PHP, Process Thyself

Andrei Zmievski
Analog

http://zmievski.org/talks

demo

Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.

Buffalo, NY

Buffalo buffalo Buffalo buffalo buffalo buffalo buffalo Buffalo buffalo.

bison, *pl.*

Buffalo **buffalo** Buffalo buffalo buffalo buffalo buffalo Buffalo buffalo.

Buffalo buffalo Buffalo buffalo

buffalo buffalo Buffalo buffalo.

to bully,
intimidate, *v.*

Buffalo buffalo

Buffalo buffalo Buffalo buffalo buffalo buffalo

Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.

why preprocess?

- use new/different syntax
- real macros (via GPP)
- server-specific configuration
- optimization (inlining, etc)

what is prep?

- hooks into the compilation process
- uses dependency resolution to ensure it's loaded last
- invokes external commands to process the input files
- almost useless without an opcode cache, but you need one anyway

commands

- INI setting `prep.command`
  - via `php.ini`, `.htaccess`, or `-d`
- specifies an executable
  - input: filename as the first argument
  - output: the result to stdout
- example (with `chmod +x devprod.php`):

```
prep.command=/usr/local/prep-scripts/devprod.php
```

commands

- placeholders can be used to alter argument position:
  - `%s` - file to be processed
  - `%o` - file with the original source
- example:

```
prep.command=/usr/bin/gpp -C -D DEV=1 %s
```

suppression

- `PHP_SUPPRESS_PREP=1` disables prep
- used by prep itself to avoid recursive processing
- shell:

```
PHP_SUPPRESS_PREP=1 php myscript.php
```

- Apache:

```
<Location /noprep>
  SetEnv PHP_SUPPRESS_PREP 1
</Location>
```

chaining

- prep supports chaining of commands (up to 9 currently)
- output of one feeds into the next:
  - decode whitespace → remove dev portions
- additional INI settings:

```
prep.command2
...
prep.command9
```

- why `%o` placeholder might be useful
- commands can communicate with each other via filesystem, memcache, etc

exit code

- command exit codes:
  - 0 - normal, processing successful
  - 1 - skip processing, compile normally
  - 255 - processing error, provide extra error info
  - anything else - processing error, standard error message
- not coincidentally, PHP uses exit code 255 to report compilation errors

advantages

- runs in pre-compile stage, the result can be stored in opcode cache
- can use any combination of processors, in any language

disadvantages

- small (depending on the command) performance hit
- prep scripts have to be carefully constructed to avoid breaking apps

other approaches

- custom streams handler
- processing can be done only in PHP
- sees only a portion of the input file at a time

other approaches

- offline/build-time processing
- need infrastructure
- big pain for dynamic stuff, especially during development

simple examples

- development vs. production changes
- custom namespace separator

using prep with other tools

- gpp
- tokenizer extension
- tokalizer (ask Sean Coates to resurrect it)
- lex-pass (if you can grok it)

gpp

- general-purpose preprocessor
- fully-customizable macro syntax
- at the basic level, use –C for cpp compatibility mode

```
echo "Environment: ";
#ifdef DEV
echo "dev";
#else
echo "production";
#endif
echo "\n";

#define DEVSLEEP(n) sleep(n)
DEVSLEEP(1);
```

tokenizer

~ extension that converts a PHP file into a stream of lexical tokens

```php
$tokens = token_get_all(file_get_contents($argv[1]));

$max = count($tokens);
for ($i = 0; $i < $max; $i++) {

    /* Single-character token */
    if (!is_array($tokens[$i])) {
        echo $tokens[$i];
        continue;
    }

    /* Named tokens */
    if (T_STRING == $tokens[$i][0] &&
        $tokens[$i][1] == 'phpversion' &&
        $i+2 < $max && '(' == $tokens[$i+1] && ')' == $tokens[$i+2]) {
        echo '"'.phpversion().'"';
        $i += 2;
    } else {
        echo $tokens[$i][1];
    }
}
```

tokalizer

- Sean Coates' (@coates) quest to improve tokenizer (needs some TLC though)
- OO-interface to access and manipulate token stream
- not speedy, but doesn't matter in this case

```
$ts = TokenSet::fromFile($argv[1]);
foreach ($ts as $token) {
   if ($token instanceof ProceduralFunctionCallToken &&
      $token->getValue() == "phpversion") {
      echo '"'.phpversion().'"';
      $ts->next();
      $ts->next();
   } else {
      echo $token->reconstruct();
   }
}
```

lex-pass

- FB project
- manipulation of codebase via abstract syntax tree (AST) transformations
- written in and uses Haskell for transformers!

```haskell
renameFunc :: String -> String -> Ast -> Transformed Ast
renameFunc oldF newF = modAll $ \ a -> case a of
  ROnlyValFunc (Right (Const [] f)) w args ->
    if f == oldF
      then pure $ ROnlyValFunc (Right $ Const [] newF) w args
      else transfNothing
  _ -> transfNothing
```

what else?

single-command chaining

- may be easier to manage the chain from one script

```
#!/bin/bash

TMPFILE=.prep_multi_output
DIR=`dirname $0`
${DIR}/foo.php $1 > $TMPFILE
${DIR}/bar.php $TMPFILE
rm $TMPFILE
```

decorators

- a way in Python to programmatically wrap callables in other code
- syntactic sugar, but very useful

```
def entryExit(f):
    def new_f():
        print "Entering", f.__name__
        f()
        print "Exited", f.__name__
    return new_f

@entryExit
def func1():
    print "inside func1()"
```

decorators

~ simplified compile-time code expansion via prep

```
function logger($func, $args)
{
    echo "entering $func()\n";
    $return = call_user_func_array($func, $args);
    echo "exiting $func()\n";
    return $return;
}

@logger
function square($a)
{
    print "calculating square($a)\n";
    return $a * $a;
}

print square(2)."\n";
```

what else?

- ~ chaining via one script
- ~ decorators
- ~ inlining code
- ~ inlining includes
- ~ inlining Lithium filters
- ~ gathering statistics
- ~ baking in translations
- ~ code verification
- ~ DSLs

dirty tricks

- export your prep jobs to `gearman` if they're really heavy for some reason (such as re-generating phpdoc on the fly)
- replace all hardcoded images with `data:` URLs
- auto-inline JS
- running encrypted code (not really practical)
- more?

future work

- might make its way into PECL
- selectively processing the files via regexp applied to the filename

resources

- prep: http://github.com/andreiz/prep
- slides: http://zmievski.org/talks
- http://en.nothingisreal.com/wiki/GPP
- http://php.net/tokenizer
- http://github.com/scoates/tokalizer
- http://github.com/facebook/lex-pass

Thank you!

~Andrei