Asynchronous web application with PSGI and Twiggy

PSGI | Plack::Request | AnyEvent | Twiggy

Prev Next (pro)

If our web application has requests that take a relatively long time process, and if there are several clients connecting at the same time, they can easily tie up all the process of the web server. Creating more workers is possible, and as our benchmark shows it can provide good performance, but each such worker is a separate process. They require both memory allocation and processing power.

An alternative is to use an asynchronous web server, such as Twiggy, that can handle a lot of request in a singple process.

Of course we can't just start using Twiggy, for this to work well, we'll also have to rewrite the application.

Echo using Asynchronous programming

This is the simple echo application, with the 2 seconds delay added in this version, rewritten using AnyEvent to be used with Twiggy.

The anonymous subroutine assigned to `$app` will run for every request. The line `my $request = Plack::Request->new($env);` gets the current request object and then we check if the `field` parameter has been supplied. If not, we immediately return (after the if-block) with the simple content returned by the `get_html` function.

```
#!/usr/bin/perl
use strict;
use warnings;

use Plack::Request;

my $app = sub {
    my $env = shift;

    my $html = get_html();

    my $request = Plack::Request->new($env);
    if ($request->param('field')) {
        return sub {
            my $response = shift;
```

```
                my $t;
                $t = AnyEvent->timer(after => 2, cb => sub {
                    undef $t;
                    $html .= 'You said: ' . $request->param('field') . '<br>';
                    return $response->([
                        '200',
                        [ 'Content-Type' => 'text/html' ],
                        [ $html ],
                    ]);
                });
            };
        }

        return [
            '200',
            [ 'Content-Type' => 'text/html' ],
            [ $html ],
        ];
    };

    sub get_html {
        return q{
          <form>

          <input name="field">
          <input type="submit" value="Echo">
          </form>
          <hr>
        }
}
```

The interesting part is what happens when the "field" has a true value and we enter the if-block. In there, we create and return an anonymous function. Once that function is returned, the application has sort of two directions. On one hand it has finished to handle the current request and it is ready to handle the next request, on the other hand, it has not sent a response yet and thus the connection to the client is still alive. In that sense it has not finished to handle the request yet.

The function itself creates an Asynchronous time object that will execute its call-back (supplied a the value of `cb`, `after 2 seconds`. We don't use `sleep` as in the earlier example, as that would cause the application to be irresponsive for 2 seconds. Instead we ask AnyEvent to call our call-back 2 seconds later. The call-back itself will create the 3-element response required by PSGI.

The variable `$t` holds the timer object. It needs a bit more attention than usual variables. It has to exists already before the call to the `timer` method, hence we declare it with `my $t;` in a separate

statement, and it has to be destroyed when the timer is finished. Hence we call `undef $t;` inside the call-back of the timer.

Running and benchmarking

Let's save it in a file called `async_echo_delay.psgi` and run it using `twiggy async_echo_delay.psgi`.

It does not print anything on the console, but we can already access it via http://127.0.0.1:5000/

We can also use ApacheBench to measure the response time of the server as we did in this article.

We send 200 request in parallel. The reports that the whole benchmark takes 2.043 seconds, meaning that using a single process and a single thread, Twiggy was able to handle all the request in parallel.

```
$ ab -n 200 -c 200 http://127.0.0.1:5000/?field=hello
This is ApacheBench, Version 2.3 <$Revision: 1554214 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient)
Completed 100 requests
Completed 200 requests
Finished 200 requests


Server Software:
Server Hostname:        127.0.0.1
Server Port:            5000

Document Path:          /?field=hello
Document Length:        131 bytes

Concurrency Level:      200
Time taken for tests:   2.043 seconds
Complete requests:      200
Failed requests:        0
Total transferred:      35000 bytes
HTML transferred:       26200 bytes
Requests per second:    97.89 [#/sec] (mean)
Time per request:       2043.044 [ms] (mean)
Time per request:       10.215 [ms] (mean, across all concurrent requests)
Transfer rate:          16.73 [Kbytes/sec] received
```

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        3    6   1.7      5       9
Processing:  2005 2021   8.2   2021    2033
Waiting:     2005 2021   8.2   2021    2033
Total:       2010 2026   9.2   2024    2038

Percentage of the requests served within a certain time (ms)
  50%   2024
  66%   2035
  75%   2036
  80%   2036
  90%   2037
  95%   2038
  98%   2038
  99%   2038
 100%   2038 (longest request)
```