今日からはじめる PHPエクステンション

関山隆介<rsky0711@gmail.com>

What?

PHPエクステンションって何？

- C言語またはC++言語で書かれたPHPの機能を拡張するモジュール
- PHPスクリプトより高速に動作
- 要コンパイル
- 公式のエクステンションはPECLというリポジトリにまとめられており、“PECL”はPHPエクステンションの代名詞として用いられることもある

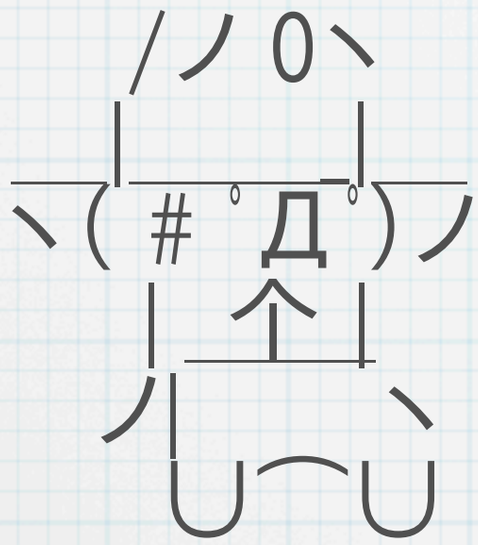
Why?

なぜPHPエクステンションをつくるのか？

- Pure PHPでは実現が困難な機能がほしい
  - PHPはPerl, Python, Rubyに比べてC/C++で書かれたライブラリのバインディングが少ない
  - SWIG (http://www.swig.org/) のPHPサポートが他言語に比べて不十分なのも原因のひとつ
- ボトルネックの解消
- ただ作ってみたかった
  - エクステンション作成を通じてPHPをより深く知ることができた

Who?

だれのためのPHPエクステンション？



- Good for you! (俺によし)
- Good for me! (お前によし)
- Good for all! (みんなによし)
- エクステンションを作ったら、ぜひ公開してみましょう。きっと、より良いものになります。

When, Where?

いつ、どこで？

Now and Here!

いま、ここで！

How?

PHPエクステンションをつくるには？

- ひな形を生成するスクリプトを利用する
- **ext_skel**
  - phpのソースコードに付属するスクリプト
    (php-x.x.x/ext/ext_skel)
- **PEAR::CodeGen_PECL**
  - XMLで記述されたspecファイルからヘッダ、
    ソースコード、テストケース、マニュアルなどの
    ひな形を生成する

開発環境

- PHP バージョン 5.1 以降、5.2 を推奨
- *BSD, Linux, Mac OS X などの UNIX互換OS
  (またはCygwinなどのWindows上でUNIXのコマンドが使える環境)
- CodeGen_PECL バージョン 1.1.0

CodeGen_PECLをインストール

PEARインストーラで通常のPEARパッケージと同様にインストールする

```
% sudo pear install --alldeps CodeGen_PECL
```

ひな形作成コマンド“pecl-gen”の使い方を確認する

```
% pecl-gen --help
```

“Hello, World!” を作ってみる
1. specファイルを書く

helloworld.xml

```
% cat helloworld.xml
<?xml version="1.0"?>
<extension name="helloworld" version="1.0.0">
  <function name="helloworld">
    <proto>void helloworld()</proto>
    <code><![CDATA[
php_printf("Hello, World!");
    ]]></code>
  </function>
</extension>
```

✓ PHPエクステンションでは printf(3) の代わりに php_printf 関数を使う

“Hello, World!” を作ってみる
2. pecl-genを実行

specファイルからソースコードを生成

```
% pecl-gen helloworld.xml
Creating 'helloworld' extension in './helloworld'

Your extension has been created in directory
./helloworld.
See ./helloworld/README and/or ./helloworld/INSTALL for
further instructions.
```

“Hello, World!” を作ってみる
3. コンパイル&インストール

phpizeコマンドでconfigureスクリプトを生成

```
% cd helloworld
% phpize
% ./configure --help
```

./configure && make && make install

```
% ./configure --enable-helloworld
% make
% sudo make install
```

“Hello, World!” を作ってみる
4. helloworld()関数をコール

コマンドラインでhelloworld

```
% php -r ‘dl(“helloworld.so”); helloworld();’
Hello, World!
```

PHPの-dオプションを使って

```
% php -d extension=helloworld.so -r ‘helloworld();’
Hello, World!
```

“Hello, World!” を作ってみる
5. エクステンション情報をチェック

PHPの--reオプションで情報を表示

```
% php -d extension=helloworld.so --re helloworld
Extension [ <persistent> extension #46 helloworld
version 0.0.1 ] {
  - Functions {
    Function [ <internal:helloworld> function
helloworld ] {
      - Parameters [0] {
      }
    }
  }
}
```

✓ specファイルで `version="1.0.0"` としたのに...?

specファイル詳細

specファイル詳細
1. 基本情報とライセンス

```
<?xml version="1.0"?>
<extension name="helloworld" version="1.0.0">
  <summary>Hello, World!</summary>
  <description>
This is a Hello-World PHP extension.
  </description>

  <license>PHP</license>
```

- ✓ <summary>タグにエクステンションの概要を記述する
- ✓ <description>タグにエクステンションの概要を記述する
- ✓ <summary>,<description>は他の要素の情報を記述するのにも使う
- ✓ CodeGen_PECLの親クラス、CodeGenが対応しているライセンスは BSD, GPL, LGPL, PHP だが、PHPライセンスはGPLと相いれないので、CodeGen_PECLで使えるのはGPLを除いた3つ

specファイル詳細
2. メンテナ

```
<maintainers>
  <maintainer>
    <user>rsk</user>
    <name>Ryusuke SEKIYAMA</name>
    <email>rsky0711@gmail.com</email>
    <role>lead</role>
  </maintainer>
</maintainers>
```

- ✓ <maintainers>は1つ以上の<maintainer>を子要素として持つ
- ✓ <user>にはアカウントやハンドルネームなどの短い名前を、<name>にはフルネームを記述する
- ✓ <role>に指定できる値は lead, developer, contributor, helper のいずれかひとつ

specファイル詳細
3. リリース情報

```
  <release>
    <version>1.0.0</version>
    <date>2007-09-01</date>
    <state>stable</state>
    <notes>
- My First PHP Extension.
    </notes>
  </release>
```

- ✓ バージョンナンバーをソースコードに反映させるには<extension>の属性値としてではなく、<release>の子要素<version>を使う
- ✓ <state>に指定できる値は alpha, beta, stable, snapshot, devel のいずれかひとつ
- ✓ 今回は初回リリース扱いとして省略しているが、<changelog>要素を作成し、複数の<release>を子要素とすることで更新履歴を記述することもできる

specファイル詳細
4. 定数

```
<constants>
  <constant name="HW_INT" value="1" type="int">
      integer: 1
  </constant>
  <constant name="HW_FLT" value="1.0" type="float">
      float: 1.0
  </constant>
  <constant name="HW_STR" value="Hello, World!"
type="string">
      string: &amp;quot;Hello, World!&amp;quot;
  </constant>
</constants>
```

- ✓ <constant>のtype属性に指定できるは int, float, string のみ
- ✓ 定数の説明は<constant>タグ内に記述する

specファイル詳細
5. カスタムコード

```
  <code position="top"><![CDATA[
#include <ext/standard/php_smart_str.h>
  ]]></code>
```

- ✓ <extension>直下にもソースコードを記述することができる
- ✓ カスタムコードはposition属性でコードが配置される場所を指定することができる。とりうる値は top, bottom のどちらかで、デフォルト値は bottom
- ✓ 通常、<code position="top">にヘッダのインクルードや関数のプロトタイプなどを、<code position="bottom">に関数の実装を書く
- ✓ 依存するライブラリのヘッダは<deps>タグで指定する
- ✓ ext/standard/php_smart_str.hではC言語レベルで「文字列」を扱うための関数が定義されている

specファイル詳細
6. 関数

```
  <function name="helloworld" role="public">
    <proto>void helloworld()</proto>
    <summary>Hello, World!</summary>
    <description>
Prints &amp;quot;Hello, World!&amp;quot;.
    </description>
    <code><![CDATA[
php_printf("Hello, World!");
    ]]></code>
(続く)
```

- ✓ <function>のrole属性は internal, public のどちらかを値として取り、デフォルト値は public (通常のPHP関数)
- ✓ エクステンション初期化などの特別な処理をする関数を定義したいときにrole="internal"を使う
- ✓ <proto>に記述した関数プロトタイプを元に関数の戻り値や引数に関するコードが生成される

specファイル詳細
6. 関数

```
（続き）
    <test>
      <code>helloworld();</code>
      <result mode="plain">Hello, World!</result>
    </test>
  </function>
```

- ✓ 関数のテストケースは<function>内に<test>として記述する
- ✓ <code>にテスト用のPHPコード、<result>に期待される出力を記述する。省略された場合のデフォルト値は<code>が echo “OK”;、<result>が OK となっている
- ✓ <result>のmode属性に指定できる値は plain (生の出力), format (フォーマット文字列), regex (正規表現) のいずれか

specファイル詳細
7. internal関数

```
  <function name="MINFO" role="internal">
    <code><![CDATA[
php_info_print_table_start();
php_info_print_table_row(2, "Hello World Extension",
"enabled");
php_info_print_table_row(2, "Module Version", "1.0.0");
php_info_print_table_end();
    ]]></code>
  </function>
```

- ✓ <function role="internal">の場合、name属性で関数が呼ばれるタイミングを指定する。とりうる値は MINFO (モジュール情報), MINIT (モジュール初期化), MSHUTDOWN (モジュール終了), RINIT (リクエスト初期化), RSHUTDOWN (リクエスト終了) のいずれか
- ✓ php_info_print_table_*()関数を使うと、CLIではテキスト、その他SAPIではHTMLテーブルを出力することができる

specファイル詳細
9. その他

HelloWorldエクステンションのspecファイルで使っていないspecファイルのタグ

- <deps> 依存するライブラリやPHPエクステンション
- <globals> スレッドセーフなグローバル変数
  - <phpini> php.ini で設定可能なグローバル変数
- <resources> リソース
- <class> クラス (PHP4非対応)
  - <constants> クラス定数
  - <function> メソッド

関数の追加とZend API

関数の追加とZend API
1. 文字列を返す関数

```
  <function name="helloworld_get" role="public">
    <proto>string helloworld_get()</proto>
    <code><![CDATA[
RETURN_STRING("Hello, World!", 1);
    ]]></code>
  </function>
```

- ✓ <proto>に記述した戻り値の型はドキュメント生成に使われるだけで、実際の型はphp/Zend/zend_API.hで定義されている関数型マクロRETURN_*(), RETVAL_*(), ZVAL_*()で決定される
  - ✓ RETURN_*()は RETVAL_*(); return; と定義されている
  - ✓ RETVAL_*()は ZVAL_*(return_value, ...) と定義されている
  - ✓ return_valueはPHP関数を定義するためのマクロPHP_FUNCTION(*function_name*)で宣言される、戻り値の変数コンテナ

関数の追加とZend API
2. 参照を受け取る関数

```
  <function name="helloworld_set" role="public">
    <proto>void helloworld_set(mixed &amp;var)</proto>
    <code><![CDATA[
zval_dtor(var);
ZVAL_STRING(var, "Hello, World!", 1);
    ]]></code>
  </function>
```

- ✓ 参照として受け取った変数は`zval_dtor()`で変数の中身を開放してから`ZVAL_STRING()`などで新たな値を設定する
- ✓ 本来ならプロトタイプで引数を `mixed &var` と、変数名の前に `&` を付けると参照を受け取るようにコードが生成されるはずだが、CodeGen_PECL 1.1.0にはこれが正しく解釈されないバグがあるので、specファイルでは `mixed var` としておき、pecl-genで生成したヘッダを編集してやる必要がある

関数の追加とZend API
2. 参照を受け取る関数

php_helloworld.h.diff

```
--- php_helloworld.h.orig
+++ php_helloworld.h
@@ -94,10 +94,10 @@
 PHP_FUNCTION(helloworld_set);
 #if (PHP_MAJOR_VERSION >= 5)
 ZEND_BEGIN_ARG_INFO_EX(helloworld_set_arg_info,
ZEND_SEND_BY_VAL, ZEND_RETURN_VALUE, 1)
-   ZEND_ARG_INFO(0, var)
+   ZEND_ARG_INFO(1, var)
 ZEND_END_ARG_INFO()
 #else /* PHP 4.x */
-#define helloworld_set_arg_info NULL
+#define helloworld_set_arg_info first_arg_force_ref
 #endif

 PHP_FUNCTION(helloworld_put);
```

関数の追加とZend API
3. ファイルに書き込む関数

```
  <function name="helloworld_put" role="public">
    <proto>int helloworld_put(string url)</proto>
    <code><![CDATA[
php_stream *stream;
size_t len;
stream = php_stream_open_wrapper((char *)url, "wb",
    ENFORCE_SAFE_MODE | REPORT_ERRORS, NULL);
if (stream == NULL) {
   RETURN_FALSE;
}
len = php_stream_write_string(stream, "Hello, World!");
php_stream_close(stream);
RETURN_LONG((long)len);
    ]]></code>
  </function>
```

関数の追加とZend API
3. ファイルに書き込む関数

- ✓ PHPでは通常のファイルポインタの代わりにphp_stream*型のストリーム変数とphp/main/php_streams.hで定義されているストリームを操作する変数を使うことでhttp, compress.zlibなどのストリームも同じAPIで操作できる
- ✓ プロトタイプでstring型として宣言した変数に対しては、`const char *name` と `int name len` が定義される
- ✓ `php_stream_open_wrapper()`の第一引数は`char *`型として宣言されているので、`const char *url`を`char *`にキャストしている
- ✓ Zend EngineおよびPHPのAPIは`const char *`をとるべきところで`char *`、`size_t`をとるべきところで`int`というのが割と多い

関数の追加とZend API
4. オプション引数とsmart_str

```
  <function name="helloworld_repeat" role="public">
    <proto>string helloworld_repeat(int times[, string
separator])</proto>
    <code><![CDATA[
if (times < 0L) { /* PHPのint = Cのlong */

   php_error(E_WARNING,
      "Argument 1 should not be a negative number.");
   RETURN_FALSE;
} else {
   /* 次頁参照 */
}
    ]]></code>
  </function>
```

- ✓ <proto>で引数をブラケット(角括弧)で囲むと、オプション引数となる
- ✓ エラーを発生させるにはphp_error()を使う

関数の追加とZend API
4. オプション引数とsmart_str

```
smart_str str = {0}; long n = 0L;
while (n++ < times) {
    smart_str_appends(&str, "Hello, World!");
    if (separator != NULL && separator_len > 0) {
        smart_str_appendl(&str, separator, separator_len);
    } else {
        smart_str_appendc(&str, ' ');
    }
}
if (separator != NULL && separator_len > 0) {
    RETVAL_STRINGL(str.c, str.len - separator_len, 1);
} else {
    RETVAL_STRINGL(str.c, str.len - 1, 1);
}
smart_str_free(&str);
```

- ✓ ext/standard/php_smart_str.hの関数を使うとC言語レベルで「文字列」を扱うことができる

関数の追加とZend API
5. 配列を返す関数

```
  <function name="helloworld_array" role="public">
    <proto>array helloworld_array(int times)</proto>
    <code><![CDATA[
if (times < 0L) {
    zval_dtor(return_value);
    php_error(E_WARNING,
        "Argument 1 should not be a negative number.");
    RETURN_FALSE;
} else {
    long n = 0L;
    while (n++ < times) {
        add_next_index_string(return_value, "Hello, World!", 1);
    }
}
    ]]></code>
  </function>
```

関数の追加とZend API
5. 配列を返す関数

- ✓ 配列に変数を追加するにはphp/Zend/zend_API.hで定義されている配列操作関数を使う
- ✓ プロトタイプで戻り値の型をarrayとして宣言している場合、return_valueを配列変数として初期化済みのコードが生成されるので、配列以外の変数を返すときはまずreturn_valueを`zval_dtor()`で開放しておく

関数の追加とZend API
6. 配列を調べる関数とzval

```
  <function name="helloworld_in_array" role="public">
    <proto>bool helloworld_in_array(array arr[, bool
ignore_case])</proto>
    <code><![CDATA[
zval **entry = NULL;

zend_hash_internal_pointer_reset(arr_hash);

while (zend_hash_get_current_data(arr_hash,
      (void **)&entry) == SUCCESS) {
   /* 次頁参照 */
   zend_hash_move_forward(arr_hash);
}

RETURN_FALSE;
    ]]></code>
  </function>
```

関数の追加とZend API
6. 配列を調べる関数とzval

```
if (Z_TYPE_PP(entry) == IS_STRING) {
    if (ignore_case) {
        if (!strcasecmp("Hello, World!", Z_STRVAL_PP(entry))) {
            RETURN_TRUE;
        }
    } else {
        if (!strcmp("Hello, World!", Z_STRVAL_PP(entry))) {
            RETURN_TRUE;
        }
    }
}
```

関数の追加とZend API
6. 配列を調べる関数とzval

- ✓ プロトタイプでarray型として宣言した変数に対しては、`zval *name` と `HashTable *name_hash` が定義される
- ✓ 配列の内容にアクセスするにはphp/Zend/zend_hash.hで定義されているハッシュテーブル操作関数を使う
- ✓ 変数の型を調べるにはphp/Zend/zend_operators.hで定義されているマクロ`Z_TYPE(zval)`, `Z_TYPE_P(zval *)`, `Z_TYPE_PP(zval **)` とphp/Zend/zend.hで定義されているマクロ`IS_*`の値を比較する
  - `IS_NULL`: NULL
  - `IS_LONG`: 整数 (符号付き長整数)
  - `IS_DOUBLE`: 浮動小数点数 (倍精度実数)
  - `IS_BOOL`: 論理値
  - `IS_ARRAY`: 配列
  - `IS_OBJECT`: オブジェクト
  - `IS_STRING`: 文字列
  - `IS_RESOURCE`: リソース

関数の追加とZend API
6. 配列を調べる関数とzval

- ✓ 変数の値を取得するはphp/Zend/zend_operators.hで定義されているマクロ`Z_*VAL(zval), Z_*VAL_P(zval *), Z_*VAL_PP(zval **)`を使う
  - `long Z_LVAL(zval)`
  - `double Z_DVAL(zval)`
  - `zend_bool Z_BVAL(zval)`
  - `HashTable * Z_ARRVAL(zval)`
  - `HashTable * Z_OBJPROP(zval)`
  - `char * Z_STRVAL(zval)`
  - `int Z_STRLEN(zval)`
- ✓ リソースの値を取得するはphp/Zend/zend_list.hで定義されているマクロ`ZEND_FETCH_RESOURCE()`を使う
- ✓ CodeGen_PECLでは引数の型をresource <リソース名>として宣言すれば、リソースの中身を取得するコードが生成される

関数の追加とZend API
7. CからPHPの関数を使う

```
  <function name="helloworld_say" role="public">
    <proto>bool helloworld_say()</proto>
    <code><![CDATA[
zval **argv[] = { NULL, NULL, NULL };
zval *command, *output, *result_code, *retval = NULL;
zval func;

MAKE_STD_ZVAL(command);
MAKE_STD_ZVAL(output);
MAKE_STD_ZVAL(result_code);

ZVAL_STRING(command, "say 'Hello, World!'", 1);
ZVAL_NULL(output);
ZVAL_NULL(result_code);
ZVAL_STRING(&func, "exec", 0);
(続く)
```

関数の追加とZend API
7. CからPHPの関数を使う

```
(続き)
argv[0] = &command;
argv[1] = &output;
argv[2] = &result_code;

if (call_user_function_ex(EG(function_table), NULL, &func,
      &retval, 3, argv, 0, NULL TSRMLS_CC) == FAILURE)
{
   RETVAL_FALSE;
} else if (Z_TYPE_P(result_code) == IS_LONG &&
    Z_LVAL_P(result_code) == 0L) {
   RETVAL_TRUE;
} else {
   RETVAL_FALSE;
}
(続く)
```

関数の追加とZend API
7. CからPHPの関数を使う

```
(続き)
if (retval != NULL) {
    zval_ptr_dtor(&retval);
}
zval_ptr_dtor(&command);
zval_ptr_dtor(&output);
zval_ptr_dtor(&result_code);
    ]]></code>
  </function>
```

- ✓ CからPHPの関数を呼ぶには、call_user_function_ex()を使う
  - ✓ 呼び出しに成功したときの戻り値はSUCCESS、失敗したときの戻り値はFAILURE
- ✓ 変数コンテナのメモリ領域を確保して初期化するにはMAKE_STD_ZVAL(zval *)、開放するにはzval_ptr_dtor(zval **)を使う

関数の追加とZend API
8. コールバック関数をとる関数

```
  <function name="helloworld_apply" role="public">
    <proto>mixed helloworld_apply(callback func[, array
extra_args])</proto>
    <code><![CDATA[
zval ***argv, ***argp;
zval *arg1;
int argc = 1;
zval *retval = NULL;

if (extra_args_hash != NULL) {
   argc += zend_hash_num_elements(extra_args_hash);
}

argv = (zval ***)emalloc(sizeof(zval **) * argc);
argp = argv;
(続く)
```

関数の追加とZend API
8. コールバック関数をとる関数

```
(続き)
MAKE_STD_ZVAL(arg1);
ZVAL_STRING(arg1, "Hello, World!", 1);
*argp++ = &arg1;
if (extra_args_hash != NULL) {
    zval **entry;
    zval *arg;
    zend_hash_internal_pointer_reset(extra_args_hash);
    while (zend_hash_get_current_data(extra_args_hash,
     (void **)&entry) == SUCCESS) {
        MAKE_STD_ZVAL(arg);
        ZVAL_ZVAL(arg, *entry, 1, 0);
        *argp++ = &arg;
        zend_hash_move_forward(extra_args_hash);
    }
}
(続く)
```

関数の追加とZend API
8. コールバック関数をとる関数

```
(続き)
if (call_user_function_ex(EG(function_table), NULL, func,
        &retval, (zend_uint)argc, argv, 0,
        NULL TSRMLS_CC) == FAILURE) {
    RETVAL_FALSE;
} else {
    RETVAL_ZVAL(retval, 1, 1);
}

while (argp > argv) {
    zval_ptr_dtor(*(--argp));
}
efree(argv);
    ]]></code>
  </function>
```

関数の追加とZend API
8. コールバック関数をとる関数

helloworld.c.diff

```
--- helloworld.c.orig
+++ helloworld.c
@@ -424,7 +424,9 @@
    if (!zend_is_callable(func, 0, NULL)) {
      php_error(E_WARNING, "Invalid comparison
function.");
      return;    }
-   extra_args_hash = HASH_OF(extra_args);
+   if (extra_args != NULL) {
+      extra_args_hash = HASH_OF(extra_args);
+   }
```

関数の追加とZend API
8. コールバック関数をとる関数

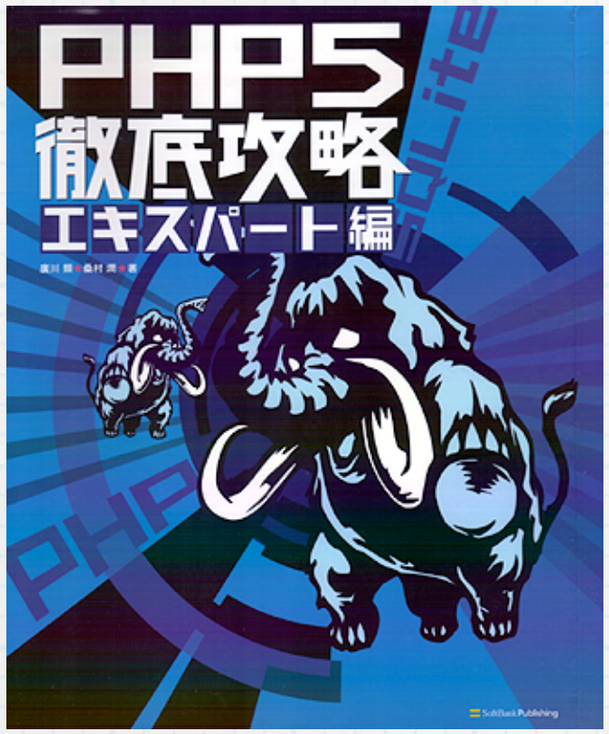
- ✓ プロトタイプでcallback型として宣言した変数に対しては、`zval *name` が定義され、正しいコールバック関数かどうかのチェックをするコードも生成される
- ✓ 変数のコピーにはマクロ`ZVAL_ZVAL(zval *dst, zval *src, int copy, int dtor)`を使う
- ✓ `copy`が`1`なら変数の内容（文字列なら`char *`、配列なら`HashTable *`など）も新たに確保したメモリ領域にコピーされる
- ✓ `dtor`が`1`なら`zval_ptr_dtor()`によって`src`が開放される。このとき`copy`が`0`なら変数の内容までは開放されない
- ✓ `ZVAL_ZVAL`はPHP4では定義されておらず、利用できない
  - ✓ PHP5のphp/Zend/zend_API.hからコピペして使うことは可能
- ✓ `ZVAL_ZVAL`の派生として`RETVAL_ZVAL(zv, copy, dtor)`、`RETURN_ZVAL(zv, copy, dtor)`もある

関数の追加とZend API
8. コールバック関数をとる関数

- ✓ PHPエクステンションでは`malloc()`の代わりに`emalloc()`、`free()`の代わりに`efree()`を使う（`realloc()`や`calloc()`についても同様に接頭辞`e`をつけたものを使う）
  - ✓ PHPのメモリ管理用関数・マクロはphp/Zend/zend_alloc.hで定義されている
- ✓ CodeGen_PECLのバグにより、オプション引数にした配列変数についても常にハッシュテーブルを取得しようとするので、生成されたコードを修正する必要がある

参考文献

参考書籍



- PHP5徹底攻略 エキスパート編
  - 通称・青マンモス本
  - 出版社: ソフトバンク パブリッシング
  - 著者: 廣川 類 / 桑村 潤
  - 和書で唯一PHPエクステンションの作り方を詳細に解説している書籍
  - http://www.amazon.co.jp/dp/4797331305/

参考サイト

- CodeGen_PECLマニュアル
  - http://php-baustelle.de/CodeGen_PECL/manual.html
- CodeGen_PECL specファイルの文書型定義
  - http://codegenerators.php-baustelle.de/trac/browser/trunk/CodeGen_PECL/docs/extension.dtd
- Do You PHP? - PEAR::CodeGen_PECLを使ってPECLモジュールを作ってみる
  - http://www.doyouphp.jp/tips/tips_pear_codegen_pecl.shtml
- はてなダイアリー「PECL」を含む日記
  - http://d.hatena.ne.jp/keyworddiary/PECL

Appendix

本講演で作成したエクステンションのソースコードの入手先

- http://www.opendogs.org/pub/phpcon2007/helloworld.xml
  (最初に作成したシンプルなspecファイル)
- helloworld.zip (helloworld.xmlから生成したソースコード一式)
- helloworld-full.xml (後から作成したspecファイル)
- helloworld-full.zip (helloworld-full.xmlから生成したソースコード一式)
- helloworld.diff (helloworld-full.xmlから生成したソースコードの修正パッチ、helloworld-full.zipには適用済み)

Zend API超簡易リファレンス

PHPの変数はC言語レベルではzval構造体という変数コンテナに収められており、PHPエクステンションを作成するにあたっては変数の扱い方=zval構造体の操作に関する知識が必要不可欠です。

しかしながら講演ではそこまで紹介しきれなかったので、付録として最低限知っておきたい、以下のAPIについて紹介します。

- zvalを操作する関数・マクロ
- 変数を作成するマクロ
- 配列を操作する関数

これらの内容については参考書席でも紹介した「PHP5徹底攻略 エキスパート編 (青マンモス本)」を参考に作成させていただきました。より詳細な情報は青マンモス本やPHPのヘッダファイルを参照下さい。(PHPのソースコードではマクロが多用されており、読むのは少し大変かもしれませんが...)

zvalを操作する関数・マクロ

定義	説明
`INIT_ZVAL(zval z)`	初期化済みのzvalを代入する
`INIT_PZVAL(zval *z)`	リファレンスカウンタをリセットする
`ALLOC_ZVAL(zval *z)`	変数コンテナ用のメモリ領域を確保する
`ALLOC_INIT_ZVAL(zval *z)`	`ALLOC_ZVAL()` + `INI_ZVAL()`
`MAKE_STD_ZVAL(zval *z)`	`ALLOC_ZVAL()` + `INI_PZVAL()`
`ZVAL_ADDREF(zval *z)`	リファレンスカウンタをインクリメントする
`ZVAL_DELREF(zval *z)`	リファレンスカウンタをデクリメントする
`ZVAL_REFCOUNT(zval *z)`	リファレンスカウンタを取得する
`PZVAL_IS_REF(zval *z)`	変数がリファレンスかどうかを取得する
`zval_copy_ctor(zval *z)`	変数の内容を新たに確保したメモリ領域にコピーする。`*dst_zv = *src_zv;` として変数の値をコピーした後に `zval_copy_ctor(dst_zv);` として使うのが一般的だが、リファレンスカウンタの扱いを考慮すると、変数のコピーにはマクロ`ZVAL_ZVAL()`を使う方がよい
`zval_dtor(zval *z)`	変数の内容を開放する
`zval_ptr_dtor(zval **z)`	リファレンスカウンタをデクリメントし、0になったら変数の内容を開放した後、変数コンテナを開放する

変数を作成するマクロ

定義	作成される変数
ZVAL_BOOL(zval *z, zend_bool b)	(b==0) ? false : true
ZVAL_TRUE(zval *z)	bool(true)
ZVAL_FALSE(zval *z)	bool(false)
ZVAL_LONG(zval *z, long l)	int(l)
ZVAL_DOUBLE(zval *z, double d)	float(d)
ZVAL_STRING(zval *z, char *s, int dup)	長さstrlen(s)の文字列 dup!=0 のとき文字列を複製
ZVAL_STRINGL(zval *z, char *s, int len, int dup)	長さlenの文字列 dup!=0 のとき文字列を複製
ZVAL_EMPTY_STRING(zval *z)	空文字列
ZVAL_NULL(zval *z)	NULL
ZVAL_ZVAL(zval *z, zval *zv, int copy, int dtor)	zvと同じ PHP4では使えない

- ✓ 関数の戻り値を設定するRETVAL_*系マクロ、戻りを設定してreturnするRETURN_*系マクロは上記マクロの派生であり、第一引数のzval *をとらない以外は同じである
- ✓ PHP6にはUnicode文字列を扱うZVAL_UNICODE系マクロなどもある

配列を操作する関数

定義	説明
`array_init(zval *z)`	変数コンテナを配列として初期化する
`add_index_unset(zval *a, ulong i)`	i番目の要素をNULLにする
`add_assoc_unset(zval *a, char *key)`	指定したキーの要素をNULLにする

- ✓ 現在のPHPでは`add_*_unset()`は要素を削除するのではなく、NULLにする
- ✓ 指定した添字番号の要素を削除するには `zend_hash_del(Z_ARRVAL_P(a), i)`、指定したキーの要素を削除するには `zend_hash_del(Z_ARRVAL_P(a), key, strlen(key) + 1)` とする
- ✓ ハッシュテーブルのキーは終端文字`\0`も含めて管理されているので、キーの長さは `strlen(文字列変数) + 1` もしくは `sizeof(文字列リテラル)` としなければならない

配列を操作する関数

配列の最後に変数を追加する

定義	追加される変数
`add_next_index_bool(zval *a, zend_bool b)`	`(b==0) ? false : true`
`add_next_index_long(zval *a, long l)`	`int(l)`
`add_next_index_double(zval *a, double d)`	`float(d)`
`add_next_index_string(zval *a, char *s, int dup)`	長さ`strlen(s)`の文字列 `dup!=0` のとき文字列を複製
`add_next_index_stringl(zval *a, char *s, int len, int dup)`	長さ`len`の文字列 `dup!=0` のとき文字列を複製
`add_next_index_null(zval *a)`	`NULL`
`add_next_index_zval(zval *a, zval *zv)`	`zv`

配列を操作する関数

添字を指定して変数を追加または上書きする

定義	追加される変数
`add_index_bool(zval *a, ulong i, zend_bool b)`	`(b==0) ? false : true`
`add_index_long(zval *a, ulong i, long l)`	`int(l)`
`add_index_double(zval *a, ulong i, double d)`	`float(d)`
`add_index_string(zval *a, ulong i, char *s, int dup)`	長さ`strlen(s)`の文字列 `dup!=0` のとき文字列を複製
`add_index_stringl(zval *a, ulong i, char *s, int len, int dup)`	長さ`len`の文字列 `dup!=0` のとき文字列を複製
`add_index_null(zval *a, ulong i)`	`NULL`
`add_index_zval(zval *a, ulong i, zval *zv)`	`zv`

配列を操作する関数

連想配列のキーを指定して変数を追加または上書きする

定義	追加される変数
`add_assoc_bool(zval *a, char *key, zend_bool b)`	`(b==0) ? false : true`
`add_assoc_long(zval *a, char *key, long l)`	`int(l)`
`add_assoc_double(zval *a, char *key, double d)`	`float(d)`
`add_assoc_string(zval *a, char *key, char *s, int dup)`	長さ`strlen(s)`の文字列 `dup!=0` のとき文字列を複製
`add_assoc_stringl(zval *a, char *key, char *s, int len, int dup)`	長さ`len`の文字列 `dup!=0` のとき文字列を複製
`add_assoc_null(zval *a, char *key)`	`NULL`
`add_assoc_zval(zval *a, char *key, zval *zv)`	`zv`

TSRM
(Thread Safe Resource Manager)

- ZTS (マルチスレッド) モードでPHPがスレッドセーフに動作するための機構で、TLS (スレッドローカルストレージ) を使っている
- コンテクスト変数`tsrm_ls`の受け渡しには関数定義側で`void`の代わりにマクロ`TSRMLS_D`、または他の引数に続けて`TSRMLS_DC`を指定し、呼び出し側では`TSRMLS_C`、`TSRMLS_CC`を指定する
- 関数実装においてブロックの先頭で`TSRMLS_FETCH();`としてコンテクストを取得し、引数には`TSRMLS_*`を書かない方法もある
- ZTSモードで動かないエクステンションは`TSRMLS_*`を忘れていたり、マルチスレッドを考慮していないコーディングをしていると思われる
- よく使うAPIにはマクロで`TSRMLS_*`が隠ぺいされているものが多い上に、非ZTSモードでは`TSRMLS_*`を書かなくても正常に動作するために忘れられがちだが、マルチスレッドで正常に動作するエクステンションを作成するなら忘れてはいけない