



# PHP4の概略と PostgreSQLとの連携について

小山哲志

(株)ビート・クラフト

koyama@beatcraft.com



# 本日の内容

- PHPの概略
  - 基本的な文法
  - PHP3とPHP4の違い
- PHP4できちんと日本語を扱う方法
- PEAR DBクラスの使い方
- 抽象テーブルクラスの説明
- おまけ



# PHPとは

- 主にWebをターゲットにしたスクリプト言語
  - PHP: Hypertext Preprocessor
  - HTMLへ埋め込み
  - Cに似た文法
  - Apacheのモジュールとして動作
  - 各種ライブラリを拡張機能として使える
  - 最近クライアントでも使用される、汎用スクリプトへの道へ



# まずはお約束の歴史から

<http://www.php.net/manual/phpfi2.php>

## ● PHP/FI以前

- Perlで書かれたcgiのラッパー
  - Rasmus Lerdorf氏が作成
- 上記をCで書き直し
- Personal Home Page Toolsとしてリリース
- Personal Home Page Construction Kit
- 別に作成していた FI (Form Interpreter)と統合
- PHP/FI 2.0



## 歴史(2)

<http://www.zend.com/zend/founders.php>

- PHP3リリース
  - Andi Gutmans氏, Zeev Suraski氏
  - 日本人による国際化バージョン
- PHP4リリース
  - Zend社のスクリプトエンジン



# 文法の基本

<http://www.php.net/tut.php>

```
<html>
<head>
  <title>PHP test</title>
</head>
<body>
  <?php echo "Hello World<p>"; ?>
</body>
</html>
```



# 変数(1)

- \$をつけると変数

- あらかじめ定義されてる変数

- <?php echo \$HTTP\_USER\_AGENT; ?>

- Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)

- <?php phpinfo( ); ?>



# 条件文

```
<?php
if (strstr($_HTTP_USER_AGENT, "MSIE")) {
    echo "You are using Internet Explorer<br>";
}
else {
    echo "You are not using Internet Explorer<br>";
}
?>
```



# フォームによる値の送信

```
<form action="action.php" method="post">  
Your name: <input type="text" name="name">  
Your age: <input type="text" name="age">  
<input type="submit">  
</form>
```



Hi <?php echo \$name; ?>. You are <?php echo \$age; ?> years old.



## 変数(2)

- 変数の型は動的に判断される  
(0 == "") -> true
- 型と値の両方を判断もできる  
(0 === "") -> false
- 未定義な変数はその場で定義  
\$hoge = 1;  
echo \$hgoe;



## 変数(3)

- localスコープでは、そのままではglobalにアクセスできない

```
function func() {  
    global $hoge;  
    echo $hoge;  
}  
$hoge = "funya";  
func();
```



# 配列(1)

## ● 通常の配列と連想配列

- 実は中身はいっしょ

```
$ar = array( "a" => "A", "b" => "B" );
```

```
$ar[ ] = "C";
```

```
$ar[ ] = "D";
```

```
print_r( $ar );
```

**Array**

(

**[a] => A**

**[b] => B**

**[0] => C**

**[1] => D**

)



## 配列(2)

- 内部に参照ポイントを持つ

```
reset( $ar );
```

```
while (list($key, $val) = each($ar)) {  
    echo "$key: $val¥n";  
}
```

- reset() を忘れるとかなり危険



## 配列(3)

- reset() がいない構文もあります

```
foreach ($ar as $val) {  
    echo $val;  
}  
foreach ($ar as $key => $val) {  
    echo "$key: $val¥n";  
}
```



# 関数(1)

- 文法的には普通

```
function funcname( $arg1, $arg2 = "hoge" ) {  
    :  
}
```



## 関数(2)

- 返値を配列で戻せます

```
function hoge() {  
    return array( 1, "a" );  
}  
list( $num, $str ) = hoge();
```



## 関数(3)

- 引数の参照渡しもあります

```
function hoge( &$a ) {  
    $a++;  
}  
$fuga = 1;  
hoge( $fuga );
```



# クラス

- クラスもあります

```
class Base {  
    var $a;  
    function hoge() { }  
}  
class Derived extends Base {  
    var $b;  
    function fuga() { }  
}
```



# PHP3について

- 非常にすばらしい国際化(日本語化)バージョンあり
  - 最新版はPHP-3.0.18-i18n-ja-2
  - php.iniの設定により、文字コードの自動変換が可能
- ただしスクリプトエンジンとしては古い
  - PHP4でしか動作しない関数,構文が数多くある



# PHP4の利点

<http://www.zend.com/zend/whats-new.php>

- Zend Script Engineを採用
  - 参照のサポート
  - Reference Counting
  - Here Printing
  - Extention API の全面改定
  - クラス機能の強化
- セッションのサポート
- PCRE (Perl Compatible Regular Expressions)
- Output Bufferのサポート



# PHP4の欠点

- PHP3の国際化機能が使えない
  - 何も考えずに日本語を利用すると化ける(場合がある)
  - ただしベースとなるライブラリは移植されている → jstring
    - <ftp://night.fminn.nagano.nagano.jp/php4/>
  - 半自動で日本語を扱う方法を考えましょう



# PHP4で日本語を使う方法(1)

## ● 考え方

- PHP4にはOutput Bufferingがある
- 出力内容を一度全部バッファにためる
- 最後にjstringを用いてコード変換
- Content-Typeもきちんと設定
- クラス化して容易に呼び出せるようにする
  - なるべくおまじない的に記述できるように



## PHP4で日本語を使う方法(2)

- 考え方(続き)
  - Get/Post された値は変数に入るので自動変換
    - `$HTTP_POST_VARS`, `$HTTP_GET_VARS`
  - 入力文字コード判別は推測だけだと危険なので、ダミーの文字列埋め込みに対応しよう
    - 半角カタカナの入力はEUC-JPと区別がつかない
- 準備完了



# charconv.inc

## ● class CConv

- スクリプトの文字コードと出力の文字コードを別々に指定可
  - 標準ではPHP3のお勧め設定に合わせて  
内部: EUC-JP  
出力: SJIS
  - 先頭に定数で定義



# CConv使用法(1)

## ● 出力

```
<?php CConv::start(); ?>
<html>
<head><title>日本語テスト</title></head>
<body>
<?php echo "日本語です。<br>¥n"; ?>
</body>
</html>
<?php CConv::finish(); ?>
```



## CConv使用方法(2)

- フォームからの変数の受け取り

```
<?php  
CConv::convAll();  
// $input_valueを処理  
?>
```

- CConv::start() 内でも実は convAll() を呼んでいる
  - そのままHTMLを表示するような場合は start() だけでOK

- ダミーの変数

```
<input type="hidden" name="__encoding" value="適  
当な日本語文字列">
```



# PEARについて

<http://pear.php.net/>

- PHP Extension and Add-on Repository
  - TeXのCTAN、PerlのCPAN、RubyのRAA
  - PHPの標準ライブラリを目指し、鋭意開発中
  - Stig Bakken氏がメインで開発している
  - PHPのオブジェクト指向機能を存分に使いこなした Reference Code
  - PHPLibがPEARにマージされることになった



# PEAR DB

- PEAR内の抽象DBクラス
  - 使用するDBMSによってPHPの関数名、呼ぶ方法が異なる
    - 基本的にCのライブラリ関数をそのままPHPに置き直しているだけ
  - 同一インターフェースでDBの種類を気にせずにアクセスしたい
    - PerlのDBI + DBDに相当する



# PEAR DB使用法(1)

## ● 通常

```
$con_str = "host=localhost dbname=phptest user=koyama  
password=xxxx";  
$con = pg_connect( $con_str );  
$sql = "select * from test_tbl";  
$result = pg_exec( $con, $sql );  
$vals = pg_fetch_row( $result, 0 );  
foreach($vals as $val) {  
    echo "val: $val<br>¥n";  
}  
pg_freeresult( $result );  
pg_close( $con );
```



# PEAR DB使用法(2)

## ● PEAR DBを使うと

```
require_once( "DB.php" );
$dsn = "pgsql://koyama:xxxx@localhost/phptest";
$db = DB::connect( $dsn );
$sql = "select * from test_tbl";
$result = $db->simpleQuery( $sql );
$val = $db->fetchRow( $result );
foreach ( $val as $v ) {
    echo "val: $v<br>";
}
$db->freeresult( $result );
$db->disconnect();
```



# DBアクセスの不満

- SQLを毎回記述しなければならない
  - 複雑なSQLばかりではないだろう
  - あるテーブルの1レコードだけ全部取ってきたい
    - `select * from test_tbl where xxx='xxx'`
  - 単純なアクセスにもいちいちSQLを書かなくてはいけないのは納得いかない



# 抽象テーブルクラス

- 1レコード分の内容を保持する
  - 各フィールドはクラスのメンバー変数
  - primary keyを基準に読み込み/書きこみを行う
- SQLは**自動で生成される**
- 抽象化することによりあらゆるテーブルに対応
- 各DBMSの違いも吸収できる
  - 現在時刻、クォート文字



# 前提条件

- PostgreSQL-7.0.3
- PHP-4.0.4pl1
  - PHP4のみの機能を使っているのでPHP3では動作しません
  - PHP-4.0.4以前のPHP4には、セキュリティホールがあるので注意しましょう
- 以下のテーブルがすでに作られている

```
Create table t1 (  
    v1          int primary key,  
    v2          text,  
    v3          bool,  
    v4          date  
);
```



# hoge1.php

- pg\_xxx()を使用した例
  - pgsqlモジュールの関数をそのまま使用
  - DBアクセスの基本なので、一度はこのスタイルで書いてみると良いでしょう



# hoge2.php

- PEAR DBを使って書いた例
- DSN(data source name)を変更することにより、他のDBMSにも容易に移行できる
  - 客の要望はOracleだがPostgreSQLでプロトタイプ等
  - 実際には、各DBMSの違いにかなり苦しめられます(笑)



# hoge3.php

- まずテーブルクラスを作ってみる
  - テーブルの列はメンバー変数
    - 配列として保持する方法もあるが、列名を間違えたときの処理が難しい
  - メンバー関数はSQLの命令に対応
    - select / insert / update / delete
  - PostgreSQLではbool値が「t」「f」で返るので、PHPのtrue/falseに変換
  - 文字列は"で囲う
- 不満点
  - 使う側が insert/updateを切り替えないといけない



# hoge4.php

- クラスインターフェースの見直し
  - load/insert/update から setup/commit へ
    - commit()を呼ぶと、DBにレコードがあるかどうかを自動で判断し、insert/update を切り替える
  - getValsForDB()
    - insert/update の値生成



# hoge4.phpの問題点

- テーブルt1にしか対応しない
- テーブルごとに同じようなクラスをたくさん  
つくる?
- アクセス方法は同じなのでまとめてしまおう
  - そこで抽象化



# テーブルの抽象化とは？

- テーブルごとに違うものは？
  - テーブル名
  - おのおのの列名、型
  - primary key
- 抽象化するには、上記のものを内部に持たないクラスを作ればいい



# hoge5.php(1)

## ● class AbstractTable

- 抽象テーブルクラス
- これを継承するクラスは以下の3つのメンバー関数を定義する
  - getName()
  - getPrimaryKey()
  - getType()
- 制限
  - primary keyでしかレコードを指定できない
    - setup() で1レコードに限定するため



## hoge5.php(2)

- setup()
  - 該当レコードが存在すればDBから読み込む
- keyCheck()
  - primary keyが有効かどうか
    - primary keyの値が空なら、読み込み/書きこみは無意味
- makeWhereStr()
  - where句を自動生成



# hoge5.php(3)

## ● getVal()

- 各型によって、SQLに埋め込む方法が異なる
  - 文字列は ' ' で囲む
  - 空ならば NULLを入れる

## ● exists()

- 該当レコードがテーブルにあるかどうか調べる
  - primary keyを条件にしてselect文を発行
  - 結果の行があればテーブル内に存在する



## hoge5.php(4)

- commit()
  - exists()の結果により、update()/inset() を呼び出す
- load()
  - レコードをDBから読み出して、自分のメンバー変数に入れる
  - DB\_FETCHMODE\_ASSOC
    - 配列に、場所(数値)とフィールド名(文字列)両方をkeyにして、値を入れる



# hoge5.php(5)

- insert()
  - DBにレコードを挿入
- update()
  - DBのレコードを更新



## hoge5.php(6)

- Table1クラスはAbstractTableを継承して作成
  - class Table1 extends AbstractTable { ... }
  - 最低限の関数を実装しているだけ
  - 関数の呼び出し方法は hoge4.php と同じ



# 可変変数

- 文字列から、その名前の変数の値を得る
  - `$name = "job";`
  - `echo $$name; // $job`
- クラスのメンバー変数では?
  - `$name = "member";`
  - `echo $this->$name; // $this->member`



# get\_class\_vars()

- get\_object\_vars( \$this )
  - そのオブジェクトのメンバー変数を配列にする
- get\_class\_vars()
  - そのクラス名のメンバー変数を配列にする

```
class Hoge { var $val1 = ""; }  
$h = new Hoge;  
$h->val2 = "val2";  
get_object_vars( $h );           // "val1", "val2"  
get_class_vars( get_class( $h ) ); // "val1"
```



# 抽象テーブルを別ファイルに

- abstracttable.inc
  - クラス定義自体は同一
- 別ファイルにすることにより、ライブラリ的に使用できる



# hoge6.php

- 実用例らしい例
  - 顧客情報テーブル
  - SQLをまったく書かずに、DBにアクセスできる
  - メンバー関数をオーバーライド
    - 新規に作成したときにregist\_dateに現在時刻
    - 変更するたびにlast\_updatedに現在時刻



# スクリプト用設定ファイル(1)

- define() で定義する定数を、ひとつのファイルにまとめておきたい
  - さまざまな実行環境に対応して、各人ごとに設定を切り替えたい
    - メインの設定ファイルと .xxxrc の関係?
  - localconfig.incがもしあれば、その設定を優先し、そうでなければ標準どおりに動く



## スクリプト用設定ファイル(2)

- 定数定義ファイルの先頭に以下を追加

```
define( "LOCAL_CONFIG_PATH", "localconfig.inc" );  
function isLocalConfigExists() {  
    $fp = @fopen( LOCAL_CONFIG_PATH, "r", 1 );  
    if (!$fp)  
        return false;  
    fclose( $fp );  
    return true;  
}  
if (isLocalConfigExists()) {  
    include_once( LOCAL_CONFIG_PATH );  
}
```



# PHPの最近のトピック

<http://www.zend.com/>

- Zend Cache
  - スクリプトを効率良くキャッシュすることにより高速化を図る
- Zend Encoder
  - スクリプトを中間コードに変換する。ソースコードを隠せる。
- Zend IDE
  - 統合開発環境
    - Javaベース
    - ソースコードデバッガあり



# トピック続き

<http://gtk.php.net/>

- サーバサイドだけではなく、クライアントにも...
  - PHP-GTK
- 汎用スクリプト言語の道を歩む?



# Q&A