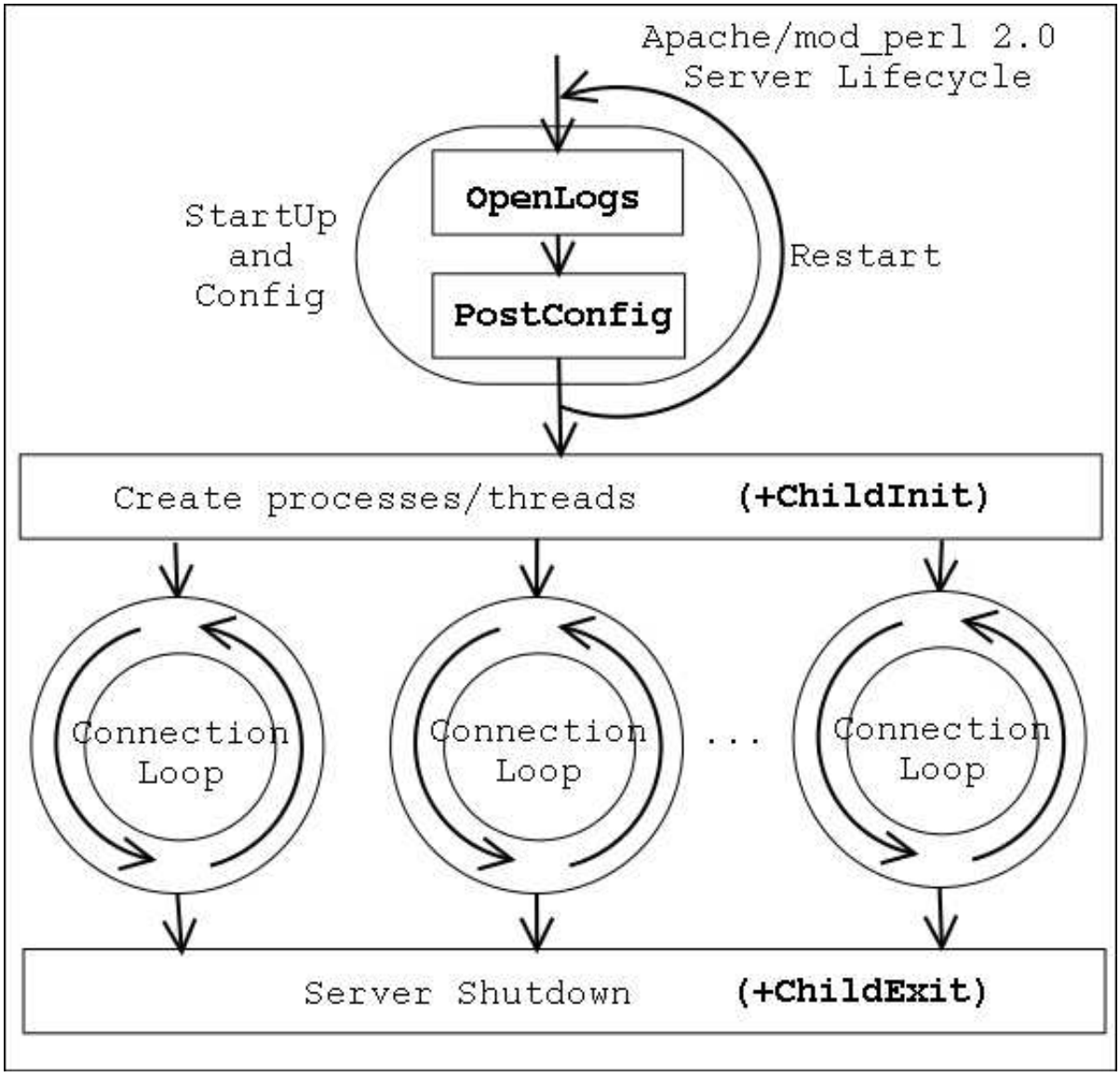
1 Server Life Cycle Handlers

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:



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 and 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 `PerlChildInit` 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:MyApache/StartupLog.pm
---------------------------
package MyApache::StartupLog;

use strict;
use warnings;

use Apache::Log ();
use Apache::ServerUtil ();

use File::Spec::Functions;

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

my $log_file = catfile "logs", "startup_log";
my $log_fh;

sub open_logs {
    my($conf_pool, $log_pool, $temp_pool, $s) = @_;
    my $log_path = Apache::server_root_relative($conf_pool, $log_file);

    $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 Apache::OK;
}

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

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

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

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

END {
    say("process $$ is shutdown\n");
}

1;
```

And the *httpd.conf* configuration section:

```
<IfModule prefork.c>
  StartServers         4
  MinSpareServers      4
  MaxSpareServers      4
  MaxClients           10
  MaxRequestsPerChild  0
</IfModule>

PerlModule            MyApache::StartupLog
PerlOpenLogsHandler   MyApache::StartupLog::open_logs
PerlPostConfigHandler MyApache::StartupLog::post_config
PerlChildInitHandler  MyApache::StartupLog::child_init
PerlChildExitHandler  MyApache::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 it its log information. So when we perform:

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

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

```
[Thu May 29 13:11:08 2003] - open_logs  : process 21823 is born to reproduce
[Thu May 29 13:11:08 2003] - post_config: configuration is completed
[Thu May 29 13:11:09 2003] - END        : process 21823 is shutdown

[Thu May 29 13:11:10 2003] - open_logs  : process 21825 is born to reproduce
[Thu May 29 13:11:10 2003] - post_config: configuration is completed
[Thu May 29 13:11:11 2003] - child_init : process 21830 is born to serve
```

```
[Thu May 29 13:11:11 2003] - child_init : process 21831 is born to serve
[Thu May 29 13:11:11 2003] - child_init : process 21832 is born to serve
[Thu May 29 13:11:11 2003] - child_init : process 21833 is born to serve
[Thu May 29 13:11:12 2003] - child_exit : process 21833 now exits
[Thu May 29 13:11:12 2003] - child_exit : process 21832 now exits
[Thu May 29 13:11:12 2003] - child_exit : process 21831 now exits
[Thu May 29 13:11:12 2003] - child_exit : process 21830 now exits
[Thu May 29 13:11:12 2003] - END        : process 21825 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 only.

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 `MyApache::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`.

As we have seen in the `MyApache::StartupLog::open_logs` handler, the *open_logs* phase handlers accept four arguments: the configuration pool, the logging stream pool, the temporary pool and the server object:

```
sub open_logs {
    my($conf_pool, $log_pool, $temp_pool, $s) = @_;
    my $log_path = Apache::server_root_relative($conf_pool, $log_file);

    $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 Apache::OK;
}
```

In our example the handler uses the function `Apache::server_root_relative()` to set the full path to the log file, which is then opened for appending and set to unbuffered mode. Finally it 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 *httpd.conf*:

```
PerlOpenLogsHandler MyApache::StartupLog::open_logs
```

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 `Apache::Directive`).

This phase is of type `RUN_ALL`.

The handler's configuration scope is `SRV`.

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

```
sub post_config {
    my($conf_pool, $log_pool, $temp_pool, $s) = @_;
    say("configuration is completed");
    return Apache::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 MyApache::StartupLog::post_config
```

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 `Apache::Resource` sets the process' resources limits.

This phase is of type `VOID`.

The handler's configuration scope is `SRV`.

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

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

The *child_init()* handler accepts two arguments: the child process pool and the server object. 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  MyApache::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`.

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

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

The *child_exit()* handler accepts two arguments: the child process pool and the server object. 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  MyApache::StartupLog::child_exit
```

1.3 Maintainers

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

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

1.4 Authors

-

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

Table of Contents: