1   mod_perl internals: Apache 2.0 Integration

1.1 Description

This document should help to understand the initialization, request processing and shutdown process of the mod_perl module. This knowledge is essential for a less-painful debugging experience. It should also help to know where a new code should be added when a new feature is added.

Internals of mod_perl-specific features are discussed in mod_perl internals: mod_perl-specific functionality flow.

Make sure to read also: Debugging mod_perl C Internals.

1.2 Startup

Apache starts itself and immediately restart itself. The following sections discuss what happens to mod_perl during this period.

1.2.1 The Link Between mod_perl and httpd

mod_perl.c includes a special data structure:

```
module AP_MODULE_DECLARE_DATA perl_module = {
    STANDARD20_MODULE_STUFF,
    modperl_config_dir_create, /* dir config creater */
    modperl_config_dir_merge,  /* dir merger --- default is to override */
    modperl_config_srv_create, /* server config */
    modperl_config_srv_merge,  /* merge server config */
    modperl_cmds,              /* table of config file commands      */
    modperl_register_hooks,    /* register hooks */
};
```

Apache uses this structure to hook mod_perl in, and it specifies six custom callbacks which Apache will call at various stages that will be explained later.

`STANDARD20_MODULE_STUFF` is a standard macro defined in *httpd-2.0/include/http_config.h*. Currently its main use is for attaching Apache version magic numbers, so the previously compiled module won't be attempted to be used with newer Apache versions, whose API may have changed.

`modperl_cmds` is a struct, that defines the mod_perl configuration directives and the callbacks to be invoked for each of these.

1.3 Configuration Tree Building

At the `ap_read_config` stage the configuration file is parsed and stored in a parsed configuration tree is created. Some sections are stored unmodified in the parsed configuration tree to be processed after the `pre_config` hooks were run. Other sections are processed right away (e.g., the `Include` directive includes extra configuration and has to include it as soon as it was seen) and they may or may not add a subtree to the configuration tree.

`ap_build_config` feeds the configuration file lines from to `ap_build_config_sub`, which tokenizes the input, and uses the first token as a potential directive (command). It then calls `ap_find_command_in_modules()` to find a module that has registered that command (remember mod_perl has registered the directives in the `modperl_cmds command_rec` array, which was passed to `ap_add_module` inside the `perl_module` struct?). If that command is found and it has the `EXEC_ON_READ` flag set in its *req_override* field, the callback for that command is invoked. Depending on the command, it may perform some action and return (e.g., `User foo`), or it may continue reading from the configuration file and recursively execute other nested commands till it's done (e.g., `<Location ...>`). If the command is found but the `EXEC_ON_READ` flag is not set or the command is not found, the current node gets added to the configuration tree and will be processed during the `ap_process_config_tree()` stage, after the `pre_config` stage will be over.

If the command needs to be executed at this stage as it was just explained, `execute_now()` invokes the corresponding callback with `invoke_cmd`.

Since `LoadModule` directive has the `EXEC_ON_READ` flag set, that directive is executed as soon as it's seen and the modules its supposed to load get loaded right away.

For mod_perl loaded as a DSO object, this is when mod_perl starts its game.

1.3.1 Enabling the mod_perl Module and Installing its Callbacks

mod_perl can be loaded as a DSO object at startup time, or be prelinked at compile time.

For statically linked mod_perl, Apache enables mod_perl by calling `ap_add_module()`, which happens during the `ap_setup_prelinked_modules()` stage. The latter is happening before the configuration file is parsed.

When mod_perl is loaded as DSO:

```
<IfModule !mod_perl.c>
    LoadModule perl_module "modules/mod_perl.so"
</IfModule>
```

mod_dso's `load_module` first loads the shared mod_perl object, and then immediately calls `ap_add_loaded_module()` which calls `ap_add_module()` to enable mod_perl.

`ap_add_module()` adds the `perl_module` structure to the top of chained module list and calls `ap_register_hooks()` which calls the `modperl_register_hooks()` callback. This is the very first mod_perl hook that's called by Apache.

`modperl_register_hooks()` registers all the hooks that it wants to be called by Apache when the appropriate time comes. That includes configuration hooks, filter, connection and http protocol hooks. From now on most of the relationship between httpd and mod_perl is done via these hooks. Remember that in addition to these hooks, there are four hooks that were registered with `ap_add_module()`, and there are: `modperl_config_srv_create`, `modperl_config_srv_merge`, `modperl_config_dir_create` and `modperl_config_dir_merge`.

Finally after the hooks were registered, `ap_single_module_configure()` (called from mod_dso's `load_module` in case of DSO) runs the configuration process for the module. First it calls the `modperl_config_srv_create` callback for the main server, followed by the `modperl_config_dir_create` callback to create a directory structure for the main server. Notice that it passes `NULL` for the directory path, since we at the very top level.

If you need to do something as early as possible at mod_perl's startup, the `modperl_register_hooks()` is the right place to do that. For example we add a `MODPERL2` define to the `ap_server_config_defines` here:

```
  *(char **)apr_array_push(ap_server_config_defines) =
      apr_pstrdup(p, "MODPERL2");
```

so the following code will work under mod_perl 2.0 enabled Apache without explicitly passing `-DMOD-PERL2` at the server startup:

```
  <IfDefine MODPERL2>
      # 2.0 configuration
      PerlSwitches -wT
  </IfDefine>
```

This section, of course, will see the define only if inserted after the `LoadModule perl_module ...`, because that's when `modperl_register_hooks` is called.

One inconvenience with using that hook, is that the server object is not among its arguments, so if you need to access that object, the next earliest function is `modperl_config_srv_create()`. However remember that it'll be called once for the main server and one more time for each virtual host, that has something to do with mod_perl. So if you need to invoke it only for the main server, you can use a `s->is_virtual` conditional. For example we need to enable the debug tracing as early as possible, but we need the server object in order to do that, so we perform this setting in `modperl_config_srv_create()`:

```
  if (!s->is_virtual) {
      modperl_trace_level_set(s, NULL);
  }
```

1.4 The `pre_config` Phase

After Apache processes its command line arguments, creates various pools and reads the configuration file in, it runs the registered *pre_config* hooks by calling `ap_run_pre_config()`. That's when `modperl_hook_pre_config` is called. And it does nothing.

1.4.1 Configuration Tree Processing

`ap_process_config_tree` calls `ap_walk_config`, which scans through all directives in the parsed configuration tree, and executes each one by calling `ap_walk_config_sub`. This is a recursive process with many twists.

Similar to `ap_build_config_sub` for each command (directive) in the configuration tree, it calls `ap_find_command_in_modules` to find a module that registered that command. If the command is not found the server dies. Otherwise the callback for that command is invoked with `invoke_cmd`, after fetching the current directory configuration:

```
invoke_cmd(cmd, parms, dir_config, current->args);
```

The `invoke_cmd` command is the one that invokes mod_perl's directives callbacks, which reside in *modperl_cmd.c*. `invoke_cmd` knows how the arguments should be passed to the callbacks, based on the information in the `modperl_cmds` array that we have just mentioned.

Notice that before `invoke_cmd` is invoked, `ap_set_config_vectors()` is called which sets the current server and section configuration objects for the module in which the directive has been found. If these objects were't created yet, it calls the registered callbacks as `create_dir_config` and `create_server_config`, which are `modperl_config_dir_create` and `modperl_config_srv_create` for the mod_perl module. (If you write your custom module in Perl, these correspond to the `DIR_CREATE` and `SERVER_CREATE` Perl subroutines.)

The command callback won't be invoked if it has the `EXEC_ON_READ` flag set, because it was already invoked earlier when the configuration tree was parsed. `ap_set_config_vectors()` is called in any case, because it wasn't called during the `ap_build_config`.

So we have `modperl_config_srv_create` and `modperl_config_dir_create` both called once for the main server (at the end of processing the `LoadModule perl_module ...` directive), and one more time for each virtual host in which at least one mod_perl directive is encountered. In addition `modperl_config_dir_create` is called for every section and subsection that includes mod_perl directives (META: or inherits from such a section even though specifies no mod_perl directives in it?).

1.4.2 Virtual Hosts Fixup

After the configuration tree is processed, `ap_fixup_virtual_hosts()` is called. One of the responsibilities of this function is to merge the virtual hosts configuration objects with the base server's object. If there are virtual hosts, `merge_server_configs()` calls `modperl_config_srv_merge()` and `modperl_config_dir_merge()` for each virtual host, to perform this merge for mod_perl configuration objects.

META: is that's the place where everything restarts? it doesn't restart under debugger since we run with NODETACH I believe.

1.4.3 The open_logs Phase

After Apache processes the configuration it's time for the *open_logs* phase, executed by `ap_run_open_logs()`. mod_perl has registered the `modperl_hook_init()` hook to be called for this phase.

META: complete what happens at this stage in mod_perl

META: why is it called modperl_hook_init and not open_logs? is it because it can be called from other functions?

1.4.4 The post_config Phase

Immediately after *open_logs*, the *post_config* phase follows. Here `ap_run_post_config()` calls `modperl_hook_post_config()`

1.5 Request Processing

META: need to write

1.6 Shutdown

META: need to write

1.7 Maintainers

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

- Stas Bekman <stas (at) stason.org>

1.8 Authors

-

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

Table of Contents: