Apache 2.0 Filters

_ Greg Ames
_ Jeff Trawick

Agenda

- Why filters?
- Filter data structures and utilites
- An example Apache filter
- Filter types
- Configuration directives

Agenda cont...

- Frequently used Apache filters
  - Output filters
  - Input filters
- Pitfalls
  - ways to avoid them
- mod_ext_filter
- Debugging hints
- The big filter list

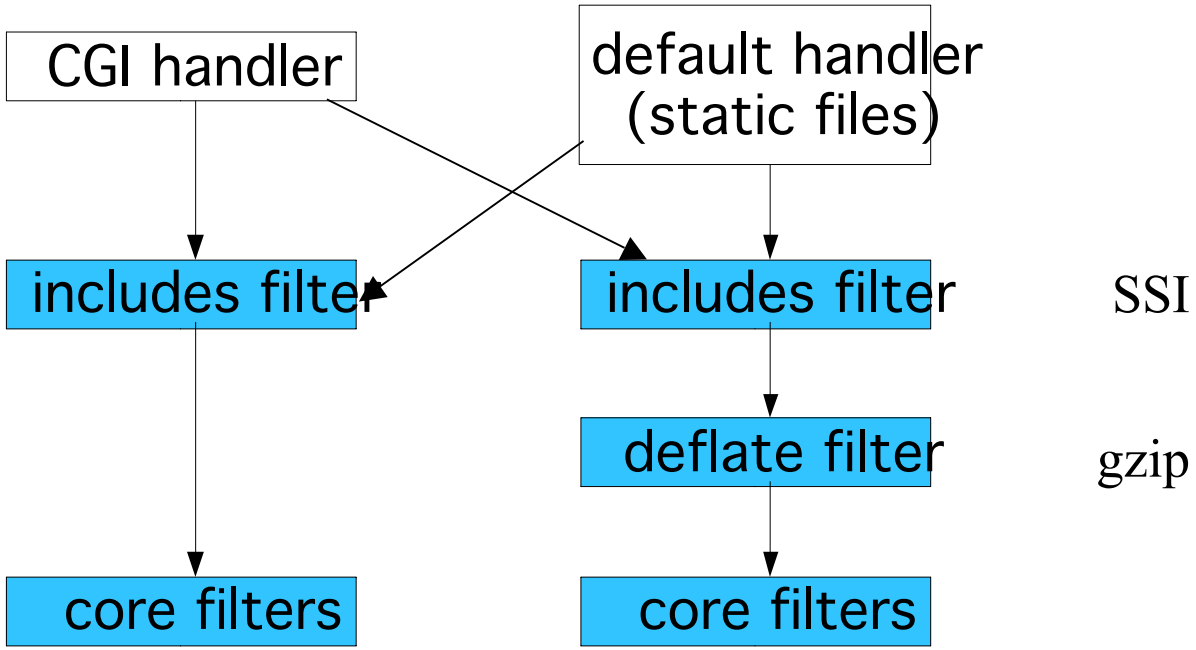
Why filters?

- Same idea as Unix command line filters:

```
ps ax | grep "apache.*httpd" | wc -l
```

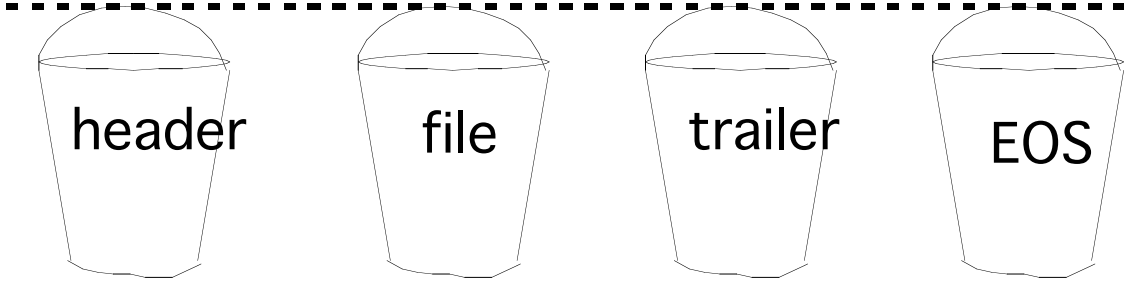
- Allows independent, modular manipulations of HTTP data stream
  - if a 1.3 CGI creates SSI or PHP tags, they aren't parsed
  - CGI created SSI tags can be parsed in 2.0
  - possible Zend issue w/PHP at present

The general idea



data can be manipulated independently from how it's generated

Bucket brigades



a complex data stream that can be passed thru layered I/O
without unnecessary copying

SPECWeb99 uses dynamically generated headers and trailers around static files
header and trailer live in memory based buckets
file bucket contains the fd
End Of Stream metadata bucket is the terminator

Filter utilities and structures

- ap_register_[input|output]_filter
  - creates ap_filter_rec_t
- ap_add_[input|output]_filter[_handle]
  - creates ap_filter_t
- ap_pass_brigade
  - passes a bucket brigade to the next output filter
- ap_get_brigade
  - gets a bucket brigade from the next input filter

ap_filter_t

```
struct ap_filter_t {
    ap_filter_rec_t *frec;     <- description
    void *ctx;                 <- context (instance variables)
    ap_filter_t *next;
    request_rec *r;            <- HTTP request info
    conn_rec *c;               <- connection info
};
```

created by ap_add_[input|output]_filter[_handle]

the blue boxes on previous slide

ap_filter_rec_t

```
struct ap_filter_rec_t {
    const char *name;                       <- normalized to
    ap_filter_func filter_func;
    ap_init_filter_func filter_init_func;
    ap_filter_type ftype;                   <- determines ins
    struct ap_filter_rec_t *next;
};
```

created by ap_register_[inputloutput]_filter

An example Apache filter

- mod_case_filter
- lives in modules/experimental
- mission: convert data to upper case

mod_case_filter

```
static apr_status_t CaseFilterOutFilter(ap_filter_t *f,
                                        apr_bucket_brigade *pbbIn)
    {
    request_rec *r = f->r;
    conn_rec *c = r->connection;
    apr_bucket *pbktIn;
    apr_bucket_brigade *pbbOut;

    pbbOut=apr_brigade_create(r->pool, c->bucket_alloc);
```

mod_case_filter...

```
APR_BRIGADE_FOREACH(pbktIn,pbbIn)      <-- iterates thru all the buckets
      {
      const char *data;
      apr_size_t len;
      char *buf;
      apr_size_t n;
      apr_bucket *pbktOut;

      if(APR_BUCKET_IS_EOS(pbktIn))
         {                                   /* terminate output brigade */
         apr_bucket *pbktEOS=apr_bucket_eos_create(c->bucket_alloc);
         APR_BRIGADE_INSERT_TAIL(pbbOut,pbktEOS);
         continue;
         }
```

mod_case_filter...

```
    /* read */                              <-- morphs bucket into memory based type
    apr_bucket_read(pbktIn,&data,&len,APR_BLOCK_READ);

    /* write */
    buf = apr_bucket_alloc(len, c->bucket_alloc);     <-- allocates buffer
    for(n=0 ; n < len ; ++n)
        buf[n] = apr_toupper(data[n]);

    pbktOut = apr_bucket_heap_create(buf, len, apr_bucket_free,  <-- creates bucket
                                c->bucket_alloc);
    APR_BRIGADE_INSERT_TAIL(pbbOut,pbktOut);    <-- adds to output brigade
    }

return ap_pass_brigade(f->next,pbbOut);      <-- passes brigade to next output filter
}
```

Filter types

- determines where filter is inserted
- AP_FTYPE_RESOURCE (SSI, PHP, case filter)
- AP_FTYPE_CONTENT_SET (deflate, cache)
- AP_FTYPE_PROTOCOL (HTTP)
- AP_FTYPE_ TRANSCODE (chunk)
- AP_FTYPE_NETWORK (core_in, core_out)
- see include/util_filter.h for details

Filter configuration directives

SetOutputFilter

- activates the named filter in this scope:

  <Directory /www/data/ >
  SetOutputFilter INCLUDES
  </Directory>

- SetInputFilter is the same for input

AddOutputFilter

- like SetOutputFilter, but uses extension:

  <Directory /www/moredata/>
  AddOutputFilter INCLUDES;DEFLATE html
  </Directory>

- AddInputFilter is the same for input

RemoveOutputFilter

- resets filter to extension mappings

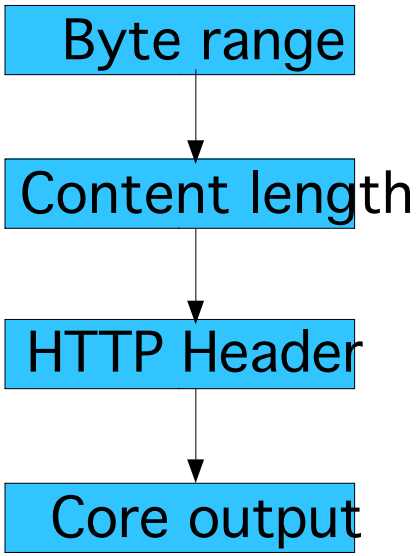
  <Directory /www/moredata/dont_parse>
  RemoveOutputFilter html
  </Directory>

  removes .html filters in this subdirectory

- RemoveInputFilter is the same for input

Frequently used Apache filters

Output filters



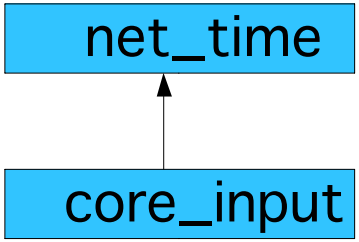
<-- removes itself if no Range: header

<-- calculates total content length, if practical

<-- generates headers from headers_out table

<-- writes to the network

Input filters



<-- sets socket timeouts

<-- reads socket buckets; length cop

Pitfalls

Excessive memory consumption

- what will happen if this filter is fed a 2G file and the client has a 9600 bps modem?
- solution:
  - limit your buffer sizes, or don't buffer
  - call ap_pass_brigade periodically
- ap_pass_brigade will block when socket buffers are full and downstream filters are well behaved

Holding up streaming output

- a CGI writes a "search in progress" message
- ...then does a lengthy database query
- filters are called before all the output exists
- filter sees a pipe bucket
- naïve approach will block reading the pipe
- client won't see the message

Holding up streaming output...

Specific solution (stolen from protocol.c::ap_content_length_filter):

```
read_type = nonblocking;
for each bucket e in the input brigade:
. if e is eos bucket, we're done;
. rv = e->read();
. read_type = non_blocking;
. if rv is APR_SUCCESS then process the data received;
. if rv is APR_EAGAIN:
. . add flush bucket to end of brigade containing data
    already processed;
. . ap_pass_brigade(data already processed);
. . read_type = blocking;
. if rv is anything else:
. . log error and return failure;
ap_pass_brigade(data already processed);
```

Avoiding the pitfalls with a simple well-behaved output filter

Make sure that the filter doesn't:

- consume too much virtual memory by touching a lot of storage before passing partial results to the next filter

- break streaming output by always doing blocking reads on pipe buckets

- break streaming output by not sending down a flush bucket when the filter discovers that no more output is available for a while

- busy-loop by always doing non-blocking reads on pipe buckets

Design for simple well-behaved output filter

```
read_mode = nonblock;
output brigade = empty; output bytes = 0;

foreach bucket in input brigade:
. if eos:
. . move bucket to end of output brigade; ap_pass_brigade();
. . return;
. if flush:
. . move bucket to end of output brigade; ap_pass_brigade();
. . output brigade = empty; output bytes = 0;
. . continue with next bucket
. read bucket;
. if rv is APR_SUCCESS:
. .  read_mode = non_block;
. .  process_bucket(); /* see next slide */
. if rv is EAGAIN:
. . add flush bucket to end of output brigade; ap_pass_brigade();
. . output brigade = empty; output bytes = 0;
. . read_mode = block;
. if output bytes > 8K:
. . ap_pass_brigade();
. . output brigade = empty; output bytes = 0;
ap_pass_brigade();
```

Design for simple well-behaved output filter...

```
process_bucket:

if len is 0:
. move bucket to output brigade;
while len > 0:
. n = min(len, 8K);
. get new buffer to hold output of processing n bytes;
. process the data;
. get heap bucket to represent new buffer;
. add heap bucket to end of output brigade;
. if output bytes > 8K:
. . ap_pass_brigade();
. . output brigade = empty; output bytes = 0;
. len = len - n;
```

mod_ext_filter

- Allows command-line filters to act as Apache filters
- Useful tool when implementing native Apache filters
  - Quick prototype of desired function
  - Can trace a native filter being tested

Simple mod_ext_filter example

"tidy" up the HTML

```
ExtFilterDefine tidy-filter \
cmd="/usr/local/bin/tidy"

<Location /manual/mod>
SetOutputFilter tidy-filter
ExtFilterOptions LogStderr
</Location>
```

mod_ext_filter and prototyping

Use a normal Unix filter to transform the response body

mod_ext_filter: perform some header transformations

mod_headers: add any required HTTP header fields

mod_setenvif: set environment variables to enable/disable the filter

mod_ext_filter header transformations

Content-Type

set to value of *outtype* parameter (unchanged otherwise)

```
ExtFilterDefine foo outtype=xxx/yyy
```

Content-Length

preserved or removed (default), based on the presence of *preservescontentlength* parameter

```
ExtFilterDefine foo preservescontentlength
```

Using mod_header to add headers

```
ExtFilterDefine gzip mode=output \
cmd=/bin/gzip
<Location /gzipped>
SetOutputFilter gzip
Header set Content-Encoding  gzip
</Location>
```

Enabling a filter via envvar

```
ExtFilterDefine nodirtywords \
cmd="/bin/sed s/damn/darn/g" \
EnableEnv=sanitize_output

<Directory />
SetOutputFilter nodirtywords
SetEnvIf Remote_Host ceo.mycompany.com \
sanitize_output
</Directory>
```

Tracing another filter

In httpd.conf:

```
# Trace the data read and written by another filter

# Trace the input:
ExtFilterDefine tracebefore EnableEnv=trace_this_client \
cmd="/usr/bin/tee /tmp/tracebefore"

# Trace the output:
ExtFilterDefine traceafter EnableEnv=trace_this_client \
cmd="/usr/bin/tee /tmp/traceafter"

<Directory /usr/local/docs>
  SetEnvIf Remote_Addr 192.168.1.31 trace_this_client
  SetOutputFilter tracebefore;some_other_filter;traceafter
</Directory>
```

General Apache debugging trick

- use the prefork MPM w/ 2 Listen statements
- ps axO ppid,wchan | grep httpd
  - The process to get the next connection is in poll
  - It will look unique (Linux: do_pol / schedu)
- attach debugger...gdb bin/httpd <pid>
- easy to script
- saves having to restart with -x
- can quickly detach/reattach

Filter debugging hints

- .gdbinit
  - dump_filters
  - dump_brigade
  - dump_bucket
- Breakpoints
  - core_[input|output]_filter
  - ap_[get|pass]_brigade

The big filter list

- doesn't include filters already covered
- mod_cache
  - has three filters
  - based on 1.3 mod_proxy cache
- mod_charset_lite
  - translates character encodings
  - essential on EBCDIC platforms

big list...

- mod_deflate
  - gzip/ungzip
- mod_include
  - SSI
- CHUNK
  - manages HTTP outbound chunking
- mod_logio
  - allows logging of total bytes sent/recv'd, incl. headers

big list...

- mod_header
  - lets you set arbitrary HTTP headers
- proxy_ftp
  - filter to send HTML directory listing
- mod_ssl
  - low level interfaces to SSL libraries

big list...

- mod_bucketeer
  - debugging tool for filters/buckets/brigades
  - separates, flushes, and passes data stream
  - uses control characters to specify where
- subrequest filter
  - eats EOS bucket used to end subreq output stream
- old_write
  - feeds buffered output of ap_rput* into filter chain