Routing with PSGI

PSGI

Prev Next

For any web application one of the critical questions is how the various URLs are handled? Are they mapped to subroutines in a single file, like in Dancer, or are they mapped to subroutines in modules as it is done inCatalyst.

As PSGI is the low level building block for any modern web application in Perl, you can write your own routing. In this article we'll see a simple case of routing.

We create the following script that we call `echo_routing.psgi` and we run it using `plackup echo_routing.psgi` . If we browse to `http://localhost:5000/` we see an input box and a button. We type in "hello" and click on the button. The URL changes to `http://localhost:5000/echo?field=hello` and we get back: `You said: hello` .

If we click on the "Back" button of the browser, clear the input box and click on the button again the URL changes to `http://localhost:5000/echo?field=` and the browser displays: `You did not say anything.` .

If we now change the URL to be `http://localhost:5000/abc` , the browser will display `404 Not Found` .

Details of the usage

Please, note after clicking on the button the URL changes. Both the actual path that now includes the word "echo", and it also has some parameters after it. The parameters are just as they were in the earlier echo example. The new thing is the change in the path. This happens because in the `form` in the HTML below (in the get_html function) we added an `action="/echo"` attribute. By default a form will submit to the same URL where it came from, but the `action` attribute can control the URL.

The code

For each route we create a subroutine. `server_root` will serve the request to `/` and `serve_echo` will serve the request to `/echo` . If you look at the code, it is quite straight forward returning the 3-element array ref as required by the PSGI specification. It is explained in the getting started with PSGI article.

The interesting part is that we have created a hash called `%ROUTING` which is a dispatch-table. The keys are the possible routes, the values are references to functions handling the give route.

The main part of the code is the anonymous subroutine assigned to `$app` . We use the Plack::Request module to fetch the `path_info` , which is essentially the route. We check if it is an existing key in the `%ROUTING` hash and if it has a

value: `my $route = $ROUTING{$request->path_info};`. The variable `$route` will either be `undef` if the route the user accessed was not defined (as is in the case of `/abc`), or it will be a reference to a function. If it is a reference to a function we de-reference the variable and call the function. Using the `$route->($env)` syntax we also pass the `$env` variable as the first parameter.

If the `path_info` did not match any of the keys in the routing table, we return the HTTP Error code 404 Not Found.

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

use Plack::Request;

my %ROUTING = (
    '/'      => \&serve_root,
    '/echo'  => \&serve_echo,
);


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

    my $request = Plack::Request->new($env);
    my $route = $ROUTING{$request->path_info};
    if ($route) {
        return $route->($env);
    }
    return [
        '404',
        [ 'Content-Type' => 'text/html' ],
        [ '404 Not Found' ],
    ];
};

sub serve_root {
    my $html = get_html();
    return [
        '200',
        [ 'Content-Type' => 'text/html' ],
        [ $html ],
    ];
}
```

```
    sub serve_echo {
        my $env = shift;

40.     my $request = Plack::Request->new($env);
41.     my $html;
        if ($request->param('field')) {
            $html = 'You said: ' . $request->param('field');
        } else {
45.         $html = 'You did not say anything.';
        }
        return [
            '200',
            [ 'Content-Type' => 'text/html' ],
50.         [ $html ],
51.     ];
    }

    sub get_html {
55.     return q{
          <form action="/echo">

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