1 mod_perl internals: mod_perl-specific functionality flow

1.1 Description

This document attempts to help understand the code flow for certain features. This should help to debug problems and add new features.

This document auguments mod_perl internals: Apache 2.0 Integration and discusses the internals of the mod_perl-specific features.

Make sure to read also: Debugging mod_perl C Internals.

META: these notes are a bit out of sync with the latest svn, but will be updated once the innovation dust settles down.

1.2 Perl Interpreters

How and when Perl interpreters are created:

1. modperl_hook_init is invoked by one of two paths: Either normally, during the open_logs phase, or during the configuration parsing if a directive needs perl at the early stage (e.g. PerlLoadModule).

```
ap_hook_open_logs()            -> # normal mod_perl startup
load_module() -> modperl_run() -> # early startup caused by PerlLoadModule
```

2. modperl_hook_init() -> modperl_init():

```
o modperl_startup()
  - parent perl is created and started ("-e0"),
  - top level PerlRequire and PerlModule are run

o modperl_interp_init()
  - modperl_tipool_new() # create/init tipool
  - modperl_interp_new() # no new perls are created at this stage

o modperl_init_vhost() # vhosts are booted, for each vhost run:
  if +Parent
    - modperl_startup()  # vhost gets its own parent perl (not perl_clone()!)
  else
    - vhost's PerlModule/PerlRequire directives are run if any
  if +(Parent|Clone)
    - modperl_interp_init() (new tipool, no new perls created)
```

3. Next the post_config hook is run. It immediately returns for non-threaded mpms. Otherwise that's where all the first clones are created (and later their are created on demand when there aren't enough in the pool and more are needed).

```
o modperl_init_clones() creates pools of clones
  - modperl_tipool_init() (clones the PerlStartInterp number of perls)
    - interp_pool_grow()
      - modperl_interp_new()
        ~ this time perl_clone() is called
        ~ PL_ptr_table is scratched
          modperl_xs_dl_handles_clear
```

1.3 Filters

Apache filters work in the following way. First of all, a filter must be registered by its name, in addition providing a pointer to a function that should be executed when the filter is called and the type of resources it should be called on (e.g., only request's body, the headers, both and others). Once registered, the filter can be inserted into a chain of filters to be executed at run time.

For example in the pre_connection phase we can add connection phase filters, and using the ap_hook_insert_filter we can call functions that add the current request's filters. The filters are added using their registered name and a special context variable, which is typed to (void *) so modules can store anything they want there. You can add more than one filter with the same name to the same filter chain.

Here is how mod_perl uses this infrastructure:

There can be many filters inserted via mod_perl, but they all seen by Apache by four filter names:

```
MODPERL_REQUEST_OUTPUT
MODPERL_REQUEST_INPUT
MODPERL_CONNECTION_OUTPUT
MODPERL_CONNECTION_INPUT
```

XXX: which actually seems to be lowercased by Apache (saw it in gdb), (it handles these in the case insensitive manner?). how does then modperl_filter_add_request works, as it compares *fname with M.

These four filter names are registered in modperl_register_hooks():

```
ap_register_output_filter(MP_FILTER_REQUEST_OUTPUT_NAME,
                          MP_FILTER_HANDLER(modperl_output_filter_handler),
                          AP_FTYPE_RESOURCE);

ap_register_input_filter(MP_FILTER_REQUEST_INPUT_NAME,
                         MP_FILTER_HANDLER(modperl_input_filter_handler),
                         AP_FTYPE_RESOURCE);

ap_register_output_filter(MP_FILTER_CONNECTION_OUTPUT_NAME,
                          MP_FILTER_HANDLER(modperl_output_filter_handler),
                          AP_FTYPE_CONNECTION);

ap_register_input_filter(MP_FILTER_CONNECTION_INPUT_NAME,
                         MP_FILTER_HANDLER(modperl_input_filter_handler),
                         AP_FTYPE_CONNECTION);
```

At run time input filter handlers are always called by modperl_input_filter_handler() and output filter handler by modperl_output_filter_handler(). For example if there are three MODPERL_CONNECTION_INPUT filters in the filters chain, modperl_input_filter_handler() will be called three times.

The real Perl filter handler (callback) is stored in ctx->handler, which is retrieved by modperl_{output|input}_filter_handler and run as a normal Perl handler by modperl_run_filter() via modperl_callback():

```
                      retrieve ctx->handler
modperl_output_filter_handler -> modperl_run_filter -> modperl_callback
```

This trick allows to have more than one filter handler in the filters chain using the same Apache filter name (the real filter's name is stored in ctx->handler->name.

Now the only missing piece in the puzzle is how and when mod_perl filter handlers are inserted into the filter chain. It happens in three stages.

1. When the configuration file is parsed, every time a PerlInputFilterHandler or a PerlOutputFilterHandler directive is encountered, its argument (filter handler) is inserted into dcfg->handlers_per_dir[idx] by modperl_cmd_input_filter_handlers() and modperl_cmd_output_filter_handlers(). idx is either MP_INPUT_FILTER_HANDLER or MP_OUTPUT_FILTER_HANDLER. Since they are stored in the dcfg struct, normal merging of parent and child directories applies.

2. Next, modperl_hook_post_config calls modperl_mgv_hash_handlers which works through dcfg->handlers_per_dir[idx] and resolves the handlers (via modperl_mgv_resolve), so they are resolved by the time filter handlers are added to the chain in the next step (e.g. the attributes are set if any).

3. Now all is left is to add the filters to the appropriate chains at the appropriate time.

   modperl_register_hooks() adds a pre_connection hook modperl_hook_pre_connection() which inserts connection filters via:

   ```
   modperl_input_filter_add_connection();
   modperl_output_filter_add_connection();
   ```

   modperl_hook_pre_connection() is called during the pre_connection phase.

   modperl_register_hooks() directly registers the request filters via ap_hook_insert_filter():

   ```
   modperl_output_filter_add_request
   modperl_input_filter_add_request
   ```

   functions registered with ap_hook_insert_filter(), will be called when the request record is created and they are supposed to insert request filters if any.

   All four functions perform a similar thing: loop through dcfg->handlers_per_dir[idx], where idx is per filter type: MP_{INPUT|OUTPUT}_FILTER_HANDLER, pick the filters of the appropriate type and insert them to filter chain using one of the two Apache functions that add filters. Since we have connection and request filters there are four different combinations:

```
ap_add_input_filter( name, (void*)ctx, NULL, c);
ap_add_output_filter(name, (void*)ctx, NULL, c);
ap_add_input_filter( name, (void*)ctx, r,    r->connection);
ap_add_output_filter(name, (void*)ctx, r,    r->connection);
```

Here the name is one of:

```
MODPERL_REQUEST_OUTPUT
MODPERL_REQUEST_INPUT
MODPERL_CONNECTION_OUTPUT
MODPERL_CONNECTION_INPUT
```

ctx, storing three things:

```
SV *data;
modperl_handler_t *handler;
PerlInterpreter *perl;
```

we have mentioned ctx->handler already, that's where the real Perl filter handler is stored. ctx->perl stores the current perl interpreter (used only in the threaded environment).

the last two arguments are the request and connection records.

notice that dcfg->handlers_per_dir[idx] stores connection and request filters in the same array, so we have only two arrays, one for input and one for output filters. We know to distinguish between connection and request filters by looking at ctx->handler->attrs record, which is derived from the handler subroutine's attributes. Remember that we can have:

```
sub Foo : FilterRequestHandler {}
```

and:

```
sub Bar : FilterConnectionHandler {}
```

For example we can figure out what kind of handler is that via:

```
if (ctx->handler->attrs & MP_FILTER_CONNECTION_HANDLER)) {
    /* Connection handler */
}
else if (ctx->handler->attrs & MP_FILTER_REQUEST_HANDLER)) {
    /* Request handler */
}
else {
    /* Unknown */
}
```

1.4 Maintainers

Maintainer is the person(s) you should contact with updates, corrections and patches.

Stas Bekman [http://stason.org/]

1.5 Authors

- Stas Bekman [http://stason.org/]

Only the major authors are listed above. For contributors see the Changes file.

Table of Contents: