{ MAGNUM SOLUTIONS LIMITED }

Modern Perl Web Development Day 1 - PSGI

Dave Cross
Magnum Solutions Ltd
dave@mag-sol.com

Schedule

- 09:45 – Begin
- 11:15 – Coffee break (15 mins)
- 13:00 – Lunch (60 mins)
- 14:00 – Begin
- 15:30 – Coffee break (15 mins)
- 17:00 – End

What We Will Cover

- Perl and the Web
- PSGI and Plack
- Plack Middleware
- Plack Apps

What We Will Cover

- Debugging PSGI Apps
- Testing PSGI Apps
- Deploying PSGI Apps
- Switching to PSGI

What We Will Cover

- Today we will cover the basics of PSGI and Plack
- Tomorrow we will look at web frameworks in more detail

{MAGNUM
SOLUTIONS LIMITED}

Perl and the Web

Web Development

- People have been developing web applications for over 20 years
- Surely it is easy now
- Lessons have been learned
- Best practices have been worked out

History of Perl & Web

- Common Gateway Interface 1993
- Defined the interaction between a web server and a program
- Dynamic web pages

CGI

- Request includes parameters
- Program processes parameters and produces response
- Response includes program's output

Mid-Late 1990s

- Every web site gained dynamic pages
- Form to email
- Guestbook
- Hit counter
- Etc...
- Most of them were written in Perl

CGI Problems

- CGI can be slow
- Perl compiler starts up on every request
- Can be very slow
- Not useful for heavy traffic sites

mod_perl

- Everyone used Apache
- Apache allowed loadable modules
- mod_perl loads a Perl compiler
- Persistent Perl processes
- No more start-up costs
- Huge performance improvements

Downsides

- Can't just use your CGI programs
  - ModPerl::Registry
- Program is now called as a subroutine
- Global variable issues
- Many programs needed rewrites
- Different input and output methods

Other Environments

- FastCGI
- Microsoft IIS
- nginx
- Etc...
- Lack of portability
- Hold that thought

CGI Programs

- CGI programs do three things
- Read user input
- Process data
- Produce output
- Let's look at input and output in more detail

Output

- CGI programs produce two types of output
- Headers
  - Content-type
- Body
  - The actual data (HTML, etc)

Simple CGI Output

- ```
  #!/usr/bin/perl

  print “Content-type: text/plain\n\n”;
  print 'The time is: ',
        scalar localtime;
  ```
  ```

HTML

- ```
  #!/usr/bin/perl

  print “Content-type: text/html\n\n”;
  my $time = localtime;
  print <<END_HTML;
  <html><head><title>Time</title></head>
  <body><h1>Time</h1>
  The time is: $time. </body></html>
  END_HTML
  ```
  ```

Enter CGI.pm

- CGI.pm standard part of Perl distribution
  - Until 5.22
- Handles CGI processing for you
- Input and output
- Output in the form of CGI & HTML helper functions
  - HTML helper functions now deprecated

HTML With CGI.pm

```
#!/usr/bin/perl
use CGI ':standard';

print header; # default text/html
my $time = localtime;
print start_html(title => 'Time'),
      h1('Time'),
      p(“The time is: $time”);
      end_html;
```

Downsides

- Mixing HTML and Perl code is nasty
- What if you have a designer?
- Front-end programmers don't always know Perl
- Use a templating system instead

Template Toolkit

```
<html>
  <head>
    <title>Time</title>
  </head>
  <body>
    <h1>Time</h1>
    <p>The time is: [% time %].</p>
  </body>
</html>
```

Template Toolkit

```
<html>
  <head>
    <title>Time</title>
  </head>
  <body>
    <h1>Time</h1>
    <p>The time is: [% time %].</p>
  </body>
</html>
```

Template Toolkit

- Separate the HTML into a separate file
- Use tags where the variable output goes
- Easier to edit by your HTML team

Template Toolkit & CGI

- ```
  #!/usr/bin/perl

  use Template;
  use CGI 'header';
  print header;
  my $tt = Template->new;
  my $time = localtime;
  $tt->process('time.tt',
                  { time => $time }
    or die $tt->error;
  ```
  ```

User Input

- Users send input to CGI programs
- Parameters encoded in the URL
- `http://example.com/cgi-bin/stuff?name=davorg&lang=Perl`
- Need to access these parameters
- N.B. I'm deliberately ignoring POST requests for simplicity

Old Style

```
# DON'T USE THIS!
@pairs = split /&/, $ENV{QUERY_STRING};

foreach $pair (@pairs) {
  ($k, $v) = split /=/, $pair;
  $k =~ tr/+/ /;
  $k =~ s/%([a-f0-9]{2})/pack 'C', hex($1)/ieg;
  $v =~ tr/+/ /;
  $v =~ s/%([a-f0-9]{2})/pack 'C', hex($1)/ieg;
  $form{$k} = $v;
}

# And then later...
my $name = $form{name};
```

CGI.pm Style

- ```
  use CGI ':standard';

  my $name = param('name');
  ```
  ```

However

- mod_perl has a different method for accessing parameters
- Apache2::Request
- Other environments have different methods
- This is where PSGI comes in

PSGI & Plack

PSGI/Plack

- *"PSGI is an interface between Perl web applications and web servers, and Plack is a Perl module and toolkit that contains PSGI middleware, helpers and adapters to web servers."*
  - http://plackperl.org/

PSGI/Plack

- PSGI is a specification (based on Python's WSGI)
- Plack is a reference implementation and toolbox (based on Ruby's Rack)

The Problem

- There are many ways to write web applications
- There are many ways to write web applications in Perl
- Each is subtly different
- Hard to move an application between server architectures

Server Architectures

- CGI
- FastCGI
- mod_perl
- etc...

Frameworks

- There are many Perl web application frameworks
- Each creates applications in subtly different ways
- Hard to port applications between them

The Goal

- What if we had a specification that allowed us to easily move web applications between server architectures and frameworks
- PSGI is that specification

Separation

- We know the advantages of separating processing from display
- PSGI separates processing from deployment

PSGI Application

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

   return [
     200,
     [ 'Content-Type', 'text/plain' ],
     [ 'Hello World' ],
   ];
};
```

PSGI Application

- A code reference
- Passed a reference to an environment hash
- Returns a reference to a three-element array
  - Status code
  - Headers
  - Body

A Code Reference

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

    return [
      200,
      [ ‘Content-Type’, ‘text/plain’ ],
      [ ‘Hello World’ ],
    ];
  };
```

A Code Reference

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

    return [
      200,
      [ 'Content-Type', 'text/plain' ],
      [ 'Hello World' ],
    ];
};
```

Environment Hash

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

    return [
      200,
      [ 'Content-Type', 'text/plain' ],
      [ 'Hello World' ],
    ];
};
```

Environment Hash

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

    return [
      200,
      [ 'Content-Type', 'text/plain' ],
      [ 'Hello World' ],
    ];
};
```

Return Array Ref

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

   return [
     200,
     [ ‘Content-Type’, ‘text/plain’ ],
     [ ‘Hello World’ ],
   ];
};
```

Return Array Ref

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

    return [
      200,
      [ 'Content-Type', 'text/plain' ],
      [ 'Hello World' ],
    ];
};
```

Return Array Ref

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

    return [
        200,
        [ 'Content-Type', 'text/plain' ],
        [ 'Hello World' ],
    ];
};
```

Return Array Ref

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

   return [
     200,
     [ ‘Content-Type’, ‘text/plain’ ],
     [ ‘Hello World’ ],
   ];
};
```

Return Array Ref

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

    return [
      200,
      [ 'Content-Type', 'text/plain' ],
      [ 'Hello World' ],
    ];
};
```

Running PSGI App

- Put code in app.psgi
- Drop in into a configured PSGI-aware web server
- Browse to URL

PSGI-Aware Server

- Plack contains a simple test server called plackup
- ```
  $ plackup app.psgi
  HTTP::Server::PSGI: Accepting connections at
  http://localhost:5000/
  ```
  ```

More About $env

```
use Data::Dumper;

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

  return [
    200,
    [ 'Content-type', 'text/plain' ],
    [ Dumper $env ],
  ];
}
```

More About $env

- ```
  $VAR1 = {
            'psgi.streaming' => 1,
            'psgi.multithread' => '',
            'HTTP_UPGRADE_INSECURE_REQUESTS' => '1',
            'SERVER_PROTOCOL' => 'HTTP/1.1',
            'psgi.errors' => *::STDERR,
            'PATH_INFO' => '/',
            'HTTP_ACCEPT_LANGUAGE' => 'en-GB,en-US;q=0.8,en;q=0.6',
            'psgi.multiprocess' => '',
            'SERVER_NAME' => 0,
            'psgi.version' => [ 1, 1 ],
            'psgix.input.buffered' => 1,
            'psgi.input' => *{'HTTP::Server::PSGI::$input'},
            'psgi.url_scheme' => 'http',
            'REQUEST_URI' => '/',
            'REMOTE_PORT' => 59948,
            'HTTP_ACCEPT' =>
  'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8',
            'REQUEST_METHOD' => 'GET',
            'psgix.io' => bless( *Symbol::GEN1,
  'IO::Socket::INET' ),
            'psgi.run_once' => '',
            'SCRIPT_NAME' => '',
            'HTTP_ACCEPT_ENCODING' => 'gzip, deflate, sdch',
            'SERVER_PORT' => 5000,
            'HTTP_HOST' => 'localhost:5000',
  ```
  ```

Plack::Request

- Plack::Request turns the environment into a request object

Plack::Request

```
use Plack::Request;
use Data::Dumper;

my $app = sub {

  my $req = Plack::Request->new(shift);
  return [
    200,
    [ 'Content-type', 'text/plain' ],
    [ Dumper $req ],
  ];
}
```

Plack::Request

- ```
  $VAR1 = bless( {
    'env' => {
      'SCRIPT_NAME' => '',
      'psgi.nonblocking' => '',
      'psgi.errors' => *::STDERR,
      'SERVER_PROTOCOL' => 'HTTP/1.1',
      'HTTP_USER_AGENT' =>
        'Wget/1.16.3 (linux-gnu)',
      'psgi.url_scheme' => 'http',
      'REMOTE_PORT' => 57218,
      'SERVER_PORT' => 5000,
      'REQUEST_METHOD' => 'GET',
      'REQUEST_URI' => '/',
      'psgi.version' => [ 1, 1 ],
      'psgix.io' =>
        bless( *Symbol::GEN1, 'IO::Socket::INET' ),
  ```
  ```

Plack::Request

-
```
      'HTTP_ACCEPT_ENCODING' => 'identity',
      'psgix.input.buffered' => 1,
      'psgi.run_once' => '',
      'psgi.streaming' => 1,
      'HTTP_ACCEPT' => '*/*',
      'HTTP_CONNECTION' => 'Keep-Alive',
      'psgi.input' =>
        \*{'HTTP::Server::PSGI::$input'},
      'psgi.multiprocess' => '',
      'psgi.multithread' => '',
      'QUERY_STRING' => '',
      'HTTP_HOST' => 'localhost:5000',
      'psgix.harakiri' => 1,
      'PATH_INFO' => '/',
      'SERVER_NAME' => 0
    }
  }, 'Plack::Request' );
```

Plack::Response

- Plack::Response builds a PSGI response object

Plack::Response

```
use Plack::Request;
use Plack::Response;
use Data::Dumper;

my $app = sub {
   my $req = Plack::Request->new(shift);
   my $res = Plack::Response->new(200);
   $res->content_type('text/plain');
   $res->body(Dumper $req);
   return $res->finalize;
}
```

Time Example

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

    my $res = Plack::Response->new(200);
    $res->content_type('text/plain');
    $res->body(scalar localtime);
    return $res->finalize;
};
```

Time With HTML

```
my $app = sub {
  my $env = shift;
  my $now = localtime;
  my $res = Plack::Response->new(200);
  $res->content_type('text/html');
  $res->body(
    "<html>
     <head><title>Time</title></head>
     <body><h1>Time</h1><p>The time is $now</p>
     </body>
     </html>"
  );
  return $res->finalize;
};
```

Time With TT

```
use Template;
my $app = sub {
   my $tt = Template->new;
   my $out;
   my $res = Plack::Response->new(200);
   $res→content_type('text/html');
   $tt->process('time.tt',
                { time => scalar localtime },
                \$out) or die $tt->error;
   $res->body($out);
   return $res->finalize;
};
```

User Input

- Get user input from two places
- Parse the query string from the $env hash
- Ask the Plack::Request object

Input

```
use Plack::Request;
use Data::Dumper;
my $app = sub {
  my $req = Plack::Request->new(shift);
  my $content;
  if (keys %{$req->parameters}) {
    $content = response($req);
  } else {
    $content = form();
  }

  return [ 200, [ 'Content-type', 'text/html' ],
           [ $content ], ];
};
```

Displaying a Form

```
sub form {
  return <<END_OF_FORM;
<html>
... stuff...
  <form>
    <input name=”name”>
    ... stuff ...
  </form>
</html>
END_OF_HTML
}
```

Displaying the Response

```
sub response {
  my $req = shift;
  my $name    = $req->parameters->{name};
  return <<END_OF_HTML
<html>
... stuff ...
<p>Name: $name</p>
... stuff ...
</html>
END_OF_HTML
}
```

Using Templates

- Both form and response can be produced using TT
- This is left as an exercise for the reader

Your Turn

- Try out some of the simple Plack applications from the previous slides
- Convert the last example to using templates

Building Applications

- A PSGI program can be an entire application
- The secret is in the 'path_info' input
- $env->{PATH_INFO}
- /
- /person/1
- /person/1/delete

Building Applications

```
my $app = sub {
  my $env = shift;
  my $req = Plack::Request->new($env);

  my $response = get_response_for(
    $req->path_info;
  );

  return $response;
}
```

Building Applications

```
use Some::Web::App;

my $webapp = Some::Web::App->new;

my $app = sub {
  my $env = shift
  my $resp = $webapp->get_response_for(
    $req->path_info($env);
  );

  return $response;
}
```

Building Applications

- ```
  Use Some::Web::App;

  return Some::Web::App->to_app;
  ```
  ```

Frameworks

- At this point you should probably look at a framework
- Catalyst
- Dancer2
- Mojolicious
- etc...

Dancer2 Example

- Use `dancer2` to create skeleton app
- `dancer2 gen -a MyWebApp`
- Creates many files
- `MyWebApp/bin/app.psgi`

Dancer2 app.psgi

- ```
  #!/usr/bin/env perl

  use strict;
  use warnings;
  use FindBin;
  use lib "$FindBin::Bin/../lib";

  use MyApp;
  MyApp->to_app;
  ```
  ```

Running Dancer2 App

- The main program is in MyWebApp/bin/app.psgi
- `plackup MyWebApp/bin/app.psgi`
- Browse to http://localhost:5000/

Web Frameworks

- We will return to web frameworks in more detail tomorrow

Your Turn

- Get a basic Dancer app up and running

Plack Middleware

Middleware

- Middleware wraps around an application
- Returns another PSGI application
- Simple spec makes this easy
- Plack::Middleware::*
- Plack::Builder adds middleware configuration language

Middleware

- ```
  package Plack::Middleware::Foo;
  use parent qw( Plack::Middleware );

  sub call {
      my($self, $env) = @_;
      # Do something with $env

      # $self->app is the original app
      my $res = $self->app->($env);

      # Do something with $res
      return $res;
  }
  ```
  ```

Middleware Example

```
package Plack::Middleware::Runtime;
use strict;
use parent qw(Plack::Middleware);
use Plack::Util;
use Plack::Util::Accessor qw(header_name);
use Time::HiRes;

sub call {
    my($self, $env) = @_;
    my $start = [ Time::HiRes::gettimeofday ];
    my $res = $self->app->($env);
    my $header = $self->header_name || 'X-Runtime';
    $self->response_cb($res, sub {
        my $res = shift;
        my $req_time = sprintf '%.6f',
                       Time::HiRes::tv_interval($start);
        Plack::Util::header_set($res->[1], $header, $req_time);
    });
}
```

Middleware Example

```
use Plack::Builder;
use Plack::Middleware::Runtime;

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

  return [
    200,
    [ 'Content-type', 'text/plain' ],
    [ 'Hello world' ],
  ]
};

builder {
  enable 'Runtime';
  $app;
}
```

Middleware Example

- ```
  $ HEAD http://localhost:5000
  200 OK
  Date: Sun, 06 Dec 2015 14:15:11 GMT
  Server: HTTP::Server::PSGI
  Content-Length: 11
  Content-Type: text/plain
  Client-Date: Sun, 06 Dec 2015 14:15:11 GMT
  Client-Peer: 127.0.0.1:5000
  Client-Response-Num: 1
  X-Runtime: 0.000082
  ```
  ```

Middleware Example

- ```
  $ HEAD http://localhost:5000
  200 OK
  Date: Sun, 06 Dec 2015 14:15:11 GMT
  Server: HTTP::Server::PSGI
  Content-Length: 11
  Content-Type: text/plain
  Client-Date: Sun, 06 Dec 2015 14:15:11 GMT
  Client-Peer: 127.0.0.1:5000
  Client-Response-Num: 1
  X-Runtime: 0.000082
  ```
  ```

Middleware Examples

- Many more examples included with Plack
- Plack::Middleware::AccessLog
- Plack::Middleware::ErrorDocument
- Plack::Middleware::Auth::Basic
- Plack::Middleware::Static
- Etc...

Plack::Middlware::Static

- Bypass app for static files
- ```
  use Plack::Builder;
  builder {
      enable "Plack::Middleware::Static",
          path => qr{^/(images|js|css)/},
          root => './htdocs/';
      $app;
  };
  ```
  ```

Your Turn

- Add some Plack Middleware to your application
- Edit the app.psgi file

{MAGNUM
SOLUTIONS LIMITED}

Plack Apps

Plack::App::*

- Ready-made solutions for common situations
- Plack::App::CGIBin
  - Cgi-bin replacement
- Plack::App::Directory
  - Serve files with directory index
- Plack::App::URLMap
  - Map apps to different paths

Plack::App::*

- Many more bundled with Plack
- Configured using Plack::Builder

Plack::App::CGIBin

- ```
  use Plack::App::CGIBin;
  use Plack::Builder;

  my $app = Plack::App::CGIBin->new(
    root => '/var/www/cgi-bin'
  )->to_app;

  builder {
    mount '/cgi-bin' => $app;
  };
  ```
  ```

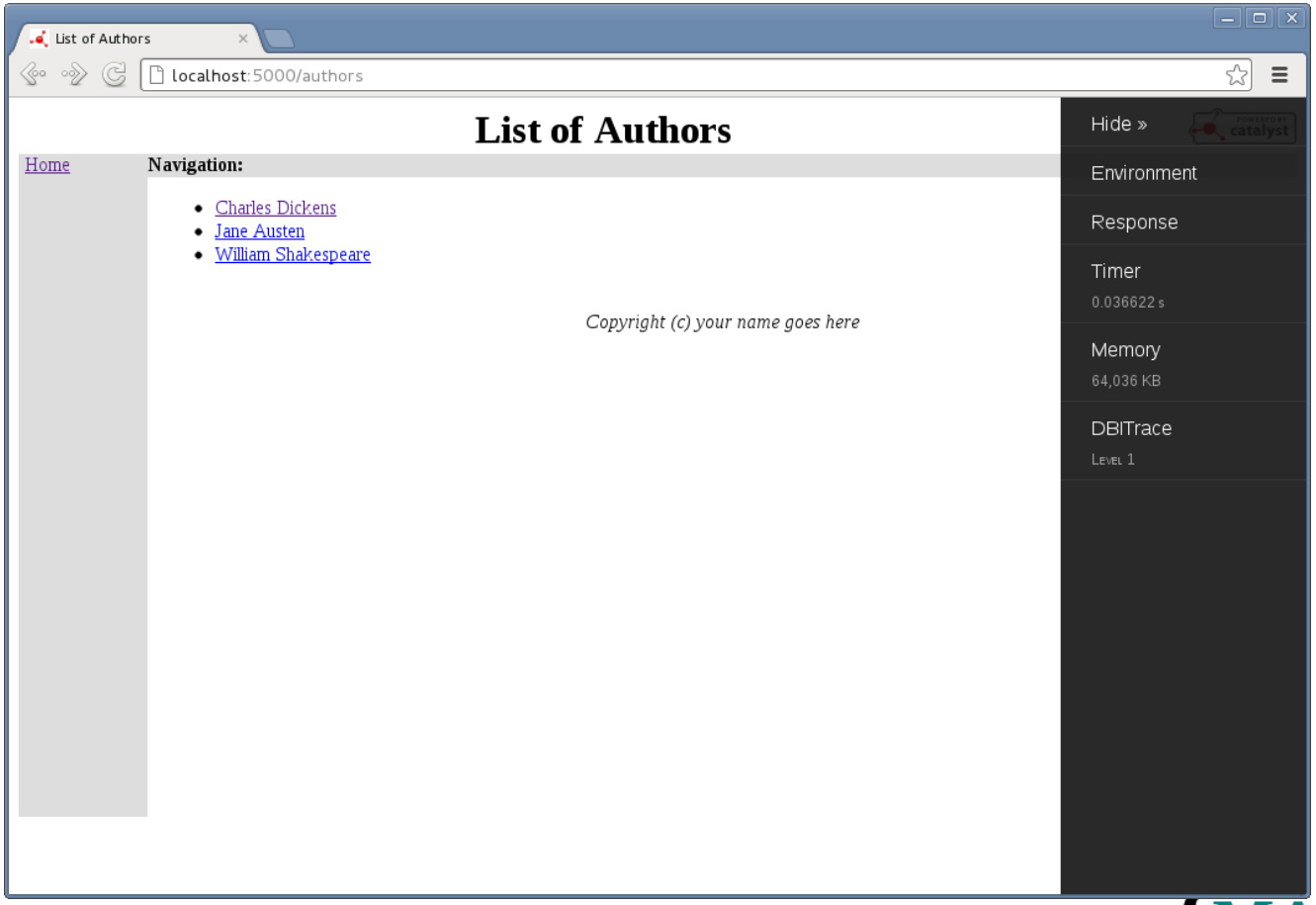
Plack::App::Directory

- ```
  use Plack::App::Directory;

  my $app = Plack::App::Directory->new(
    root => '/home/dave/Dropbox/psgi'
  )->to_app;
  ```
  ```

Your Turn

- Try setting up a few of the Plack apps that are bundled with Plack

Debugging PSGI Apps

Debugging Web Apps

- Debugging web apps is difficult
- Hard to trace execution
- Log messages

Plack::Middleware::Debug

- Adds a very useful debug panel to your application
- ```
  #!/usr/bin/env perl
  use strict
  use warnings;
  use Plack::Builder;
  use Literature; # Dancer app

  my $app = Literature->to_app;

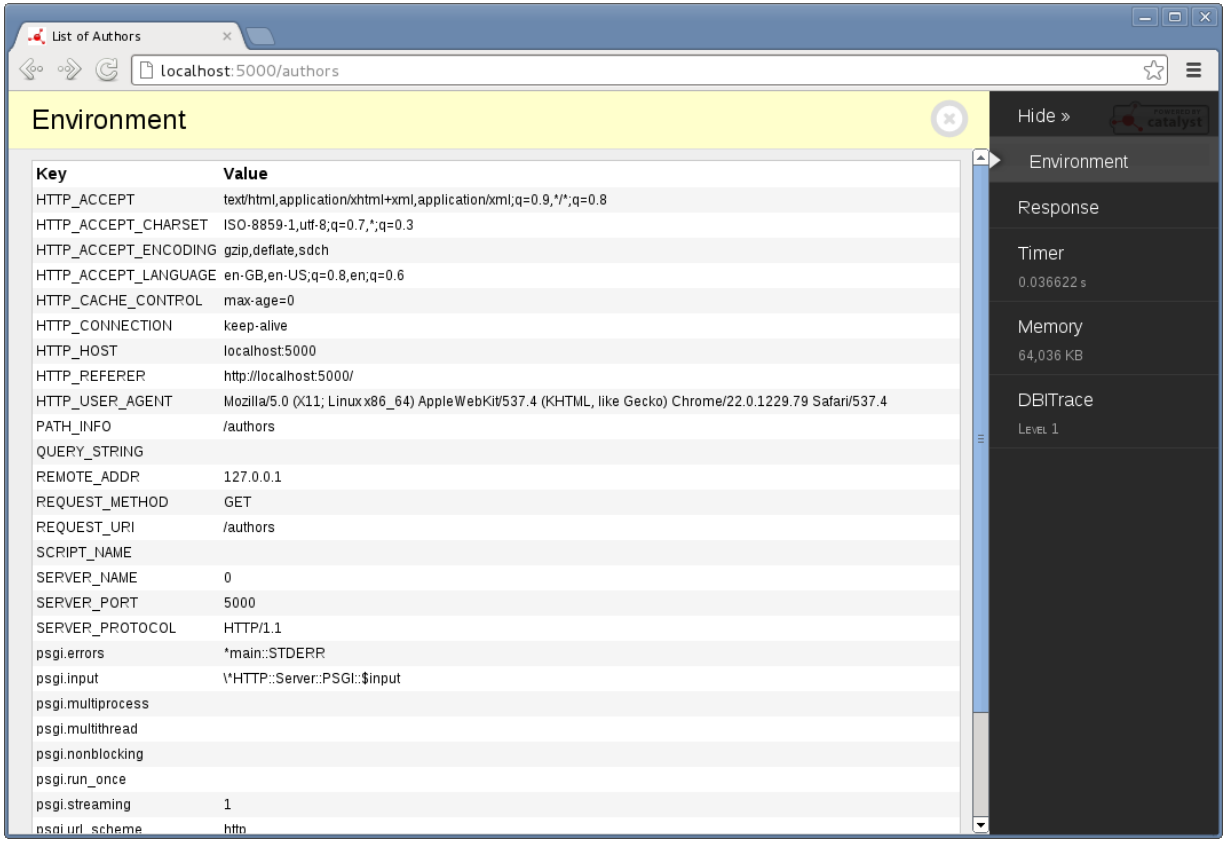
  builder {
    enable 'Debug';
    $app;
  }
  ```
  ```

Plack::Middleware::Debug

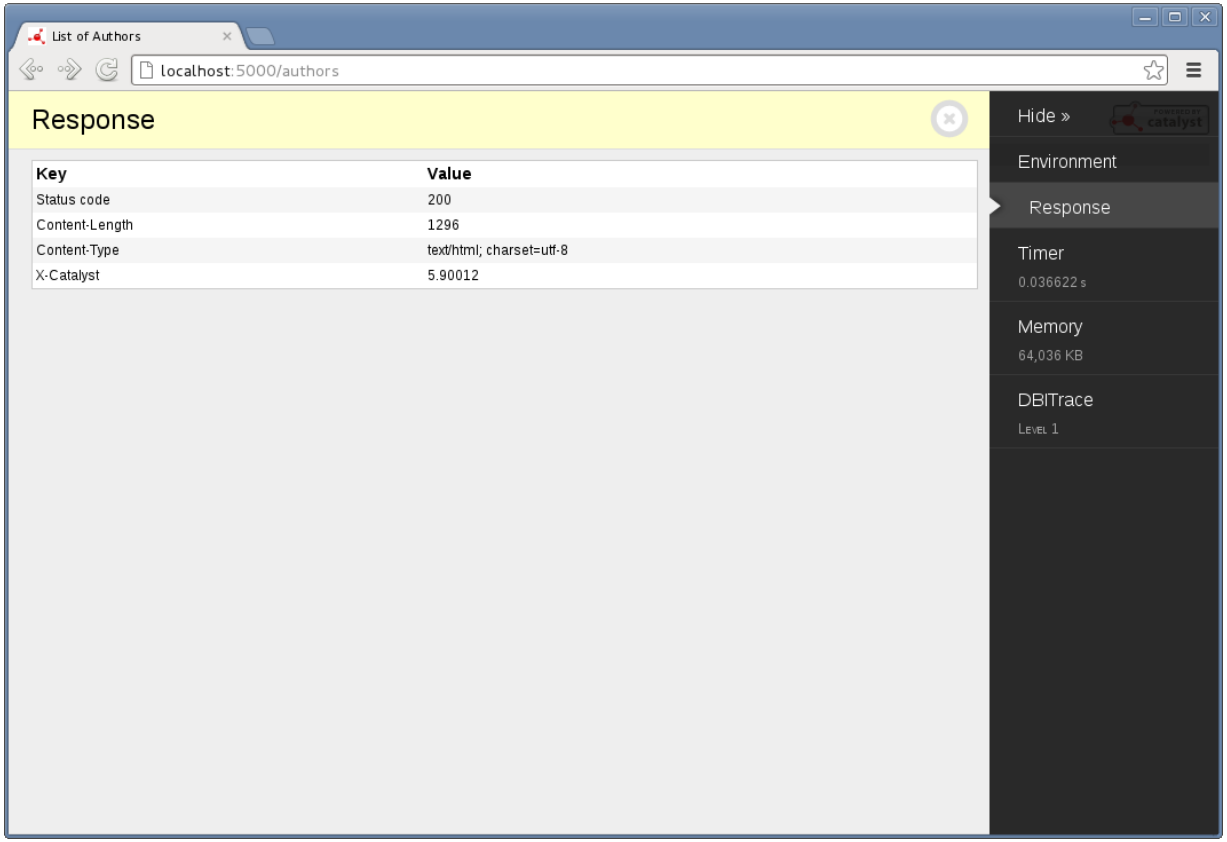


MAGNUM SOLUTIONS LIMITED

Plack::Middleware::Debug

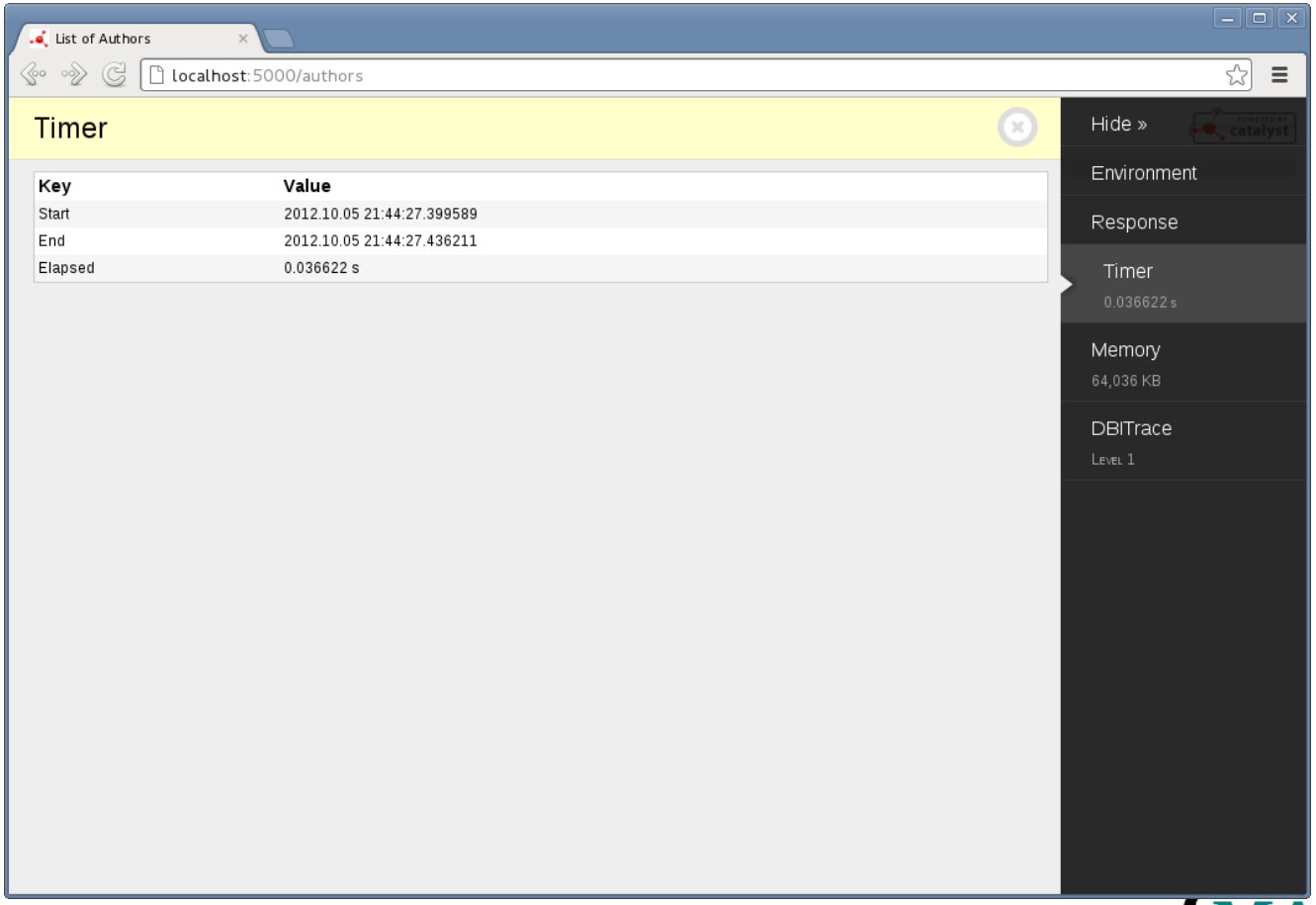


Plack::Middleware::Debug



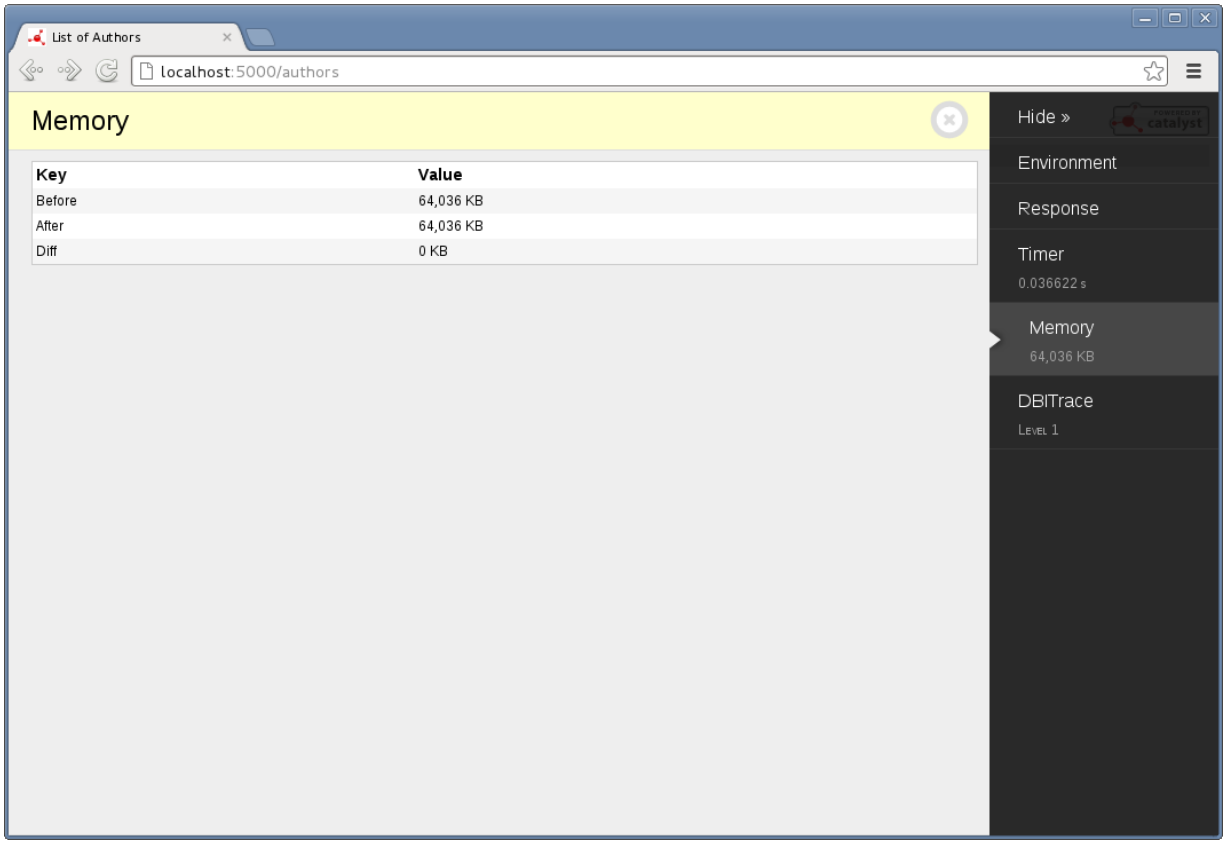
MAGNUM SOLUTIONS LIMITED

Plack::Middleware::Debug

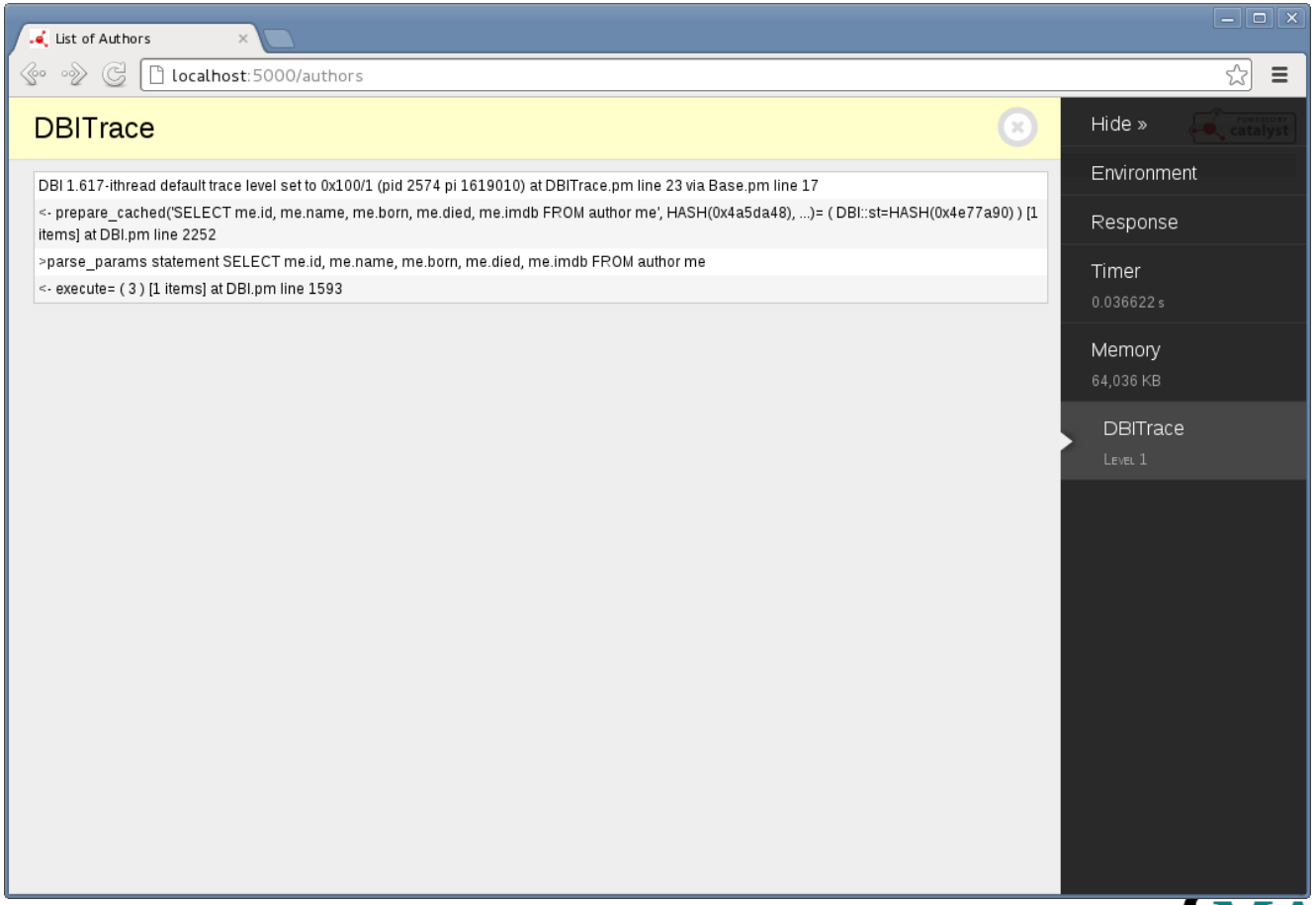


MAGNUM SOLUTIONS LIMITED

Plack::Middleware::Debug



Plack::Middleware::Debug



MAGNUM SOLUTIONS LIMITED

Plack::Debugger

- Plack::Debugger is a replacement for Plack::Middleware::Debug
- Harder to configure
- More flexible
- Work in progress
- Worth watching

Your Turn

- Add the debug panel to your application
  - Note: It only works on HTML output

Testing PSGI Apps

Testing Web Apps

- Testing web apps is hard
- For the same reasons as debugging
- WWW::Mechanize
- Selenium

Plack::Test

- The simple Plack specification makes it easy to write a testing layer
- Plack::Test
- Standard for all PSGI apps
- Dancer2::Test (for example) is now deprecated

Using Plack::Test

- Three modes of use
- Examples all require the following
- ```
  use Plack::Test;
  use HTTP::Request::Common;
  ```
  ```

Positional Params

```
my $app = sub {
  return [ 200, [], [ "Hello "] ]
};
my $client = sub {
    my $cb  = shift;
    my $res = $cb->(GET "/");
    is $res->content, "Hello";
};

test_psgi $app, $client;
```

Named Params

```
my $app = sub {
  return [ 200, [], [ "Hello "] ]
};

my $client = sub {
  my $cb  = shift;
  my $res = $cb->("GET /");
  is $res->content, “Hello”;
}

test_psgi app    => $app,
          client => $client;
```

Object Oriented

- ```
  my $app = sub {
    return [ 200, [], [ "Hello "] ]
  };

  my $test = Plack::Test->create($app);

  my $res = $test->request(GET "/");
  is $res->content, "Hello";
  ```
  ```

Deploying PSGI Apps

Deploying PSGI Apps

- PSGI separates implementation from deployment
- This is a major advantage
- Concentrate on the right problems at the right time
- Easily switch between deployment environments

PSGI Server Support

- Many new web servers support PSGI
- Starman, Starlet, Twiggy, Corona, HTTP::Server::Simple::PSGI
- Perlbal::Plugin::PSGI

PSGI Server Support

- nginx support
- mod_psgi
- Plack::Handler::Apache2

Deploy as CGI

- ```
  #!/usr/bin/plackup

  use Plack;

  my $app = sub {
    return [ 200,
      [ 'Content-Type' => 'text/plain' ],
      [ 'Hello World' ] ];
  };
  ```
  ```

Deploy as CGI

- `#!/usr/bin/plackup`

```
use Plack;

my $app = sub {
   return [ 200,
     [ 'Content-Type' => 'text/plain' ],
     [ 'Hello World' ] ];
};
```

Deploy as CGI

- Any PSGI program can be deployed using CGI
- Just use `#!/usr/bin/plackup`
- Not recommended
- PSGI positives
- CGI negatives

Plack::Handler::Apache2

- ```
  <Location /psgi>
    SetHandler perl-script
    PerlResponseHandler Plack::Handler::Apache2
    PerlSetVar psgi_app /path/to/app.psgi
  </Location>
  ```
  ```

Deploy Under nginx

- Use plackup to start the app
- plackup -S Starman app.psgi
  - Starman is a high-performance preforking PSGI/Plack web server
- Configure nginx to proxy requests to port 5000

nginx Configuration

```
Server {
  location / {
      proxy_set_header Host $http_host;
      proxy_set_header X-Forwarded-Host $http_host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For
                        $proxy_add_x_forwarded_for;
      proxy_pass http://localhost:5000;
    }
}
```

Your Turn

- Get some PSGI apps running in different deployment environments

Switching to PSGI

Switching To PSGI

- How do you move legacy CGI code to PSGI?
- Do you need to?
- Replace on rewrite?
- Write new code using PSGI
- Eventually replace old code
- Can take a long time

Why Switch?

- PSGI advantages
- Middleware
- Easier debugging
- Flexible deployment

Easiest Switch

- CGI::Emulate::PSGI
- Emulates a CGI environment inside a PSGI environment
- ```
  my $app = CGI::Emulate::PSGI->handler(sub {
      # Existing CGI code
  });
  ```
  ```

Easiest Switch

- CGI::PSGI
- A PSGI-compatible CGI.pm replacement
- Some rewriting required
- ```
  use CGI::PSGI;
  my $app = sub {
      my $env = shift;
      my $q = CGI::PSGI→new($env);
      return [ $q->psgi_header, [ $body ] ];
  };
  ```
  ```

Rewriting Code

- Use Plack to rewrite simple code
- Use `#!/usr/bin/plackup`
- Put code in `sub { … }`
- Use Plack::Request to access input
- Build up output in a variable
- Return PSGI output at the end

Rewriting Example

```
use CGI ':standard';
my $time_cookie = cookie(-name=>'time',
                         -value=>scalar localtime,
                         -expires=>'+1y');

print header(-cookie => $time_cookie),
      start_html(-title=>'Cookie test'),
      h1('Cookie test');

if (my $time = cookie('time')) {
  print p("You last visited this page at $time");
} else {
  print p("You haven't visited this page before");
}
print end_html;
```

Rewriting Example

```
use Plack::Request;
use Plack::Response;
use HTML::Tiny;

my $app = sub {
  my $time =
    Plack::Request→new(shift)->cookies->{time};
  my $time_text;
  if ($time) {
    $time_text =
      "You last visited this page at $time";
  } else {
    $time_text =
      "You haven't visited this page before";
  }
```

Rewriting Example

```
my $res  = Plack::Response→new(200);

$res->cookies->{time} = {
    value   => scalar localtime,
    expires => '+1y',
  };

$res->content_type('text/html');
```

Rewriting Example

```
my $h = HTML::Tiny->new;

my $body = $h->html([
                $h→head(
                  $h->title('Cookie Test'),
                ),
                $h->body([
                  $h->h1('Cookie Test'),
                  $h->p($time_text),
                ]),
              ]);
```

Rewriting Example

```
•    $res→body($body);  return $res→
     return $res->finalize;
   };
```

More Examples

- See http://perlhacks.com/2016/01/easy-psgi/
- https://github.com/davorg/easy-psgi

Your Turn

- Find an old CGI program
- Wrap it in CGI::Emulate::PSGI
- Rewrite it as a PSGI program
- Add middleware to both versions

Further Information

- perldoc PSGI
- perldoc Plack
- http://plackperl.org/
- http://blog.plackperl.org/
- http://github.com/miyagawa/Plack
- #plack on irc.perl.org

That's all folks

- Any questions?