか等しい	$aが$bより多いか等しい場合にTrue

```
<?
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$firstの値がTrue（非ゼロ）の場合、二番目の部分式$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。

```
<?
echo "10", (10==10.0) ? "==" : "!=", "10.0 です。<BR>";
                        //  “10==10.0 です。”を出力します
?>
```

●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Trueまたは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 1-8 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Trueの場合にTrue
$a or $b	論理和	$aまたは$bのどちらかがTrueの場合にTrue
$a xor $b	排他的論理和	$aまたは$bのどちらかがTrueでかつ両方ともTrueでない場合にTrure
! $a	否定	$aがTrueでない場合にTrue
$a && $b	論理積	$aおよび$bがともにTrueの場合にTrue
$a || $b	論理和	$aまたは$bのどちらかがTrueの場合にTrue

●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。

```
$output = `ls -al`;
echo "<pre>$output</pre>";
```

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）。本書推奨の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nobody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nobody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、Apacheのエラーログ[*17]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。詳細は第3部の関数リファレンスを参照してください。

*17 本書のインストールに従えば、/usr/local/apache/logs/error_log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1-9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。

表 1-9　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

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。

```
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。ブロックの終わりがはっきりわかるので、慣れないうちはENDIF構文のほうがよいかもしれません。

```
IF ($visit_count==0):    // コロン(:)をつけることに注意(以下同様)
  echo 'はじめまして(^O^)'¥n;
ELSEIF ($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(;)をつけることに注意
```

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True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False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われます。

```
WHILE(1)   echo 'これは無限ループです';
```

条件つきループの例です。

```
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。

```
$i = 0;
WHILE($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。

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

●FOR

```
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実行され、主にループカウンタ(ループを実行する回数を数えるための変数)を初期化するのに使われます。そしてループの本体が評価され、ループを1回実行するごとに式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)を行うことが多いようです。そのあと式2がループの終了判定条件として評価され、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(終了条件)を空にすると暗黙のうちにTrue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、外側のネストした任意のループからの脱出を行うことができます。

```
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";
    }
}
```

⇩ 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。

```
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
}
```

⇩ 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。

```
switch ($button)    {
    case "登録":
        print "登録処理を行います";
        break;
    case "修正":
        print "修正処理を行います";
        break;
    case "削除":
        print "削除処理を行います";
        break;
    default:
        print "デフォルト処理を行います";
}
```

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ることになります。

● **FOREACH**

（PHP3ではなく）PHP4には、Perlやほかの言語とよく似たFOREACH構文があります。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。よって、次のループでは次の要素を処理することになります。

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。

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。

注意

FOREACHは、指定した配列自体に対してではなく、そのコピーに対して処理を行うことに注意してください。このためeach()のように、配列のポインタ位置が変更されることはありません。以下の文は機能的に等価です。

```
reset($arr);                                  // 配列の先頭を指す
WHILE(list( , $value) = each($arr)) {         // 値をひとつ取り出して$valueへ代入
    echo "Value: $value<br>¥n";
}

FOREACH($arr as $value) {
   echo "Value: $value<br>¥n";
}
```

以下の文も機能的に等価です。

```
reset($arr);                                  // 配列の先頭を指す
WHILE(list($key, $value) = each($arr)) {      // キーと値のペアを取り出す
   echo "Key: $key; Value: $value<br>¥n";
}

FOREACH($arr as $key => $value)    {
   echo "Key: $key; Value: $value<br>¥n";
}
```

●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、スクリプトの実行に先立って行われます。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。

```
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。

```
$files = array('first.inc', 'second.inc', 'third.inc');
for ($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（**<?**）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。

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。サンプルを見てください。

```
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;
        $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);
    }
}
```

```
$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
```

⇩ 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。
```

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。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的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、別の変数」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。

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（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）。

上記の例では、foo()、add()、change()でそれぞれメンバ変数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()** でメンバ関数の呼び出しを行います。

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定義（オーバライド）することによって、一部のメンバ関数の振る舞い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。

```
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();
$baz->add();
$baz->change(5);
$baz->change();                 //  デフォルト引数を使う
```

⇩ 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クラスでは定義していませんが、メインルーチンから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、指定された値の二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。

Hypertext Preprocessor

Chapter-4 PHPを使いこなそう

このセクションでは、第2章で紹介したサンプルに機能を追加し、多少は実用的なサンプルにしてみたいと思います。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。このセクションでは、以下のような肉づけを行ってみましょう。

- **デバッグルーチンの組み込み**
- **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**

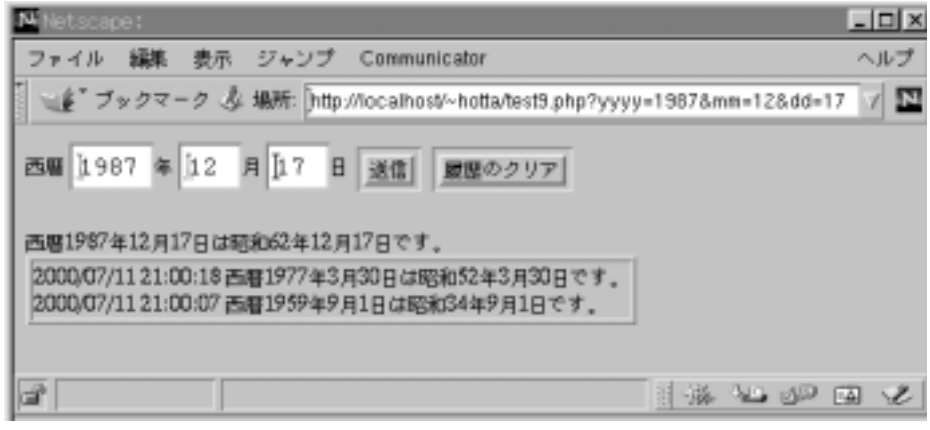
汎用性がありそうな関数（ほかのスクリプトでも使えそうなもの）は別ファイルにして、メインスクリプトからインクルードするようにしています。なお、実践的なスクリプトの例ということで、デバッグ文も実行に影響を与えない範囲であえて残しています。まずは**図1-15**の実行画面を見てください。

図1-15 初期画面

```
http://localhost/~hotta/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下の行に表示されます。さらに、それまでに入力した履歴が、入力した日付＋時刻つきで線で囲まれて表示されます。この履歴は**履歴のクリア**ボタンを押すと消去されます。不適切な入力を行うと、赤でエラーメッセージが表示されます。

図 1-16　実行中の画面



では、その実現方法を見ていきましょう。

4.1 デバッグルーチン

リスト 1-20　printd.inc

```
1 <?
2 //  デバッグ文の表示
3 function    printd($msg)
4 {
5     printf("<!-- %s -->¥n", $msg);
6 }
7 ?>
```

これは与えられた文字列を、HTMLのコメントの書式である**<!--**と**-->**で囲んで表示するだけのものです。たったこれだけのコードですが、プログラムのあちこちでこれを利用して変数のその時点の値などを表示してやると、出力に影響を与えないでプログラムの振る舞いを調べられるので便利です。デバッグ表示を見たい場合は、適宜「表示」－「ページのソース」を使って表示してやります。変数の中身をそのまま出力すると、ブラウザの画面に表示されるので一見デバッグしやすそうなのですが、デバッグ終了後にこれらのデバッグ文を取り除いてやらねばならず、間違いの元です。以下の処理ロジックの中でも、ここで定義したprintd()が頻繁に出てきます。

このように、ある機能を付加するためにできあいの処理を「くるむ」プログラムのことをラッパー（wrapper）プログラムと呼びます。

Column

assert機能

デバッグの手法のひとつとして、PHP4でサポートされたassert()について紹介しておきます。これはC言語のassert(3)と同等の機能で、プログラムのあちこちにassert()の引数として任意の評価式を埋め込んでおき、評価式が偽になったら警告を出したりプログラムの実行をそこで打ち切ったりするというものです。

リスト1-21 assert.php

```
<?
for ($i=0; $i<10; $i++) {
    printf("¥$i=$i<br>¥n");
    assert($i<5);
}
?>
```

⇩ 実行結果

```
$i=0
$i=1
$i=2
$i=3
$i=4
$i=5

Warning: Assertion failed in /home/hotta/public_html/assert.php on line 4
```

このプログラムは本来10回のループを行います。

しかしながらassert()の機能により、$i<5の評価式を満たさなくなったところで強制的に実行が打ち切られました。なお、assert()の動作はphp.iniファイルで制御されます。php.iniのassert.bailをOnにするとassert()条件を満たさなくなった時点でスクリプトの実行が打ち切られますが、Off（デフォルト）では警告を出すのみで、実行は継続されます。

assert()は画面出力を乱すので、場合によっていろいろ使い分けてください。デバッグが終了したらassert.activeをOff（デフォルトはOn）にすることで、assert()を無効にすることもできます。

4.2 日付チェック

日付チェックのルーチンは、次のようになりました。

リスト 1-22 date_is_valid.inc

```
 1 <?
 2 //  入力値のチェック
 3 //  返り値：True.妥当な入力
 4 function    date_is_valid($yyyy, $mm, $dd)
 5 {
 6     printd("¥$yyyy=$yyyy, ¥$mm=$mm, ¥$dd=$dd");
 7     if (! ereg("^[0-9][0-9]+$", $yyyy))     return  False;
 8     if (! ereg("^[0-9][0-9]*$", $mm))       return  False;
 9     if (! ereg("^[0-9][0-9]*$", $dd))       return  False;
10     printd("date_is_valid(): check1");
11     if ($mm < 1 || 12 < $mm)                return  False;
12     printd("date_is_valid(): check2");
13     if ($dd < 1 || 31 < $dd)                return  False;
14     printd("date_is_valid(): check3");
15     if ($dd==31)    {
16         if ($mm==2 || $mm==4 || $mm==6 || $mm==9 || $mm==11) {
17             printd("date_is_valid(): check4");
18             return  False;
19         }
20         return  True;
21     }
22     if ($dd==30)    {
23         if ($mm==2) {
24             printd("date_is_valid(): check5");
25             return  False;
26         }
27         return  True;
28     }
29     if ($mm==2 && $dd==29)  {
30         if ($yyyy % 4 == 0) {
31             if ($yyyy % 100 == 0)   {
32                 if ($yyyy % 400 == 0)   {
33                     return  True;
34                 }
35                 printd("date_is_valid(): check6");
36                 return  False;
37             }
38             return  True;
39         }
40         printd("date_is_valid(): check7");
41         return  False;
42     }
43     return  True;
44 }
45 ?>
```

この関数は引数として年月日を受け取り、その日付が有効な場合はTrue、そうでなければFalseを返します。Falseが返った場合にどこで引っかかったのかを調べるために、あちこちにprintd()を仕掛けています。なにが原因でエラーと判定したのかをユーザにわかるように、人間が読んでわかるメッセージを表示してやってもよいかもしれませんが、この例ではそこまではやっていません。

●7～9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。7行目の正規表現は「2個以上の数字の並び」、8～9行目は「1個ないし2個の数字」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。

●11～14行目

月と日の範囲チェックを行っています。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わずにすむので、処理速度の向上が図れ、間違いも少なくなります。

●15～21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Trueを返して関数を抜けています。

●22～28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。

●29～42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わないようにしています。考え方としては以下の通りです。

- **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**

●43行目

31日および30日以外でかつ2月29日以外なら、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。

4.3 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。やや汎用性に欠けますが、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。

リスト1-23 calc_wareki.inc

```
<?
//  元号の計算
//  返り値：計算結果の文字列
function    calc_wareki($yyyy, $mm, $dd)
{
    printd("¥$yyyy=$yyyy, ¥$mm=$mm, ¥$dd=$dd");
    $border = array(
        array(開始日 => 18680908, 終了日 => 19120730, 元号 => 明治),
        array(開始日 => 19120730, 終了日 => 19261225, 元号 => 大正),
        array(開始日 => 19261225, 終了日 => 19890107, 元号 => 昭和),
        array(開始日 => 19890107, 終了日 => 20091231, 元号 => 平成)
    );

    $target = sprintf("%04d%02d%02d", $yyyy, $mm, $dd);
    printd("¥$target=$target");

    if ($target < $border[0][開始日])    return  "計算対象外";
    printd("calc_wareki(): check1");
    for ($i=0; $border[$i]; $i++)  {
        printd("calc_wareki(): ¥$border[$i][元号]="
        . $border[$i][元号]);
        if ($border[$i][開始日] <= $target &&
            $target <= $border[$i][終了日]) {
            $temp = ($yyyy - substr($border[$i][開始日], 0, 4) + 1);
            $wareki = $border[$i][元号]
                    . ($temp == "1" ? "元" : $temp);
            break;
        }
    }
    if ($i > 3) return  "計算対象外";
    printd("calc_wareki(): check2");
    return  sprintf("%s年%d月%d日", $wareki, $mm, $dd);
}
?>
```

● 7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。

● 14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。

● 17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。

● 19～29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。

● 22～23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。

● 24行目

この元号に該当すれば、一時変数$tempに和暦の年を格納しています。substr()は文字列の一部を取り出す関数です。ここでは$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。

● 25～26行目

$warekiに答えを格納しています。$border[$i][元号]元号を取り出し、24行目の計算結果が1であれば「元年」となるようにしています。

● 27行目

ループを抜けます。

● 30行目

これに合致すると、どの元号にも属していない（すなわち2009年12月31日以降）ということなので、計算対象外とします。

● 32行目

答えを整形して、得られた文字列を呼び出し側に返します。

4.4 メインルーチン

前記以外の部分はあまり汎用性はなさそうなので、メインルーチンとしてひとつにまとめてしまいました。スクリプト名は**test9.php**です。

リスト1-24 test9.php

```
 1 <?
 2 session_start();                  //  セッションを開始／復帰
 3 if ($button)    {
 4     $hist = "";
 5     session_destroy();            //  セッションデータを破棄
 6 }
 7 session_register(hist);           //  セッションへ変数を登録
 8
 9 include("printd.inc");
10 include("date_is_valid.inc");
11 include("calc_wareki.inc");
12
13 //
14 //  入力画面の表示
15 //
16 function    display_input_area($yyyy, $mm, $dd, $hist)
17 {
18 ?>
19 <form action="test9.php?<?=SID?>">
20 西暦
21 <input type=text name=yyyy size=5 value="<? echo $yyyy; ?>">年
22 <input type=text name=mm   size=3 value="<? echo $mm; ?>">月
23 <input type=text name=dd   size=3 value="<? echo $dd; ?>">日
24 <input type=submit value="送信">
25 <input type=submit name=button value="履歴のクリア">
26 </form>
27 <?
28 }
29
30 //
31 //  メイン
32 //
33 printd("前回のセッション名=".session_name());
34 printd("前回のセッションID=".session_id());
35 printd("セッションモジュール名=".session_module_name());
36 printd("セッションデータ保存path名=".session_save_path());
```

```
37 printd("¥$histがセッションに登録されているか="
38     . (session_is_registered("hist") ? "True" : "False"));
39 printd("前回の¥$hist=".$hist);
40 display_input_area($yyyy, $mm, $dd, $hist);     //  入力エリアの表示
41 if (empty($yyyy) && empty($mm) && empty($dd))   exit;
42 if (date_is_valid($yyyy, $mm, $dd)) {           //  入力のチェック
43     $wareki = calc_wareki($yyyy, $mm, $dd);     //  元号の計算
44     $result = sprintf("西暦%d年%d月%d日は%sです。<br>¥n",
45         $yyyy, $mm, $dd, $wareki);
46     echo $result;                               //  結果の表示
47 } else  {
48     print("<font color=red>日付の入力が誤っています</font>¥n");
49 }
50 if ($hist)  {
51     print("<table border>¥n<tr><td>¥n$hist</table>¥n"); //  履歴の表示
52 }
53 if ($result)    {
54     $hist = date("Y/m/d H:i:s ") . $result . $hist;     //  履歴に追加
55 }
56 printd("今回の¥$hist=".$hist);
57 ?>
```

基本的な流れは第2章の**リスト1-19**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 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。これを消さずに毎回履歴として表示してやるためには、どこかに保存しておかねばなりません。保存しておく場所としては、以下の二つが考えられます。

- **クライアント内のどこか**
- **サーバ内のどこか**

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持ちまわ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処理のことをセッションと呼び、これを実現することを「セッション管理」と呼びます。実現方法としては、クライアントが持ち回る形式が一般的です。サーバに情報を保存する場合でも、保存した情報を取り出すためのキーになる情報自体はクライアント側に持たせてやらなければ、正確なセッション管理はできません。これは、Webのしくみだけではサーバ側がクライアントを一意に識別することが困難であるからです。

4.6 PHP4のセッション管理

PHP4ではセッション管理が言語レベルで実装されました。**test9.php**でもPHP4のセッション管理機能を利用しているので、**test9.php**はPHP4のみで実行可能です（PHP3では未サポート関数のエラーになります）。

PHP4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変数名で、デフォルトでは**PHPSESSID**です。セッションIDは、セッションが新たに開始されるごとにPHP4によって動的に割り当てられる識別子です。では**test9.php**を見てください。

● 2行目

session_start()で、PHP4に対してセッションを開始することを宣言しています。

● 3行目

$buttonが真（NULLでない）かを聞いています。この値は、25行目で表示している**履歴のクリア**ボタンを押すことによりセットされます。スクリプト側では履歴を$histという変数で管理していますが、履歴をクリアするためには、$histをクリアするとともに、session_destroy()関数によりセッションで保持している$histの中身をクリアしてやる必要があります。

● 7行目

セッションに変数を登録するのがsession_register()関数です。これ以降は$histを通常の変数として使用できます。またスクリプト終了時点の$histの内容は、自動的にセッション情報としてサーバ内部に保持されます。session_register()に渡すのは「変数名」であり、変数の値ではないので注意してください。

● **19行目**

この<FORM>タグでは**<?=SID?>**というちょっと見慣れない表現があります。これはPHP4においては

```
<? echo session_name() . "=" . session_id();?>
```

の短縮形です。このおまじない的な記述により、セッションIDをクライアント側に渡しています。

● **51行目**

$histの内容を<table>タグを使って罫線で囲んで表示しています。

● **54行目**

$histに今回表示した内容を追加しています。date()関数は、日付や時刻をいろいろなかたちで取り出す関数です。

セッション情報の実体は、PHP4のデフォルトでは**/tmp/セッションID**というファイルに保存されています。このファイルの中身は次のようになっています。パーミッションの関係で、root にならないと読めません。

```
hotta@star:~$ ls -l /tmp
合計 4
-rw-------   1 nobody   nobody  410 Jul 11 21:00 sess_d0c90571763696244103 21e04db1d3a3
hotta@star:~$ su -c "cat /tmp/sess_d0c90571763696244103 21e04db1d3a3"
Password:
hist|s:262:"2000/07/11 21:00:26 西暦1987年12月17日は昭和62年12月17日です。<br>
2000/07/11 21:00:18 西暦1977年3月30日は昭和52年3月30日です。<br>
2000/07/11 21:00:07 西暦1959年9月1日は昭和34年9月1日日です。<br>
";
```

先頭の**hist**が変数名で、管理情報のあとに変数の内容が二重引用符でくくられてそのままのかたちで保存されているのがわかると思います。ファイル名は**sess_セッションID**になっています。

なお参考までに、この時点で「ページのソース」の表示を行った、いわゆるデバッグ画面を以下に示します。こちらはページ幅の都合で適当に改行しています。適宜ソースや説明文と見比べてみてください。なお、セッションに関する設定値は**php.ini**ファイルで設定されています。詳細は第3部の関数リファレンスを参照してください。

図 1-17 「ページのソース」の実行画面

```
 1 <!-- 前回のセッション名=PHPSESSID -->
 2 <!-- 前回のセッションID=d0c90571763696244103 21e04db1d3a3 -->
 3 <!-- セッションモジュール名=files -->
 4 <!-- セッションデータ保存path名=/tmp -->
 5 <!-- $histがセッションに登録されているか=True -->
 6 <!-- 前回の$hist=
 7      2000/07/11 21:00:26 西暦1987年12月17日は昭和62年12月17日です。<br>
 8      2000/07/11 21:00:18 西暦1977年3月30日は昭和52年3月30日です。<br>
 9      2000/07/11 21:00:07 西暦1959年9月1日は昭和34年9月1日です。<br>
10 -->
11 <form action="test9.php?">
12 西暦
13 <input type=text name=yyyy size=5 value="1987">年
14 <input type=text name=mm   size=3 value="12">月
15 <input type=text name=dd   size=3 value="17">日
16 <input type=submit value="送信">
17 <input type=submit name=button value="履歴のクリア">
18 </form>
19 <!-- $yyyy=1987, $mm=12, $dd=17 -->
20 <!-- date_is_valid(): check1 -->
21 <!-- date_is_valid(): check2 -->
22 <!-- date_is_valid(): check3 -->
23 <!-- $yyyy=1987, $mm=12, $dd=17 -->
24 <!-- $target=19871217 -->
25 <!-- calc_wareki(): check1 -->
26 <!-- calc_wareki(): $border[0][元号]=明治 -->
27 <!-- calc_wareki(): $border[1][元号]=大正 -->
28 <!-- calc_wareki(): $border[2][元号]=昭和 -->
29 <!-- calc_wareki(): check2 -->
30 西暦1987年12月17日は昭和62年12月17日です。<br>
31 <table border>
32 <tr><td>
33 2000/07/11 21:00:18 西暦1977年3月30日は昭和52年3月30日です。<br>
34 2000/07/11 21:00:07 西暦1959年9月1日は昭和34年9月1日です。<br>
35 </table>
36 <!-- 今回の$hist=
37 2000/07/11 21:00:18 西暦1977年3月30日は昭和52年3月30日です。<br>
38 2000/07/11 21:00:07 西暦1959年9月1日は昭和34年9月1日です。<br>
39 -->
```

4.7 PHP3のセッション管理

PHP3ではセッション管理の概念がないので、プログラマが自前で管理し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。

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

いずれの手法においても、セッションの情報は主に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ではないかと思います。それでは順に説明します。なお、これらの手法は混在もできますし、もちろんPHP4でも使用可能です。

4.7.1 HTTPクッキーを使う

これは、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手法です。クッキーを使う場合は、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るので、**test9.php**のような使い方をする場合には、出力をバッファリング（貯めておくこと）するなど多少の工夫が必要です。クッキーはブラウザによってはサポートされていないこともあり、またブラウザの設定で無効にすることもできるので、クッキーを採用する際には十分検討してください。

HTTPクッキーを使うように**test9.php**を書き直してみました。なお、このスクリプトをPHP3で動かすには、拡張子を**php3**とする必要があります。

リスト1-25 test9-cookie.php

```
 1 <?
 2 include("date_is_valid.inc");
 3 include("calc_wareki.inc");
 4
 5 //  デバッグ文の表示
 6 function    printd($msg)
 7 {
 8 //  printf("<!-- %s -->¥n", $msg);
 9     syslog(LOG_INFO, sprintf("<!-- %s -->¥n", $msg));
10 }
11 //  入力画面の表示
```

```
12 //
13 function    display_input_area($yyyy, $mm, $dd, $hist)
14 {
15 ?>
16 <form action="test9-cookie.php">
17 西暦
18 <input type=text name=yyyy size=5 value="<? echo $yyyy; ?>">年
19 <input type=text name=mm   size=3 value="<? echo $mm; ?>">月
20 <input type=text name=dd   size=3 value="<? echo $dd; ?>">日
21 <input type=submit value="送信">
22 <input type=submit name=button value="履歴のクリア">
23 </form>
24 <?
25 }
26
27 //
28 //  メイン
29 //
30 if ($button)    {   $hist = ""; }
31 printd("¥$hist=$hist");
32 if ($yyyy && $mm && $dd)    {
33     if (date_is_valid($yyyy, $mm, $dd)) {           //  入力のチェック
34         $wareki = calc_wareki($yyyy, $mm, $dd);     //  元号の計算
35         $result = sprintf("西暦%d年%d月%d日は%sです。<br>¥n",
36             $yyyy, $mm, $dd, $wareki);
37         $errmsg = "";
38     } else  {
39         $result = "";
40         $errmsg = "<font color=red>日付の入力が誤っています</font>¥n";
41     }
42 }
43 if ($hist)  {
44     $history = "<table border>¥n<tr><td>¥n$hist</table>¥n";
45 }
46 if ($result)    {
47     $hist = date("Y/m/d H:i:s ") . $result . $hist; //  履歴に追加
48 }
49 SetCookie("hist", $hist, time()+3600);      //  クッキーの設定
50 display_input_area($yyyy, $mm, $dd, $hist); //  入力エリアの表示
51 echo $result;                               //  結果の表示
52 echo $errmsg;                               //  エラー表示
53 echo $history;                              //  履歴の表示
54 ?>
```

● **9行目**

デバッグ用のprintd()がそのままでは使えない（クッキーの出力に先立って、いかなる出力も行ってはならない）のでsyslog()を使うように書き直しました。デバッグ文の参照は以下のようにします[*18]。

```
hotta@star:~$ su -c "tail -f /var/log/messages"
Password:
Jul 11 23:43:08 star httpd: <!-- date_is_valid(): check3 -->
Jul 11 23:43:08 star httpd: <!-- $yyyy=1977, $mm=9, $dd=1 -->
Jul 11 23:43:08 star httpd: <!-- $target=19770901 -->
Jul 11 23:43:08 star httpd: <!-- calc_wareki(): check1 -->
Jul 11 23:43:08 star httpd: <!-- calc_wareki(): $border[0][元号]=明治 -->
Jul 11 23:43:08 star httpd: <!-- calc_wareki(): $border[1][元号]=大正 -->
Jul 11 23:43:08 star httpd: <!-- calc_wareki(): $border[2][元号]=昭和 -->
Jul 11 23:43:08 star httpd: <!-- calc_wareki(): check2 -->
```

これで CTRL + C で止めるまで、リアルタイムにデバッグメッセージが表示されます。

● **30行目**

「履歴のクリア」処理を実装しています。

● **49行目**

SetCookie()関数で、現在の$histの値をクライアント側にクッキー値として覚えさせています。このあと、このURLにアクセスしないままでいると、3600秒後（1時間後）に保存した情報は無効になります。

● **51～53行目**

いったん変数にためた出力データを、ここでまとめて出力しています。

なお、クッキーはクライアント側では以下のように保持されています。

```
hotta@star:~$ grep localhost .netscape/cookies
localhost       FALSE   /~hotta FALSE   963408142       hist  ⇨
 ⇨ 2000%2F07%2F12+21%3A22%3A22+%C0%BE%CE%F11959%C7%AF9%B7%EE1%C6%FC%A4 ⇨
 ⇨ %CF%BE%BC%CF%C234%C7%AF9%B7%EE1%C6%FC%A4%C7%A4%B9%A1%A3%3Cbr%3E%0A
```

（⇨編集の都合で折り返していますが、実際は1行です）

[*18] syslog()の引数およびsyslogファイルの表示方法は、Vine Linux 2.0の場合です。これ以外の環境では異なる場合もあります。詳細は使用しているOSの man 7 syslog を参照してください。

4.7.2 URLの引数として指定する

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。

この方法で書き直した**test9-url.php**を以下に示します。

リスト1-26 test9-url.php

```
 1 <?
 2 include("date_is_valid.inc");
 3 include("calc_wareki.inc");
 4
 5 //  デバッグ文の表示
 6 function    printd($msg)
 7 {
 8 //  printf("<!-- %s -->¥n", $msg);
 9     syslog(LOG_INFO, sprintf("<!-- %s -->¥n", $msg));
10 }
11 //  入力画面の表示
12 //
13 function    display_input_area($yyyy, $mm, $dd, $hist)
14 {
15 ?>
16 <form method="post" action="test9-url.php?hist=<?echo urlencode($hist)?>">
17 西暦
18 <input type=text name=yyyy size=5 value="<? echo $yyyy; ?>">年
19 <input type=text name=mm   size=3 value="<? echo $mm; ?>">月
20 <input type=text name=dd   size=3 value="<? echo $dd; ?>">日
21 <input type=submit value="送信">
22 <input type=submit name=button value="履歴のクリア">
23 </form>
24 <?
25 }
26
27 //
28 //  メイン
29 //
30 if ($button)    {   $hist = ""; }
31 printd("¥$hist=$hist");
32 $hist = urldecode($hist);
```

```
33 if ($yyyy && $mm && $dd)    {
34     if (date_is_valid($yyyy, $mm, $dd)) {           //  入力のチェック
35         $wareki = calc_wareki($yyyy, $mm, $dd);     //  元号の計算
36         $result = sprintf("西暦%d年%d月%d日は%sです。<br>¥n",
37             $yyyy, $mm, $dd, $wareki);
38         $errmsg = "";
39     } else  {
40         $result = "";
41         $errmsg = "<font color=red>日付の入力が誤っています</font>¥n";
42     }
43 }
44 if ($hist)  {
45     $history = "<table border>¥n<tr><td>¥n$hist</table>¥n";
46 }
47 if ($result)    {
48     $hist = date("Y/m/d H:i:s ") . $result . $hist; //  履歴に追加
49 }
50 display_input_area($yyyy, $mm, $dd, $hist); //  入力エリアの表示
51 echo $result;                                //  結果の表示
52 echo $errmsg;                                //  エラー表示
53 echo $history;                               //  履歴の表示
54 ?>
```

test9-cookie.phpとの違いがおわかりでしょうか？ 念のためdiff（ファイルの差分を出力するUNIXのコマンド、およびその出力）もつけておきます。行頭の**<**が旧（**test9-cookie.php**）ファイル、**>**が新（**test9-url.php**）ファイルの内容を表しています。数字は双方のファイルにおける行番号で、数字間のアルファベット1文字はそれぞれ**c**（hange）、**a**（ppend）、**d**（elete）という操作を表しています。

```
hotta@star:~/public_html$ diff test9-cookie.php test9-url.php
16c16
< <form action="test9-cookie.php">
---
> <form method="post" action="test9-url.php?hist=<?echo urlencode($hist)?>">
31a32
> $hist = urldecode($hist);
49d49
< SetCookie("hist", $hist, time()+3600);      //  クッキーの設定
```

● **16行目**

<FORM>タグのaction属性の中で、$histの値を引数として渡しています。urlencode()関数により、変数の内容の中でURL文字として問題がありそうな文字を別の文字に置き換えています。

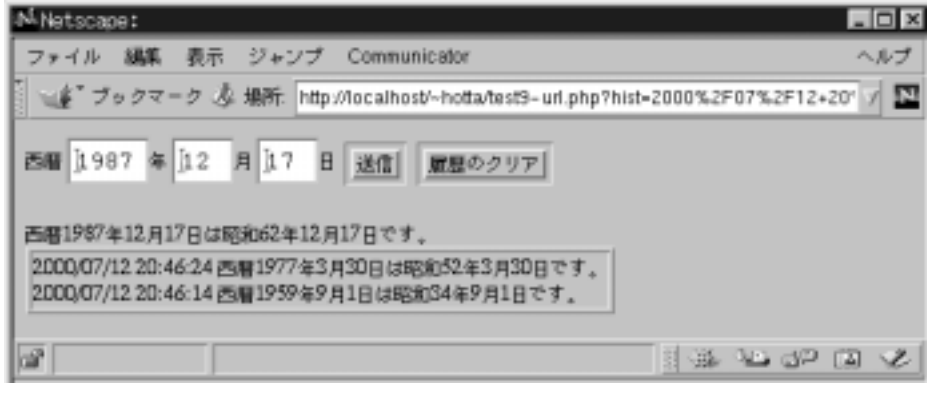
ところで、<FORM>タグにおけるサーバへの情報の引き渡し方法は、method属性をつけないとデフォルトのget（変数をURLで受け渡す）になります。しかしこのままでは18～22行目の<INPUT>タグによる情報が、$histの情報を上書きしてしまいます。これを避けるため、ここではmethodをpost（標準入力による受け渡し）に変更しています。<FORM>タグの詳細は「5.1.1　フォームの開始」を参照してください。

● **32行目**

サーバ側で受け取った$histはurlencode()されたものなので、urldecode()で復元して人間が読んでわかる形式にします。

test9-url.phpにおける実行画面を**図1-18**に示します。URLのうしろにエンコードされた文字列がついているのがわかると思います。

図1-18　test9-url.phpの実行画面



4.7.3　hidden属性を使う

これもブラウザに依存しない方法です。URLは変わらないので見た目はすっきりします。欠点としては、サーバ側では一切の情報の保持を行わないので、いったんブラウザを終了してしまうと、それらの変数を復元するすべがないことです。

hidden属性を使って書き直した**test9-hidden.php**を以下に示します。今度も違いがわかりにくいので、また**test9-cookie.php**とのdiffをつけておきます。

リスト1-27 test9-hidden.php

```
 1 <?
 2 include("date_is_valid.inc");
 3 include("calc_wareki.inc");
 4
 5 //  デバッグ文の表示
 6 function    printd($msg)
 7 {
 8 //  printf("<!-- %s -->¥n", $msg);
 9     syslog(LOG_INFO, sprintf("<!-- %s -->¥n", $msg));
10 }
11 //  入力画面の表示
12 //
13 function    display_input_area($yyyy, $mm, $dd, $hist)
14 {
15 ?>
16 <form action="test9-hidden.php">
17 西暦
18 <input type=text name=yyyy size=5 value="<? echo $yyyy; ?>">年
19 <input type=text name=mm   size=3 value="<? echo $mm; ?>">月
20 <input type=text name=dd   size=3 value="<? echo $dd; ?>">日
21 <input type=submit value="送信">
22 <input type=submit name=button value="履歴のクリア">
23 <input type=hidden name=hist value="<? echo $hist; ?>">
24 </form>
25 <?
26 }
27
28 //
29 //  メイン
30 //
31 if ($button)    {   $hist = ""; }
32 printd("¥$hist=$hist");
33 if ($yyyy && $mm && $dd)    {
34     if (date_is_valid($yyyy, $mm, $dd)) {           //  入力のチェック
35         $wareki = calc_wareki($yyyy, $mm, $dd);     //  元号の計算
36         $result = sprintf("西暦%d年%d月%d日は%sです。<br>¥n",
37             $yyyy, $mm, $dd, $wareki);
38         $errmsg = "";
39     } else  {
40         $result = "";
41         $errmsg = "<font color=red>日付の入力が誤っています</font>¥n";
42     }
43 }
44 if ($hist)  {
45     $history = "<table border>¥n<tr><td>¥n$hist</table>¥n";
46 }
47 if ($result)    {
48     $hist = date("Y/m/d H:i:s ") . $result . $hist; //  履歴に追加
49 }
50 display_input_area($yyyy, $mm, $dd, $hist); //  入力エリアの表示
51 echo $result;                               //  結果の表示
52 echo $errmsg;                               //  エラー表示
53 echo $history;                              //  履歴の表示
54 ?>
```

```
hotta@star:~/public_html$ diff test9-cookie.php test9-hidden.php
16c16
< <form action="test9-cookie.php">
---
> <form action="test9-hidden.php">
22a23
> <input type=hidden name=hist value="<? echo $hist; ?>">
49d49
< SetCookie("hist", $hist, time()+3600);      //  クッキーの設定
```

● **16行目**

呼び出すURLを変えました。

● **23行目**

これがhidden属性を使った隠しフィールドです。「ページのソース」は以下のようになります。

図1-19 **「ページのソース」の実行画面**

```
<form action="test9-hidden.php">
西暦
<input type=text name=yyyy size=5 value="1987">年
<input type=text name=mm   size=3 value="12">月
<input type=text name=dd   size=3 value="17">日
<input type=submit value="送信">
<input type=submit name=button value="履歴のクリア">
<input type=hidden name=hist value=" ⇨
 ⇨ 2000/07/12 21:39:32 西暦1987年12月17日は昭和62年12月17日です。<br>  ⇨
 ⇨ 2000/07/12 21:39:19 西暦1977年3月30日は昭和52年3月30日です。<br>  ⇨
 ⇨ 2000/07/12 21:39:11 西暦1959年9月1日は昭和34年9月1日です。<br>
">
</form>
西暦1987年12月17日は昭和62年12月17日です。<br>
<table border>
<tr><td>
2000/07/12 21:39:19 西暦1977年3月30日は昭和52年3月30日です。<br>
2000/07/12 21:39:11 西暦1959年9月1日は昭和34年9月1日です。<br>
</table>
```

（⇨ hiddenの行は編集の都合で折り返していますが、実際は1行です）

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。

```
<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>
```

5.1.1 フォームの開始

構文
```
<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式を指定します。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**を指定して、任意のメールアドレスにメールとして送信することもできます。mailtoによるメール送信はクライアントの機能として行われるので、Webサーバ側は関与しません。

5.1.2 単一行テキスト

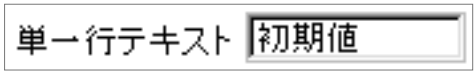
構文
```
<input type={text|password} name=変数名
       [size=入力欄の文字数]
       [maxlength=入力可能最大文字数]
       [value="デフォルト文字列"] >
```

単一行フィールド（**図1-20**）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 1-20　実行結果



5.1.3　複数行テキスト

構文

```
<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1-21**）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指定を行うと、入力フィールドの大きさに合わせて自動改行を行います。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 1-21 実行結果

5.1.4 ラジオボタン

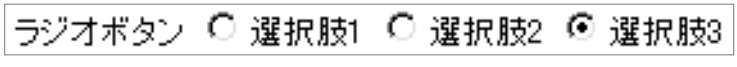
構文 `<input type=radio name=変数名 value=文字列 [checked]>`

選択可能なラジオボタン（**図1-22**）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 1-22 実行結果



5.1.5 チェックボックス

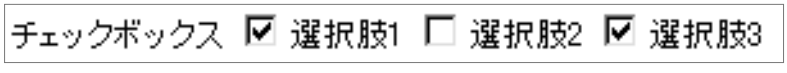
構文 `<input type=checkbox name=変数名 [value=文字列] [checked]>`

選択可能なチェックボックス（**図1-23**）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=check1 checked>選択肢1
<input type=checkbox name=check2>選択肢2
<input type=checkbox name=check3 checked>選択肢3
</form>
```

⇩

図1-23 実行結果



PHPの文法における特徴として、**変数名[]**という式に対して代入を行うと、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[]**という名前をつけておき、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うことができます。上記の例を配列を利用して書くと、以下のようになります。

```
<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>
```

5.1.6 隠しフィールド

構文 `<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。使い方は「4.7.3　hidden属性を使う」を参照してください。

5.1.7 プルダウンメニュー

構文
```
<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1-24**）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1-24　実行結果

5.1.8 リストボックス

構文
```
<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1-25**）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1-25 実行結果

5.1.9 リセットボタン

構文
```
<input type=reset [value=ボタン名]>
```

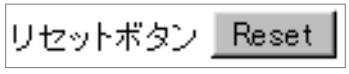
リセットボタン（**図1-26**）を表示します。リセットボタンはそのフォームに入力中のデータをすべて破棄（クリア）するためのもので、サーバの動作には影響を与えません。valueを指定しなければ、NetscapeCommunicator（以下NC）なら**Reset**、Internet Explorer（以下IE）なら**リセット**というボタンになります[*19]。

*19 Windows版のブラウザについての表示結果です。それ以外のOSでは異なる場合があります。

```
<form>
リセットボタン
<input type=reset>
</form>
```

⇩

図 1-26 実行結果



5.1.10 送信ボタン

構文 `<input type=submit [name=変数名] [value=ボタン名]>`

送信ボタン（**図1-27**）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。nameは押されたボタンを識別するためのものなので、送信ボタンがひとつしかない場合は指定する意味はありません（送信ボタンが押されたことは、<FORM>タグのaction属性で指定されたURLに制御が移ることでわかります）。valueを指定しなければ、NCなら**Submit Query**、IEなら**クエリ送信**というボタンになります[*18]。

```
<form>
送信ボタン
<input type=submit>
</form>
```

⇩

図 1-27 実行結果

5.1.11 ファイルのアップロード

構文

```
<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実現します。**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**

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ってやる必要があります。なお、一時ファイルを保存するディレクトリ名は、環境変数TMPDIRを設定することにより変更することができます。

- **$変数名_NAME**

送信元のシステムにおけるオリジナルのファイル名 。

- **$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）。

- **$変数名_TYPE**

ファイルのMIMEタイプ。ただしブラウザはこの情報を提供するとは限りません。提供された場合、**image/gif**といった値が入ります。

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。

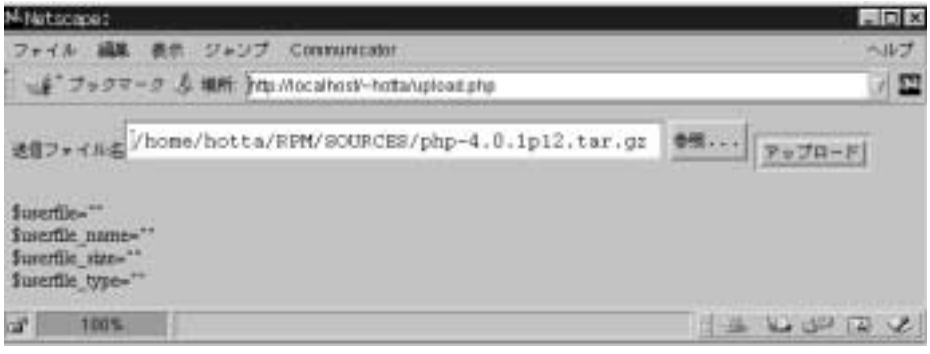
リスト1-28 upload.php

```
<form enctype="multipart/form-data" action="upload.php" method=post>
<input type=hidden name=MAX_FILE_SIZE value=10000000>
送信ファイル名<input name=userfile type=file size=60>
<input type=submit value=アップロード>
</form>
<?
echo "¥$userfile=¥"$userfile¥"<BR>";
echo "¥$userfile_name=¥"$userfile_name¥"<BR>";
echo "¥$userfile_size=¥"$userfile_size¥"<BR>";
echo "¥$userfile_type=¥"$userfile_type¥"<BR>";
system("/bin/ls -l /tmp/php*");
?>
```

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1-28**は、ブラウザのダイアログでファイルを選択したところ（まだ送信していない）です。

⇩

図1-28 実行結果



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1-29**の例ではlocalhostに対して**php-4.0.1pl2.tar.gz**というファイルを送信したところを示しています。

system()関数でUNIX OSにおける**/bin/ls**コマンドを実行することにより、アップロードされたファイルが**php**をプリフィックスとするファイル名で一時的にディスクに書き込まれることがわかります。スクリプト実行後にシェルからlsしても、そのファイルは消えてなくなってしまっています。

図 1-29 実行結果

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に限っていえば、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1-10**に示します。

表 1-10 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
mailto	電子メールアドレス
ftp	FTP（File Transfer Protocol）
file	ホスト固有のファイル名
jdbc	Java DataBase Connection データベースオブジェクト

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://サーバ名/]ディレクトリ名/]ファイル名.拡張子
[?変数名=値[&変数名=値]...]
```

URLに**?変数名＝値**オプションをつけてPHPスクリプトを呼び出した場合、PHP内部ではその変数名の前に**$**をつけるだけで自由に参照、変更することができます。**?**以降の部分のことを特に「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1-30 HTTPの流れ

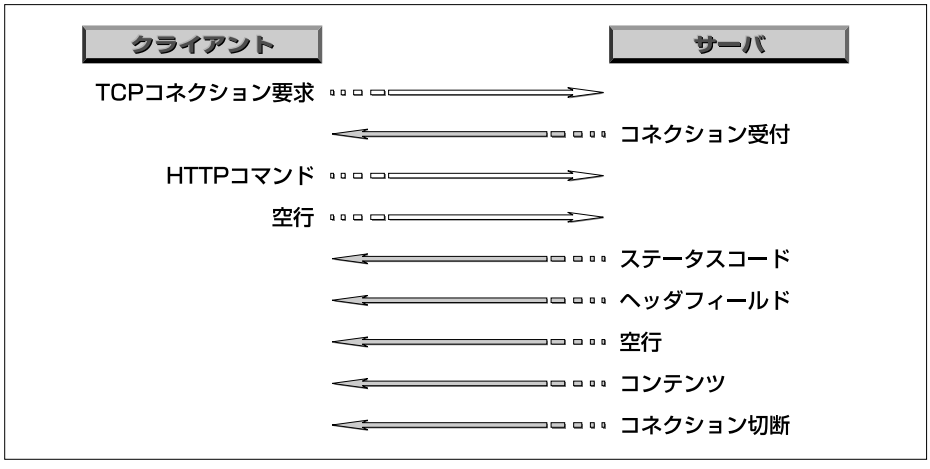


図1-30にHTTPの流れを示します。まずクライアントがWebサーバ（httpd）に対してTCPコネクションの接続を行います[*20]。コネクションが確立すると、クライアントはHTTPのコマンドおよびひとつの空行をサーバへ送ります。サーバはそれに対してステータスコードと呼ばれる行を返し、それに続いてヘッダフィールドと呼ばれる複数の行を返します。そして空行を送ることによりメタ情報（管理情報）の終わりを示し、それに続いて実際のコンテンツを返します。コンテンツを送り終えるとサーバはTCPコネクションを切断し、ひとつのHTTPセッションが終了します[*21]。

***20**
本来HTTPはTCPに依存したプロトコルではありませんが、ここではわかりやすくするためにこのような構成をとっています。

***21**
Apacheの場合、デフォルトではコンテンツを送り終えたあとも接続を持続した状態になっています。これにより、クライアントが続けて別のコンテンツを要求しても、新たなHTTPセッション（＝新しいhttpdの起動）が発生しないので応答速度が速くなり、またサーバマシンの負荷も軽減されます。この機能はKeepAliveと呼ばれています。

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（を含むファイル）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/1.0
```

GETコマンドを使ってCGIに引数を渡す場合、「5.2　URL」のところで説明したように、URLのうしろに**?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があるのは、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1-11**に示すカテゴリに分類されます。

表1-11　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
リダイレクト	300～399	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。これ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1-12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。Apacheのソースコードの中にある**apache_1.x.x/src/include/httpd.h**には、これ以外にもアプリケーションレベルで定義しているステータスコードが数多く定義されています。またApacheの設定により、ステータスコードに対応した特定の処理（特定のエラーメッセージを含む見栄えのよいコンテンツを返す、など）を定義することもできます。

表1-12　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。「4.7.1　HTTPクッキーを使う」で紹介したSetCookie()関数も、特定のヘッダフィールドを送信する関数です。

表1-13　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

5.3.4 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1-13**に示します。

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1-31**）。これはApacheで用意されたindex.htmlをtelnetコマンドを使ってロードしたところです。

図1-31　telnetでWebサーバに接続したところ

```
 1 hotta@star:~$ telnet localhost 80 Enter
 2 Trying 127.0.0.1...
 3 Connected to localhost.localdomain.
 4 Escape character is '^]'.
 5 GET / HTTP/1.0 Enter
 6 Enter
 7 HTTP/1.1 200 OK
 8 Date: Thu, 13 Jul 2000 13:16:10 GMT
 9 Server: Apache/1.3.12 (Unix) PHP/4.0.0
10 Content-Location: index.html.en
11 Vary: negotiate,accept-language,accept-charset
12 TCN: choice
13 Last-Modified: Sat, 20 Nov 1999 21:29:40 GMT
14 ETag: "1ff53-54e-383712c4;39667a6f"
15 Accept-Ranges: bytes
16 Content-Length: 1358
17 Connection: close
18 Content-Type: text/html
19 Content-Language: en
20 Expires: Thu, 13 Jul 2000 13:16:10 GMT
21
22 <!DOCtype HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
23 <HTML>
24  <HEAD>
25   <TITLE>Test Page for Apache Installation</TITLE>
26  </HEAD>
27
28 (中略)
29
30 <DIV ALIGN="CENTER"><IMG SRC="apache_pb.gif" ALT=""></DIV>
31 </BODY>
32 </HTML>
33 Connection closed by foreign host.
```

● **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（DocumentRoot[*22]）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（DirectoryIndex[*21]）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。

● **8～20行目**

サーバが返したヘッダフィールドです。

● **21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。

● **22～32行目**

コンテンツ本体。

● **33行目**

コンテンツを送り終えると、サーバはコネクションを切断します。

*22
Apache Web Serverの設定ファイルhttpd.confで指定できるディレクティブ（設定用キーワード）の一種です。

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充実しているので、C言語でコーディングすることに比べればはるかに楽ですが、PHPはさらに初心者でもとっつきやすくなっています。

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の処理系内部の話になります。PHPのコマンドライン版を使用する場合はまさにCGIそのものなので、以下のことがそのまま当てはまります。

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

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1-14**に示します。

表1-14 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

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。

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。

ここで、組み込み関数phpinfo()を呼び出すだけのスクリプトinfo.php3を実行してみます。スクリプトの内容は

```
<? phpinfo(); ?>
```

の1行だけです。これを

```
http://localhost/~hotta/info.php
```

として実行すると、**図1-32**のような大量の出力が行われます。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1-32 phpinfo()の出力

PHP Version 4.0.0

System	Linux star.1Bsoftware.co.jp 2.2.14-1vl6 #1 Tue Mar 14 09:55:12 JST 2000 i586 unknown
Build Date	Jul 8 2000
Configure Command	'./configure' '--with-apxs=/usr/local/apache/bin/apxs' '--enable-versioning' '--enable-track-vars' '--without-gd'
Server API	Apache
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/lib
ZEND_DEBUG	enabled
Thread Safety	disabled

This program makes use of the Zend scripting language engine:
Zend Engine v0.99, Copyright (c) 1998-2000 Zend Technologies Ltd.

PHP Core

Directive	Local Value	Master Value
define_syslog_variables	Off	Off
highlight.bg	#FFFFFF	#FFFFFF
highlight.comment	#FF8000	#FF8000
highlight.default	#0000BB	#0000BB
highlight.html	#000000	#000000
highlight.keyword	#007700	#007700
highlight.string	#DD0000	#DD0000
allow_call_time_pass_reference	On	On
asp_tags	Off	Off
display_errors	On	On
enable_dl	On	On
error_append_string	Off	Off
error_prepend_string	Off	Off
expose_php	On	On
ignore_user_abort	On	On
implicit_flush	Off	Off
log_errors	Off	Off
magic_quotes_gpc	On	On
magic_quotes_runtime	Off	Off
magic_quotes_sybase	Off	Off
output_buffering	Off	Off
register_argc_argv	On	On
register_globals	On	On
safe_mode	Off	Off
short_open_tag	On	On
sql.safe_mode	Off	Off
track_errors	Off	Off
track_vars	On	On
y2k_compliance	Off	Off
arg_separator	&	&
auto_append_file	*no value*	*no value*
auto_prepend_file	*no value*	*no value*
doc_root	*no value*	*no value*
default_charset	*no value*	*no value*
default_mimetype	text/html	text/html
error_log	*no value*	*no value*

extension_dir	/usr/local/lib/php/extensions/debug-non-zts-20000401	/usr/local/lib/php/extensions/debug-non-zts-20000401
gpc_order	GPC	GPC
include_path	*no value*	*no value*
max_execution_time	30	30
open_basedir	*no value*	*no value*
safe_mode_exec_dir	1	1
upload_max_filesize	2097152	2097152
upload_tmp_dir	*no value*	*no value*
user_dir	*no value*	*no value*
variables_order	*no value*	*no value*
SMTP	localhost	localhost
browscap	*no value*	*no value*
error_reporting	*no value*	*no value*
precision	14	14
sendmail_from	*no value*	*no value*
sendmail_path	/usr/sbin/sendmail -t	/usr/sbin/sendmail -t

xml

XML Support	active

standard

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

Directive	Local Value	Master Value
safe_mode_protected_env_vars	LD_LIBRARY_PATH	LD_LIBRARY_PATH
safe_mode_allowed_env_vars	PHP_	PHP_
assert.active	1	1
assert.bail	0	0
assert.warning	1	1
assert.callback	*no value*	*no value*
assert.quiet_eval	0	0

posix

Revision	$Revision: 1.14 $

pcre

PCRE (Perl Compatible Regular Expressions) Support	enabled
PCRE Library Version	3.1 09-Feb-2000

session

Session Support	enabled

Directive	Local Value	Master Value
session.save_path	/tmp	/tmp
session.name	PHPSESSID	PHPSESSID
session.save_handler	files	files
session.auto_start	0	0
session.gc_probability	1	1
session.gc_maxlifetime	1440	1440
session.serialize_handler	php	php
session.cookie_lifetime	0	0
session.cookie_path	/	/
session.cookie_domain	*no value*	*no value*
session.use_cookies	1	1
session.referer_check	*no value*	*no value*
session.entropy_file	*no value*	*no value*
session.entropy_length	0	0
session.cache_limiter	nocache	nocache
session.cache_expire	180	180

mysql

MySQL Support	enabled
Active Persistent Links	0
Active Links	0
Client API version	3.23.10-alpha
MYSQL_INCLUDE	
MYSQL_LFLAGS	
MYSQL_LIBS	

Directive	Local Value	Master Value
mysql.allow_persistent	On	On
mysql.max_persistent	Unlimited	Unlimited
mysql.max_links	Unlimited	Unlimited
mysql.default_host	*no value*	*no value*
mysql.default_user	*no value*	*no value*
mysql.default_password	*no value*	*no value*
mysql.default_port	*no value*	*no value*

db

This is GDBM version 1.8.0, as of May 19, 1999.

apache

APACHE_INCLUDE	
APACHE_TARGET	
Apache Version	Apache/1.3.12
Apache Release	10312100
Apache API Version	19990320
Hostname:Port	star.1Bsoftware.co.jp:80
User/Group	nobody(99)/99
Max Requests	Per Child: 0 Keep Alive: on Max Per Connection: 100
Timeouts	Connection: 300 Keep-Alive: 15
Server Root	/usr/local/apache
Loaded Modules	mod_php4, mod_setenvif, mod_so, mod_auth, mod_access, mod_alias, mod_userdir, mod_actions, mod_imap, mod_asis, mod_cgi, mod_dir, mod_autoindex, mod_include, mod_status, mod_negotiation, mod_mime, mod_log_config, mod_env, http_core

Apache Environment

Variable	Value
DOCUMENT_ROOT	/usr/local/apache/htdocs
HTTP_ACCEPT	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint, */*
HTTP_ACCEPT_ENCODING	gzip, deflate
HTTP_ACCEPT_LANGUAGE	ja
HTTP_CONNECTION	Keep-Alive
HTTP_HOST	star
HTTP_USER_AGENT	Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)
PATH	/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/usr/bin/X11:/usr/X11R6/bin:/usr/ucb/bin:/usr/local/bin:/usr/bin/mh:/root/bin
REMOTE_ADDR	192.168.0.3
REMOTE_PORT	1027
SCRIPT_FILENAME	/home/kota/public_html/db.php
SERVER_ADDR	192.168.0.220
SERVER_ADMIN	root@star.1Bsoftware.co.jp
SERVER_NAME	star.1Bsoftware.co.jp
SERVER_PORT	80
SERVER_SIGNATURE	Apache/1.3.12 Server at star.1Bsoftware.co.jp Port 80
SERVER_SOFTWARE	Apache/1.3.12 (Unix) PHP/4.0.0
GATEWAY_INTERFACE	CGI/1.1
SERVER_PROTOCOL	HTTP/1.1
REQUEST_METHOD	GET
QUERY_STRING	
REQUEST_URI	/~kota/db.php
SCRIPT_NAME	/~kota/db.php

HTTP Headers Information

HTTP Request Headers	
HTTP Request	GET /~hotta/info.php HTTP/1.1
Accept	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint, */*
Accept-Encoding	gzip, deflate
Accept-Language	ja
Connection	Keep-Alive
Host	star
User-Agent	Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)
HTTP Response Headers	
X-Powered-By	PHP/4.0.0
Keep-Alive	timeout=15, max=98
Connection	Keep-Alive
Transfer-Encoding	chunked
Content-Type	text/html

PHP Variables

Variable	
PHP_SELF	
HTTP_SERVER_VARS ["PATH_TRANSLATED"]	/home/hotta/public_html/info.php
HTTP_SERVER_VARS ["PHP_SELF"]	/~hotta/info.php
HTTP_SERVER_VARS ["argv"]	Array ( )
HTTP_SERVER_VARS ["argc"]	0
HTTP_ENV_VARS ["LESSOPEN"]	|lesspipe.sh %s
HTTP_ENV_VARS ["USERNAME"]	root
HTTP_ENV_VARS ["CANNA_SERVER"]	localhost
HTTP_ENV_VARS ["ENV"]	/root/.bashrc
HTTP_ENV_VARS ["HISTSIZE"]	1000

HTTP_ENV_VARS["XIM_PROG"]	canna
HTTP_ENV_VARS["HOSTNAME"]	star.18software.co.jp
HTTP_ENV_VARS["LOGNAME"]	root
HTTP_ENV_VARS["WNN6_FRONT_END"]	kinput2
HTTP_ENV_VARS["HISTFILESIZE"]	1000
HTTP_ENV_VARS["EMACS_IME"]	canna
HTTP_ENV_VARS["MAIL"]	/var/spool/mail/root
HTTP_ENV_VARS["TERM"]	kterm
HTTP_ENV_VARS["HOSTTYPE"]	i386
HTTP_ENV_VARS["PATH"]	/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/usr/bin/X11:/usr/X11R6/bi
HTTP_ENV_VARS["HOME"]	/root
HTTP_ENV_VARS["JSERVER"]	localhost
HTTP_ENV_VARS["INPUTRC"]	/etc/inputrc
HTTP_ENV_VARS["SHELL"]	/bin/bash
HTTP_ENV_VARS["NEXINIT"]	set autodetect=jp+ fileencoding=euc-jp displayencoding=euc-jp inputencoding=eu
HTTP_ENV_VARS["USER"]	root
HTTP_ENV_VARS["LC_ALL"]	ja_JP.ujis
HTTP_ENV_VARS["LANG"]	ja_JP.ujis
HTTP_ENV_VARS["OSTYPE"]	Linux
HTTP_ENV_VARS["WNN6_SERVER"]	localhost
HTTP_ENV_VARS["SHLVL"]	2
HTTP_ENV_VARS["LS_COLORS"]	no=00:fi=00:di=01;34:ln=01;36:pi=40;33:so=01;35:bd=40;33;01:cd=40;33;01:or=0
HTTP_ENV_VARS["_"]	/usr/local/apache/bin/httpd

Additional Modules

Environment

Variable	
LESSOPEN	
USERNAME	
CANNA_SERVER	
ENV	
HISTSIZE	
XIM_PROG	
HOSTNAME	
LOGNAME	
WNN6_FRONT_END	
HISTFILESIZE	
EMACS_IME	
MAIL	
TERM	
HOSTTYPE	
PATH	
HOME	
JSERVER	
INPUTRC	
SHELL	
NEXINIT	
USER	
LC_ALL	
LANG	
OSTYPE	
WNN6_SERVER	
SHLVL	
LS_COLORS	no=00:fi=00:di=01;34:ln=01;36:pi=40;33:so=01;35:bd=40;33;01:cd=40;33;01:or=01
_	

PHP License

This program is free software; you can redistribute it and/or modify it under the terms of the PHP License as published by the PHP Group and included in the distribution in the file: LICENSE

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

If you did not receive a copy of the PHP license, or have any questions about PHP licensing, please contact license@php.net.

5.5 データベースとの連携

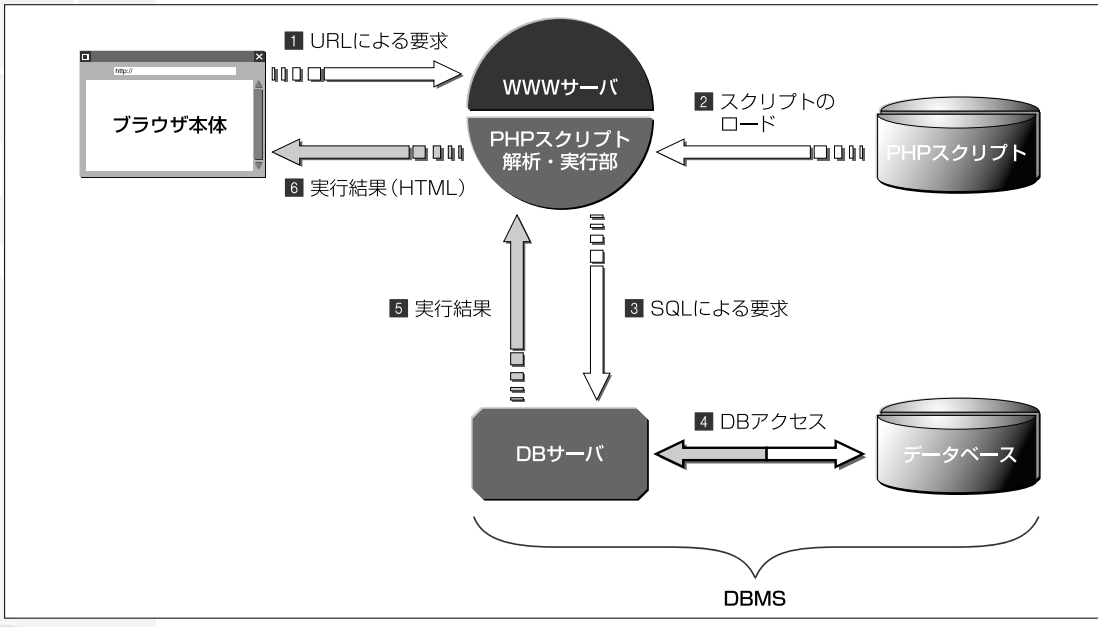
PHPには多くのDBMS（データベース・マネージメント・システム）との連携機能が組み込まれています。PHPがサポートするDBMSには、主に以下のものがあります。

dBase	PostgreSQL	Sybase
FilePro	mSQL	UNIX dbm
Informix	MySQL	ODBC
InterBase	Oracle	

SybaseインタフェースによるMicrosoft SQL Serverのサポートもあります。またODBC（Open DataBase Connectivity）は正確にはDBMSではなく、DBMSとそのほかのアプリケーションを接続するためのミドルウェアです。有名どころではODBC経由でIBM DB2がサポートされています。

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1-33**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

図1-33　PHPとDBMSとの連携



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。

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となってきます。ここでは初学の人のために、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です。ただしここでいうところの狭い意味でのデータベースとはリレーショナル（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するものです。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。表計算では、マス目のひとつひとつはセルと呼ばれます。

5.5.2 DBMSの一般的な構成

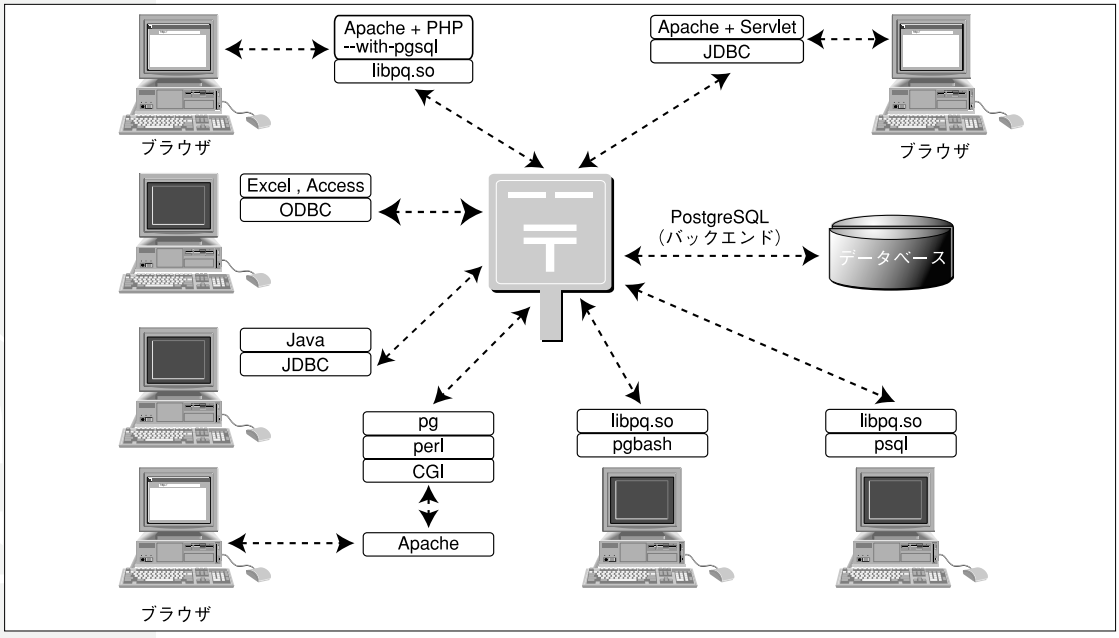
表計算ソフトではカーソルが表示され、縦横自由にカーソル（＝操作対象のセル）を動かすことができます。しかし、PostgreSQLをはじめとするDBMSの場合、各データの操作という基本的な部分のみの機能だけを提供しており、実際に取り出したデータを表示するなどのいわゆるユーザインタフェースの部分は外出しにするのが普通です。つまり実際にディスク上にあるデータを取り出したり更新したりという作業はデータベースサーバ（バックエンド）が行い、GUIなど人間が見たり入力したりする部分（フロントエンド）は別のプログラムが担当します。たとえばODBC（Open DataBase Connectivity）というミドルウェアを使えば、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1-34**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[*23]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。

*23
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（これを抽象化と呼ぶ）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 1-34 PostgreSQLの各種API



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。なお、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部で説明がありますので、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

```
CRETAE USER nobody;
```

nobodyは、Apacheが起動する際のデフォルトのユーザ名です。PHP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nobodyとして認証が行われます。

●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 nobody;
```

アクセス権の設定は各テーブルごとに行います。

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nobody権限で実行されます。

●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。なお、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

```
<?
    $conn = pg_connect("dbname=dbtest");    //  サーバへの接続
    $sql = "DELETE FROM tbltest";           //  全レコード削除依頼
    $result = pg_exec($sql);                //  SQL文の実行依頼
    for ($i=0; $i<5; $i++)  {
        $sql = sprintf("INSERT INTO tbltest VALUES('%d', 'TEST%02d')",
                $i, $i);                    //  レコード追加依頼
        $result = pg_exec($sql);            //  SQL文の実行依頼
    }
    $sql = "select * FROM tbltest";         //  レコード呼び出し依頼
    $result = pg_exec($sql);                //  SQL文の実行依頼
    $numrows = pg_numrows($result);         //  行数（レコード数）を取得

    for ($i=0; $i<$numrows; $i++)   {
        $id = pg_result($result, $i, 'id');     //  idの取り出し
        $name = pg_result($result, $i, 'name'); //  nameの取り出し
        printf("%d 行目：¥$id=$id, ¥$name=$name<br>¥n",
            $i, $id, $name);
    }
    pg_freeresult($result);                 //  一時領域を解放
    pg_close($conn);                        //  切断
?>
```

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～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としては例（前頁）のような感じになります。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が存在するものとします。エラーチェックなども省略しています。

これを実行すると、以下のような出力が得られます。

```
0 行目：$id=0, $name=TEST00
1 行目：$id=1, $name=TEST01
2 行目：$id=2, $name=TEST02
3 行目：$id=3, $name=TEST03
4 行目：$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

データベースへの接続を行います。この際データベース名を指定します。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。

●5～9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。

●10～11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。

●12行目

受信したレコードの行数（SELECTした結果が何件あったのか）を取得します。

● **13～18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_result()にループカウンタの$iと、項目名を渡していることに注意してください。つまりpg_result()を使用する場合は「受信したレコードのうちのXX番目のレコードのYYという項目の値」という指定方法で1項目ずつ取り出します。項目名のところは「何番目の項目」という指定のしかたもできます。pg_result()の動作は、表計算ソフトでセル間をカーソルで移動するのと同等の処理だと考えることができるでしょう。

● **19行目**

受信した全レコードのための一時メモリ領域を解放します。SELECTした結果はすべて破棄されます。

● **20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1-35**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1-35　DBMS連携における処理の流れ