

APACHE HACKS

Beth Skwarecki

Pittsburgh Perl Workshop

2007-10-13

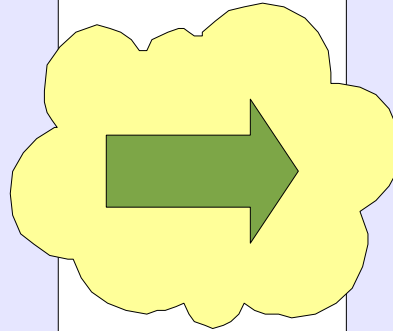
Slides and code:

<http://BethSkwarecki.com/ppw2007>

A SIMPLE OUTPUT FILTER

A horse walks
into a foo.

The footender
says, why the
long face?



A horse walks
into a **bar**.

The **bartender**
says, why the
long face?

USING THE FILTER

In your apache2 configuration:

```
PerlLoadModule Example::RewriteStuff
```

```
<Files *.html>
```

```
    PerlOutputFilterHandler Example::RewriteStuff
```

```
    RewriteWhatToWhat foo bar
```

```
</Files>
```

HOW FILTERS WORK



ESSENTIAL PARTS OF THIS FILTER

```
package Example::RewriteStuff;

# Register our module
Apache2::Module::add(__PACKAGE__, \@directives);

# This processes our custom directive
sub RewriteWhatToWhat { ... }

# This does the filtering
sub handler { ... s/foo/bar/; ...}
```

HELPFUL PERL MODULES

Apache2::Module lets your perl module become an Apache module.

Apache2::Filter provides a filter object (\$f) and helpful filter management functions.

Apache2::Const - constants

Apache2::CmdParms – \$parms

Apache2::RequestRec – request object \$r

APR::Table – unset Content-Length

DEFINE YOUR DIRECTIVES AND ADD YOUR MODULE

```
my @directives = (  
    {  
        name => 'RewriteWhatToWhat',  
        args_how => Apache2::Const::TAKE2,  
        errmsg => 'Args: FROM-STRING TO-STRING',  
    },  
);
```

```
Apache2::Module::add(__PACKAGE__, \@directives);
```

(This goes in Example::RewriteStuff)

THE DIRECTIVE

```
sub RewriteWhatToWhat {
```

```
my ($self, $parms, @args) = @_;
```

```
my $srv_cfg = Apache2::Module::get_config  
($self, $parms->server);
```

```
# process/store params for later use
```

```
($srv_cfg->{from},  
$srv_cfg->{to}) = map quotemeta, @args;
```

```
}
```

(This goes in Example::RewriteStuff)

THE HANDLER

```
sub handler{
```

```
my $f = shift;
```

```
# [unset Content-length here]
```

```
while ($f->read(my $buffer, 1024)){
```

```
$buffer =~ s/
```

```
    $cfg->{from}
```

```
/
```

```
    $cfg->{to}
```

```
/gx;
```

```
$f->print($buffer);
```

```
}
```

(This goes in Example::RewriteStuff)

WHAT ARE FILTERS GOOD FOR?

- Rewrite links in the page
- Tidy up code as you send it
- Screw with PHP (mod_perl cookbook)
- What nifty filter will YOU write?

WHAT ELSE CAN YOU DO WITH APACHE2 FILTERS?

- Input filters
- Dynamic filters (decide at request time what filters to run)
- Connection filters (munge headers as well as body)

HACKING APACHE::REGISTRY (MAJOR SURGERY)

This next example uses **Apache 1.3**

(Apache 2 users: go ahead and boo)

(Apache 1 users: you can wake up now)

ACTUALLY, IT'S REALLY EASY

1. Move & rename Apache/Registry.pm

3. Point apache to your renamed module:

```
PerlModule Example::MyRegistry
```

```
PerlHandler Example::MyRegistry
```

4. Hack away! (this part may be less easy)

CATCH A COMPILE TIME ERROR

```
# ...  
  
compile($eval);  
  $r->stash_rgy_endav($script_name);  
  if ($@) {  
    # your code here  
    xlog_error($r, $@);  
    return SERVER_ERROR unless $Debug && $Debug &  
2;  
    return Apache::Debug::dump($r, SERVER_ERROR);  
  }  
# ...
```

(This is in Example::MyRegistry)

CATCH A RUNTIME ERROR

```
use Example::Purgatory; # catches $SIG{__DIE__}

# ...

if($errsv) {
    # your code here
    xlog_error($r, $errsv);
    return SERVER_ERROR unless $Debug && $Debug
& 2;
    return Apache::Debug::dump($r, SERVER_ERROR);
}
# ...
```

(This is in Example::MyRegistry)

EMAIL THE BACKTRACE

```
use Carp::Heavy;  
use Mail::Sendmail;
```

```
$SIG{__DIE__} = sub {
```

```
    sendmail (  
        To => 'developers@yourcompany',  
        Subject => "$page died",  
        Body => Carp::longmess_heavy()  
    );
```

```
}
```

(This is in Example::Purgatory)



THE END!

<http://bethskwarecki.com/ppw2007>

APACHE HACKS

Beth Skwarecki

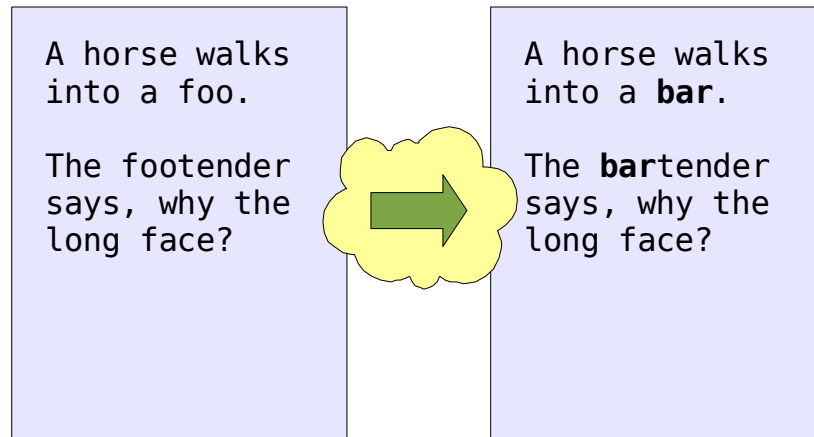
Pittsburgh Perl Workshop

2007-10-13

Slides and code:

<http://BethSkwarecki.com/ppw2007>

A SIMPLE OUTPUT FILTER



2

Goal the first (there are two): write an Apache filter that filters text, for example changing “foo” to “bar” on the fly. The box on the left is a file you have on disk; the box on the right is what the user sees when they request that page. The magical cloud in the middle is Apache.

Apache output filters can do more than just replacing strings (they're written in perl, so they can do anything perl can do) but we've chosen a simple replace for our example.

USING THE FILTER

In your apache2 configuration:

```
PerlLoadModule Example::RewriteStuff  
  
<Files *.html>  
    PerlOutputFilterHandler Example::RewriteStuff  
    RewriteWhatToWhat foo bar  
</Files>
```

3

Now, how do you USE an Apache output filter? It's an apache module, so you need to tell Apache how to use your filter.

We're calling our module `Example::RewriteStuff`. (real creative, huh?) We even let the user – the person who's configuring Apache with our module – decide what strings they'd like to replace. Same idea as arguments in a CLI program.

This goes in your `httpd.conf` or, for me, the appropriate file in `sites-enabled`.

(Yes, `apache2`. This kind of filtering doesn't work with earlier Apaches.)

HOW FILTERS WORK



4

Apache2 filters can stack; after one filter is done with its portion of data, the data goes to the next filter.

Data comes in “buckets”, and when you read up on Apache filters, you'll hear a lot about bucket brigades. [Details are elided here. My examples use the stream-oriented API, so buckets are behind the scenes.]

User-defined filters are processed in the same order as their configuration: the first-defined filter goes first. [By the time your filters are invoked, the INCLUDES filter has already been run.]

Pictured: a 3-bucket water filter that removes arsenic from water. Invented by Abul Hussam for use in his native Bangladesh.

ESSENTIAL PARTS OF THIS FILTER

```
package Example::RewriteStuff;

# Register our module
Apache2::Module::add(__PACKAGE__, \@directives);

# This processes our custom directive
sub RewriteWhatToWhat { ... }

# This does the filtering
sub handler { ... s/foo/bar/; ...}
```

5

This is not a working filter; lots of stuff is missing.
(See the example code for the rest, at
<http://bethskwarecki.com/ppw2007>)

But the basics are here: we register the module (this mainly includes defining our directives like RewriteWhatToWhat).

We have a sub that the directive executes upon parsing (so it runs each time Apache reads its config file)

and we have the handler sub that does the hard work (in our case, the regex that substitutes “bar” for “foo”).

HELPFUL PERL MODULES

Apache2::Module lets your perl module become an Apache module.

Apache2::Filter provides a filter object (\$f) and helpful filter management functions.

Apache2::Const - constants

Apache2::CmdParms – \$parms

Apache2::RequestRec – request object \$r

APR::Table – unset Content-Length

6

Speaks for itself, I think. Perldoc for more info.

APR::Table provides unset() for content-length

DEFINE YOUR DIRECTIVES AND ADD YOUR MODULE

```
my @directives = (  
    {  
        name => 'RewriteWhatToWhat',  
        args_how => Apache2::Const::TAKE2,  
        errmsg => 'Args: FROM-STRING TO-STRING',  
    },  
);  
  
Apache2::Module::add(__PACKAGE__, \@directives);
```

(This goes in Example::RewriteStuff)

7

`req_override` says where the directive can legally appear. `OR_ALL` means it can be just about anywhere.

`args_how` describes the arguments. In this case, we take 2 arguments (the from-string and to-string)

`errmsg` will be used if you invoke the directive incorrectly (for example, wrong number of arguments)

The name is the name of the directive, and `func` is the function that it maps to. They don't need to have the same name.

More info on those funky Apache constants here:
<http://perl.apache.org/docs/2.0/user/config/custom.html>

THE DIRECTIVE

```
sub RewriteWhatToWhat {
```

```
my ($self, $parms, @args) = @_;
```

```
my $srv_cfg = Apache2::Module::get_config  
($self, $parms->server);
```

```
# process/store params for later use  
($srv_cfg->{from},  
$srv_cfg->{to}) = map quotemeta, @args;
```

```
}
```

(This goes in Example::RewriteStuff)

THE HANDLER

```
sub handler{  
  
  my $f = shift;  
  # [unset Content-length here]  
  
  while ($f->read(my $buffer, 1024)){  
    $buffer =~ s/  
      $cfg->{from}  
      /  
      $cfg->{to}  
      /gx;  
    $f->print($buffer);  
  
  }  
  
  (This goes in Example::RewriteStuff)
```

9

where does \$cfg come from?

what happens if \$from crosses a 1024-byte boundary?

WHAT ARE FILTERS GOOD FOR?

- Rewrite links in the page
- Tidy up code as you send it
- Screw with PHP (mod_perl cookbook)
- What nifty filter will YOU write?

10

What filter will YOU write?

- I wrote a filter to **munge URLs** in a reverse proxy. (mod_proxy_html rewrites links in HTML but I needed to also rewrite URLs in CSS and javascript.)
- Two filters by Graham TerMarsch “minify” code – they **remove whitespace**. Apache::Clean by Geoffrey Young tidies HTML on the fly.
- Screw with PHP (mod_perl cookbook)
- [Your name here!]

WHAT ELSE CAN YOU DO WITH APACHE2 FILTERS?

- Input filters
- Dynamic filters (decide at request time what filters to run)
- Connection filters (munge headers as well as body)

HACKING APACHE::REGISTRY (MAJOR SURGERY)

This next example uses **Apache 1.3**

(Apache 2 users: go ahead and boo)

(Apache 1 users: you can wake up now)

ACTUALLY, IT'S REALLY EASY

1. Move & rename Apache/Registry.pm
3. Point apache to your renamed module:

```
PerlModule Example::MyRegistry  
PerlHandler Example::MyRegistry
```

4. Hack away! (this part may be less easy)

13

Example: `mv /usr/lib/perl5/Apache/Registry.pm /usr/local/lib/site_perl/Example/MyRegistry.pm`

Remember to change the “package” line to match (it's the first line in that file).

CATCH A COMPILE TIME ERROR

```
# ...  
  
compile($eval);  
$r->stash_rgy_endav($script_name);  
if ($@) {  
    # your code here  
    xlog_error($r, $@);  
    return SERVER_ERROR unless $Debug && $Debug &  
2;  
    return Apache::Debug::dump($r, SERVER_ERROR);  
}  
# ...
```

(This is in Example::MyRegistry)

CATCH A RUNTIME ERROR

```
use Example::Purgatory; # catches $SIG{__DIE__}

# ...

if($errsv) {
    # your code here
    xlog_error($r, $errsv);
    return SERVER_ERROR unless $Debug && $Debug
& 2;
    return Apache::Debug::dump($r, SERVER_ERROR);
}
# ...
```

(This is in Example::MyRegistry)

15

runtime_error just displays a helpful message for the user.

Example::Purgatory catches the die signal.

EMAIL THE BACKTRACE

```
use Carp::Heavy;  
use Mail::Sendmail;
```



```
$SIG{__DIE__} = sub {  
  
    sendmail (  
        To => 'developers@yourcompany',  
        Subject => "$page died",  
        Body => Carp::longmess_heavy()  
    );  
  
}
```

(This is in Example::Purgatory)

THE END!

<http://bethskwarecki.com/ppw2007>