

1 Server Life Cycle Handlers

1.1 Description

This chapter discusses server life cycle and the `mod_perl` handlers participating in it.

1.2 Server Life Cycle

The following diagram depicts the Apache 2.0 server life cycle and highlights which handlers are available to `mod_perl` 2.0:

server life cycle

Apache 2.0 starts by parsing the configuration file. After the configuration file is parsed, the `PerlOpenLogsHandler` handlers are executed if any. After that it's a turn of `PerlPostConfigHandler` handlers to be run. When the *post_config* phase is finished the server immediately restarts, to make sure that it can survive graceful restarts after starting to serve the clients.

When the restart is completed, Apache 2.0 spawns the workers that will do the actual work. Depending on the used MPM, these can be threads, processes or a mixture of both. For example the *worker* MPM spawns a number of processes, each running a number of threads. When each child process is started `PerlChildInitHandler` handlers are executed. Notice that they are run for each starting process, not a thread.

From that moment on each working thread processes connections until it's killed by the server or the server is shutdown.

1.2.1 Startup Phases Demonstration Module

Let's look at the following example that demonstrates all the startup phases:

```
#file:MyApache2/StartupLog.pm
#-----
package MyApache2::StartupLog;

use strict;
use warnings;

use Apache2::Log ();
use Apache2::ServerUtil ();

use Fcntl qw(:flock);
use File::Spec::Functions;

use Apache2::Const -compile => 'OK';

my $log_path = catfile Apache2::ServerUtil::server_root,
    "logs", "startup_log";
my $log_fh;

sub open_logs {
    my ($conf_pool, $log_pool, $temp_pool, $s) = @_;
```

```

    $s->warn("opening the log file: $log_path");
    open $log_fh, ">>$log_path" or die "can't open $log_path: $!";
    my $oldfh = select($log_fh); $| = 1; select($oldfh);

    say("process $$ is born to reproduce");
    return Apache2::Const::OK;
}

sub post_config {
    my ($conf_pool, $log_pool, $temp_pool, $s) = @_;
    say("configuration is completed");
    return Apache2::Const::OK;
}

sub child_init {
    my ($child_pool, $s) = @_;
    say("process $$ is born to serve");
    return Apache2::Const::OK;
}

sub child_exit {
    my ($child_pool, $s) = @_;
    say("process $$ now exits");
    return Apache2::Const::OK;
}

sub say {
    my ($caller) = (caller(1))[3] =~ /(?:^:)+$/;
    if (defined $log_fh) {
        flock $log_fh, LOCK_EX;
        printf $log_fh "[%s] - %-11s: %s\n",
            scalar(localtime), $caller, $_[0];
        flock $log_fh, LOCK_UN;
    }
    else {
        # when the log file is not open
        warn __PACKAGE__ . " says: $_[0]\n";
    }
}

my $parent_pid = $$;
END {
    my $msg = "process $$ is shutdown";
    $msg .= "\n". "-" x 20 if $$ == $parent_pid;
    say($msg);
}

1;

```

And the *httpd.conf* configuration section:

```

<IfModule prefork.c>
    StartServers      4
    MinSpareServers  4
    MaxSpareServers  4

```

```

    MaxClients          10
    MaxRequestsPerChild 0
</IfModule>

PerlModule             MyApache2::StartupLog
PerlOpenLogsHandler    MyApache2::StartupLog::open_logs
PerlPostConfigHandler  MyApache2::StartupLog::post_config
PerlChildInitHandler   MyApache2::StartupLog::child_init
PerlChildExitHandler   MyApache2::StartupLog::child_exit

```

When we perform a server startup followed by a shutdown, the *logs/startup_log* is created if it didn't exist already (it shares the same directory with *error_log* and other standard log files), and each stage appends to that file its log information. So when we perform:

```
% bin/apachectl start && bin/apachectl stop
```

the following is getting logged to *logs/startup_log*:

```

[Sun Jun  6 01:50:06 2004] - open_logs   : process 24189 is born to reproduce
[Sun Jun  6 01:50:06 2004] - post_config: configuration is completed
[Sun Jun  6 01:50:07 2004] - END       : process 24189 is shutdown
-----
[Sun Jun  6 01:50:08 2004] - open_logs   : process 24190 is born to reproduce
[Sun Jun  6 01:50:08 2004] - post_config: configuration is completed
[Sun Jun  6 01:50:09 2004] - child_init   : process 24192 is born to serve
[Sun Jun  6 01:50:09 2004] - child_init   : process 24193 is born to serve
[Sun Jun  6 01:50:09 2004] - child_init   : process 24194 is born to serve
[Sun Jun  6 01:50:09 2004] - child_init   : process 24195 is born to serve
[Sun Jun  6 01:50:10 2004] - child_exit   : process 24193 now exits
[Sun Jun  6 01:50:10 2004] - END         : process 24193 is shutdown
[Sun Jun  6 01:50:10 2004] - child_exit   : process 24194 now exits
[Sun Jun  6 01:50:10 2004] - END         : process 24194 is shutdown
[Sun Jun  6 01:50:10 2004] - child_exit   : process 24195 now exits
[Sun Jun  6 01:50:10 2004] - child_exit   : process 24192 now exits
[Sun Jun  6 01:50:10 2004] - END         : process 24192 is shutdown
[Sun Jun  6 01:50:10 2004] - END         : process 24195 is shutdown
[Sun Jun  6 01:50:10 2004] - END         : process 24190 is shutdown
-----

```

First of all, we can clearly see that Apache always restart itself after the first *post_config* phase is over. The logs show that the *post_config* phase is preceded by the *open_logs* phase. Only after Apache has restarted itself and has completed the *open_logs* and *post_config* phase again, the *child_init* phase is run for each child process. In our example we have had the setting *StartServers=4*, therefore you can see four child processes were started.

Finally you can see that on server shutdown, the *child_exit* phase is run for each child process and the *END* { } block is executed by the parent process and each of the child processes. This is because that *END* block was inherited from the parent on fork.

However the presented behavior varies from MPM to MPM. This demonstration was performed using *prefork* mpm. Other MPMs like *winnt*, may run *open_logs* and *post_config* more than once. Also the *END* blocks may be run more times, when threads are involved. You should be very careful when designing features relying on the phases covered in this chapter if you plan support multiple MPMs. The only thing

that's sure is that you will have each of these phases run at least once.

Apache also specifies the *pre_config* phase, which is executed before the configuration files are parsed, but this is of no use to `mod_perl`, because `mod_perl` is loaded only during the configuration phase.

Now let's discuss each of the mentioned startup handlers and their implementation in the `MyApache2::StartupLog` module in detail.

1.2.2 PerlOpenLogsHandler

The *open_logs* phase happens just before the *post_config* phase.

Handlers registered by `PerlOpenLogsHandler` are usually used for opening module-specific log files (e.g., `httpd` core and `mod_ssl` open their log files during this phase).

At this stage the `STDERR` stream is not yet redirected to *error_log*, and therefore any messages to that stream will be printed to the console the server is starting from (if such exists).

This phase is of type `RUN_ALL`.

The handler's configuration scope is `SRV`.

Arguments

The *open_logs* handler is passed four arguments: the configuration pool, the logging stream pool, the temporary pool and the main server object.

The pool arguments are:

- `$conf_pool` is the main process sub-pool, therefore its life-span is the same as the main process's one. The main process is a sub-pool of the global pool.
- `$log_pool` is a global pool's sub-pool, therefore its life-span is the same as the Apache program's one.

META: what is it good for if it lives the same life as conf pool?

- `$temp_pool` is a `$conf_pool` subpool, created before the config phase, lives through the *open_logs* phase and get destroyed after the *post_config* phase. So you will want to use that pool for doing anything that can be discarded before the requests processing starts.

All three pool arguments are instances of `APR::Pool`.

`$s` is the base server object (an instance of `Apache2::ServerRec`).

Return

The handler should return `Apache2::Const::OK` if it completes successfully.

Examples

```
sub open_logs {
    my ($conf_pool, $log_pool, $temp_pool, $s) = @_;

    $s->warn("opening the log file: $log_path");
    open $log_fh, ">>$log_path" or die "can't open $log_path: $!";
    my $oldfh = select($log_fh); $| = 1; select($oldfh);

    say("process $$ is born to reproduce");
    return Apache2::Const::OK;
}
```

In our example the handler opens a log file for appending and sets the filehandle to unbuffered mode. It then logs the fact that it's running in the parent process.

As you've seen in the example this handler is configured by adding to the top level of *httpd.conf*:

```
PerlOpenLogsHandler MyApache2::StartupLog::open_logs
```

This handler can be executed only by the main server. If you want to traverse the configured virtual hosts, you can accomplish that using a simple loop. For example to print out the configured port numbers do:

```
use Apache2::ServerRec ();
# ...
sub open_logs {
    my ($conf_pool, $log_pool, $temp_pool, $s) = @_;

    my $port = $s->port;
    warn "base port: $port\n";
    for (my $vs = $s->next; $vs; $vs = $vs->next) {
        my $port = $vs->port;
        warn "vhost port: $port\n";
    }
    return Apache2::Const::OK;
}
```

1.2.3 PerlPostConfigHandler

The *post_config* phase happens right after Apache has processed the configuration files, before any child processes were spawned (which happens at the *child_init* phase).

This phase can be used for initializing things to be shared between all child processes. You can do the same in the startup file, but in the *post_config* phase you have an access to a complete configuration tree (via `Apache2::Directive`).

This phase is of type `RUN_ALL`.

The handler's configuration scope is SRV.

Arguments

Arguments are exactly as for `PerlOpenLogsHandler`.

Return

If the handler completes successfully it should return `Apache2::Const::OK`.

Examples

In our `MyApache2::StartupLog` example we used the `post_config()` handler:

```
sub post_config {
    my ($conf_pool, $log_pool, $temp_pool, $s) = @_;
    say("configuration is completed");
    return Apache2::Const::OK;
}
```

As you can see, its arguments are identical to the `open_logs` phase's handler. In this example handler we don't do much, but logging that the configuration was completed and returning right away.

As you've seen in the example this handler is configured by adding to `httpd.conf`:

```
PerlPostConfigHandler MyApache2::StartupLog::post_config
```

Everything that applies to `PerlOpenLogsHandler` identically applies to this handler.

The `add_version_component()` includes another useful example.

1.2.4 PerlChildInitHandler

The `child_init` phase happens immediately after the child process is spawned. Each child process (not a thread!) will run the hooks of this phase only once in their life-time.

In the prefork MPM this phase is useful for initializing any data structures which should be private to each process. For example `Apache::DBI` pre-opens database connections during this phase and `Apache2::Resource` sets the process' resources limits.

This phase is of type `VOID`.

The handler's configuration scope is SRV.

Arguments

The `child_init()` handler is passed two arguments: the child process pool (`APR::Pool`) and the server object (`Apache2::ServerRec`).

Return

If the handler completes successfully it should return `Apache2::Const::OK`.

Examples

In our `MyApache2::StartupLog` example we used the `child_init()` handler:

```
sub child_init {
    my ($child_pool, $s) = @_;
    say("process $$ is born to serve");
    return Apache2::Const::OK;
}
```

The example handler logs the pid of the child process it's run in and returns.

As you've seen in the example this handler is configured by adding to `httpd.conf`:

```
PerlChildInitHandler MyApache2::StartupLog::child_init
```

1.2.5 PerlChildExitHandler

Opposite to the `child_init` phase, the `child_exit` phase is executed before the child process exits. Notice that it happens only when the process exits, not the thread (assuming that you are using a threaded mpm).

This phase is of type `RUN_ALL`.

The handler's configuration scope is `SRV`.

Arguments

The `child_exit()` handler accepts two arguments: the child process pool (`APR::Pool`) and the server object (`Apache2::ServerRec`).

Return

If the handler completes successfully it should return `Apache2::Const::OK`.

Examples

In our `MyApache2::StartupLog` example we used the `child_exit()` handler:

```
sub child_exit {
    my ($child_pool, $s) = @_;
    say("process $$ now exits");
    return Apache2::Const::OK;
}
```

The example handler logs the pid of the child process it's run in and returns.

As you've seen in the example this handler is configured by adding to *httpd.conf*:

```
PerlChildExitHandler MyApache2::StartupLog::child_exit
```

1.3 Apache Command-line Commands

Some notes on how Apache start/restart Apache commands affect mod_perl.

META: not sure this is the best place for this section, but start some notes here.

Apache re-parses *httpd.conf* at least once for **each** of the following commands (and will run any mod_perl code found in it).

- **httpd -k start**

No special issues here.

Apache start and immediately restarts itself.

- **httpd -k restart**

This will abort any processed requests and restart the server.

All kind of problems could be encountered here, including segfaults and other kind of crashes. This is because when the SIGTERM signal is sent, things in process will be aborted.

Avoid using this method.

Alternatively `httpd -k restart` can be executed `kill -HUP HTTPD_PID`.

- **httpd -k graceful**

No issues here. Apache starts and restarts itself just like with `start`, but it waits for the existing requests to finish before killing them.

Alternatively `httpd -k graceful` can be executed `kill -USR1 HTTPD_PID`.

- **httpd -k stop**

Similarly to `httpd -k restart` you may encounter all kind of issues here, due to the SIGTERM signal.

1.4 mod_perl Startup

The following sections discuss the specifics of the mod_perl startup.

1.4.1 Start Immediately Restarts

As explained in the Server Life Cycle section, on start Apache normally runs the server configuration phase, followed by `PerlOpenLogsHandler` and `PerlPostConfigHandler` phases, then immediately restarts itself. Therefore everything running at the server startup is executed twice. During the restart, Perl is completely destroyed and started again.

1.4.2 When Does perl Start To Run

If Apache is started as `'httpd -t'` (equivalent to `'apachectl configtest'`) or as `'httpd -S'`, it will run only the configuration phase and exit. Depending on your configuration file, it may or may not start perl. See the details below.

During the normal startup, `mod_perl 2.0` postpones the startup of perl until after the configuration phase is over, to allow the usage of the `PerlSwitches` directive, which can't be used after Perl is started.

After the configuration phase is over, as the very first thing during the `post_config` phase, `mod_perl` starts perl and runs any registered `PerlRequire` and `PerlModule` entries.

At the very end of the `post_config` phase any registered `PerlPostConfigRequire` entries are run.

When any of the following configuration directives is encountered (during the configuration phase) `mod_perl 2.0` is forced to start as soon as they are encountered (as these options require a running perl):

- `PerlLoadModule`
- `<Perl>` section
- `PerlConfigRequire`

Therefore if you want to trigger an early Perl startup, you could add an empty `<Perl>` section in `httpd.conf`:

```
<Perl>
# trigger an early Perl startup
</Perl>
```

right after loading the `mod_perl` module, if you are using DSO, or just before your `mod_perl` configuration section, if you're using a static `mod_perl` build. But most likely you want to use the `PerlConfigRequire` instead.

1.4.3 Startup File

A startup file with Perl code to be executed at the server startup can be loaded using `PerlPostConfigRequire`. For example:

```
PerlPostConfigRequire /home/httpd/perl/lib/startup.pl
```

It's used to adjust Perl modules search paths in @INC, pre-load commonly used modules, pre-compile constants, etc. Here is a typical *startup.pl* for mod_perl 2.0:

```
#file:startup.pl
#-----

use lib qw(/home/httpd/perl);

# enable if the mod_perl 1.0 compatibility is needed
# use Apache2::compat ();

# preload all mp2 modules
# use ModPerl::MethodLookup;
# ModPerl::MethodLookup::preload_all_modules();

use ModPerl::Util (); #for CORE::GLOBAL::exit

use Apache2::RequestRec ();
use Apache2::RequestIO ();
use Apache2::RequestUtil ();

use Apache2::ServerRec ();
use Apache2::ServerUtil ();
use Apache2::Connection ();
use Apache2::Log ();

use APR::Table ();

use ModPerl::Registry ();

use Apache2::Const -compile => ':common';
use APR::Const -compile => ':common';

1;
```

In this file @INC is adjusted to include non-standard directories with Perl modules:

```
use lib qw(/home/httpd/perl);
```

If you need to use the backwards compatibility layer load:

```
use Apache2::compat ();
```

Next we preload the commonly used mod_perl 2.0 modules and precompile common constants.

Finally as usual the *startup.pl* file must be terminated with `1 ;`.

1.4.4 Dealing with Restarts

Ideally the code running at the server startup shouldn't be affected by the apache restart. If however this is not the case, you can use `Apache2::ServerUtil::restart_count`.

1.5 Maintainers

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

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

1.6 Authors

-

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

Table of Contents:

1	Server Life Cycle Handlers	1
1.1	Description	2
1.2	Server Life Cycle	2
1.2.1	Startup Phases Demonstration Module	2
1.2.2	PerlOpenLogsHandler	5
1.2.3	PerlPostConfigHandler	6
1.2.4	PerlChildInitHandler	7
1.2.5	PerlChildExitHandler	8
1.3	Apache Command-line Commands	9
1.4	mod_perl Startup	9
1.4.1	Start Immediately Restarts	10
1.4.2	When Does perl Start To Run	10
1.4.3	Startup File	10
1.4.4	Dealing with Restarts	12
1.5	Maintainers	12
1.6	Authors	12