Plack
Superglue for Perl Web Frameworks

Tatsuhiko Miyagawa
YAPC::NA 2010
Tatsuhiko Miyagawa

- Lives in San Francisco
- Software Engineer @ Six Apart
- http://search.cpan.org/~miyagawa/
- @miyagawa
- http://bulknews.typepad.com/
Web Applications
Hello World
#!/usr/bin/perl
use strict;

print "Content-Type: text/plain\n\n\n";
print "Hello World";}
use FCGI;

my $req = FCGI::Request();
while ($req->Accept >= 0) {
    print "Content-Type: text/plain\n\n\n";
    print "Hello World";
}

package HelloWorld;
use strict;
use Apache::RequestRec;
use Apache::RequestIO;
use Apache::Const -compile => qw(OK);

sub handler {
    my $r = shift;

    $r->content_type('text/plain');
    $r->print("Hello World");

    return Apache::Const::OK;
}

1;
package HelloWorld;
use base qw(HTTP::Server::Simple::CGI);

sub handle_request {
    my($self, $cgi) = @_;  
    print "HTTP/1.0 200 OK\n";
    print "Content-Type: text/plain\n\n";
    print "Hello World";
}

1;
All similar
but slightly different
Painful to support all of them :(
There is (was) one common way to do all of this.
#!/usr/bin/perl
use CGI;

my $q = CGI->new;

print $q->header('text/plain);
print "Hello World";
Happens to work on:

CGI, FastCGI, mod_perl
(HTTP::Server::Simple::CGI)
CGI.pm?

meh
Frameworks to the rescue!
Maypole Mason Mojo Sledge Catalyst Spoon PageKit AxKit Egg Gantry Continuity Solstice Mojolicious Tripletail Konstrukt Reaction Jifty Cyclone3 WebGUI OpenInteract Squatting Dancer CGI::Application Nanoa Ark Angelos Noe Schenker Tatsumaki Amon Apache2::WebApp Web::Simple Apache2::REST SweetPea Hydrant Titanium
MANY
web frameworks
Let’s look how they handle web servers.
Gross.
CGI.pm
  Jifty, CGI::Application, Spoon

_mod_perl centric
  Mason, Sledge, PageKit, WebGUI

Adapters
  Catalyst, Maypole, Squatting
That was 2008.
Steal great idea from Python/Ruby
WSGI (Python)
Rack (Ruby)
WSGI (PEP-333)
PEP: 333
Title: Python Web Server Gateway Interface v1.0
Version: 71593
Author: Philip J. Eby <pje at telecommunity.com>
Discussions-To: Python Web-SIG <web-sig at python.org>
Status: Draft
Type: Informational
Content-Type: text/x-rst
Created: 07-Dec-2003

Contents

- Abstract
- Rationale and Goals
- Specification Overview
  - The Application/Framework Side
  - The Server/Gateway Side
  - Middleware Components that Play Both Sides
- Specification Details
  - environ Variables
    - Input and Error Streams
    - The make_scriptable Callable
    - Handling the Content-Length Header
  - Buffering and Streaming
    - Middleware Handling of Block Boundaries
    - The write Callable
  - Unicode Issues
    - String literal forms
# WSGI

def hello(environ, start_response):
    start_response("200 OK", [
        (‘Content-Type’, ‘text/plain’)  
    ])
    return [“Hello World”]
WSGI

- Django
- Bottle
- CherryPy
- Tornado
- Pylons
- Flask
- mod_wsgi
- Paste
- gunicorn
- uWSGI
- wsgiref
- Google AppEngine
Rack
Rack: a Ruby Webserver Interface

Rack provides a minimal interface between webservers supporting Ruby and Ruby frameworks.

News

October 18th, 2009
Rack 1.0.1 has been released!

April 25th, 2009
Rack 1.0.0 has been released!

January 9th, 2009
Rack 0.9.1 has been released! This is a security release, please update.

January 6th, 2009
Rack 0.9 has been released!

December 24th, 2008
Introducing the Rack Core Team

Read the FAQ
# Rack

class Hello
  def call(env)
    return [
      200,
      { "Content-Type" => "text/plain" },
      ["Hello World"]
    ]
  end
end
Rack

- Rails
- Merb
- Sinatra
- Camping
- Ramaze
- etc.

- Unicorn
- Thin
- Mongrel
- Rainbows!
- Phusion Passenger
- Heroku
PSGI
Perl Web Server Gateway Interface
Interface
# WSGI

def hello(environ, start_response):
    start_response("200 OK", [
        (‘Content-Type’, ‘text/plain’)
    ])
    return [“Hello World”]
# Rack

class Hello
  def call(env)
    return [200,
      {
        "Content-Type" => "text/plain"
      },
      ["Hello World"]
    ]
  end
end
# PSGI
my $app = sub {
    my $env = shift;
    return [
        200,
        ['Content-Type', 'text/plain'],
        ['Hello World'],
    ];
};
PSGI application

code reference
$app = sub {...};
my $app = sub {
    my $env = shift;
    return [ $status, $header, $body ];
};

CGI-like environment variables
+ psgi.input, psgi.errors etc.
my $app = sub {
    my $env = shift;
    return [ $status, $header, $body ];
};

Status code (int.): 200, 404 etc.
my $app = sub {
    my $env = shift;
    return [ $status, $header, $body ];
};

Array reference of header pairs:
[ ‘Content-Type’, ‘text/html’, … ]
my $app = sub {
    my $env = shift;
    return [ $status, $header, $body ];
};

Array reference of content chunks
Filehandle or IO::Handle-ish object
That's it.

(There's a callback based streaming interface as well)
# PSGI
my $app = sub {
    my $env = shift;
    return [
        200,
        [ 'Content-Type', 'text/plain' ],,
        [ 'Hello World' ],
    ];
};
Now you’ve got a PSGI application.
PSGI makes it so simple to:
Write a new Perl web app & framework
Write a new Perl web server
PSGI adaptation
Maypole Mason Mojo Sledge Catalyst Spoon PageKit AxKit Egg Gantry Continuity Solstice Mojolicious Tripletail Konstrukt Reaction Jifty Cyclone3 WebGUI OpenInteract Squatting Dancer CGI::Application Nanoa Ark Angelos Noe Schenker Tatsumaki Amon Apache2::WebApp Web::Simple Apache2::REST SweetPea Hydrant Titanium
Maypole Mason Mojo Sledge Catalyst Spoon PageKit
AxKit Egg Gantry Continuity Solstice Mojolicious
Tripletail Konstrukt Reaction Jifty Cyclone3 WebGUI
OpenInteract Squatting Dancer CGI::Application
Nanoa Ark Angelos Noe Schenker Tatsumaki Amon
Apache2::WebApp Web::Simple Apache2::REST
SweetPea Hydrant Titanium
Applications

Movable Type 6, WebGUI 8
# Catalyst
use MyApp;

MyApp->setup_engine(‘PSGI’);
my $app = sub { MyApp->run(@_) };

# $app is a PSGI app!
# Jifty
use MyPonyApp;
my $app = MyPonyApp->psgi_app;

# $app is a PSGI app!
# Dancer
use Dancer;

get ‘/’ => sub {
    “Hello World”;
};

dance; # returns a PSGI app!
# Mojolicious::Lite
use Mojolicious::Lite;

get ‘/:name’ => sub {
    my $self = shift;
    $self->render_text(‘Hello!’);
};

app->start; # returns PSGI app
# Web::Simple
use Web::Simple 'MyApp';

class MyApp {
  use Web::Simple::Strategy;

  dispatch {
    sub(GET) {
      [ 200, [...] ], [ 'Hello' ];
    }
  }
}

my $app = MyApp->as_psgi;

# $app is a PSGI app!
perldoc PSGI

If you want to adapt your framework to PSGI.
Plack

“PSGI toolkit”
HTTP::Server::PSGI

Reference PSGI web server bundled in Plack
Plackup

Runs PSGI app instantly from CLI
(inspired by rackup)
Plack::Handler

Connects PSGI apps to Web servers
CGI, FastCGI, Apache, SCGI
PSGI Web Servers
Starman

UNIX Preforking HTTP servers (like Unicorn.rb)
HTTP/1.1 chunk + keep-alives / Very Fast
Starlet

Simpler UNIX HTTP/1.0 Server
Best used with Server::Starter and nginx/lighttpd
Twiggy

Non-blocking web server (like Thin.rb)
based on AnyEvent framework
Corona

Coroutine for each connection based on Coro.pm
HTTP::Server::Simple::PSGI

Zero-deps other than HTTP::Server::Simple
Best for embedding PSGI applications
uWSGI

http://projects.unbit.it/uwsgi/
Perlbal plugin

http://github.com/miyagawa/Perlbal-Plugin-PSGI
nginx embedded perl

http://github.com/yappo/nginx-psgi-patchs
mod_psgi

http://github.com/spiritloose/mod_psgi
evpsgi

http://github.com/sekimura/evpsgi
Feersum

http://github.com/stash/Feersum
Middleware
PSGI Middleware

Wraps a PSGI application to add pre/post processing
my $app = sub {
    my $env = shift;
    return [ $status, $header, $body ];
};

my $mw = sub {
    my $env = shift;
    # do something with $env
    my $res = $app->($env);
    # do something with $res;
    return $res;
};
PSGI Middleware

coderef -> coderef
Higher-order functions
Middleware

Debug, Session, Logger, Runtime, Static, Lint,
AccessLog, ConditionalGET, ErrorDocument,
StackTrace, Auth::Basic, Auth::Digest, ReverseProxy,
Refresh, Auth::OAuth, Hippie, Throttle
Plack::Middleware

reusable and extensible
Middleware framework
Plack::Builder DSL in .psgi
my $app = sub {
    return [ $status, $header, $body ];
};

use Plack::Builder;

builder {
    enable "Static", root => "/htdocs",
    path => qr!^/static/!;
    enable "Deflater"; # gzip/deflate
    $app;
}
plackup compatible

plackup -e 'enable "Foo"; app.psgi
Middleware

Write once, run in every framework
Plack::App::URLMap

Multiplex multiple apps
Integrated with Builder DSL
use CatApp;
use CGIApp;

my $c1 = sub { CatApp->run }
my $c2 = sub { CGIApp->run_psgi }

use Plack::Builder;

builder {
    mount "/cat" => $c1;
    mount "/cgi-app" => builder {
        enable "StackTrace";
        $c2;
    }
}
CGI::PSGI

Easy migration from CGI.pm
CGI::Emulate::PSGI
CGI::Compile

Easiest migration from CGI scripts (like Registry)
Plack::Request

like libapreq (Apache::Request)
wrapper APIs for middleware developers
Plack::Test

Unified interface to write tests with Mock HTTP and Live HTTP
use Plack::Test;
use HTTP::Request::Common;

my $app = sub {
    my $env = shift;
    return [ $status, $header, $body ];
};

test_psgi app => $app, client => sub {
    my $cb = shift;
    my $req = GET "http://localhost/foo";
    my $res = $cb->($req);
    # test $res;
};
use Plack::Test;
use HTTP::Request::Common;
$Plack::Test::Impl = "Server";

my $app = sub {
    my $env = shift;
    return [ $status, $header, $body ];
};

test_psgi app => $app, client => sub {
    my $cb = shift;
    my $req = GET "http://localhost/foo";
    my $res = $cb->($req);
    # test $res;
};
Plack::App::*
ready-to-use applications
Plack::App::Directory

Static content file server
Plack::App::Proxy

(non-blocking) proxy server
Can be used as reverse proxy as well
Plack::App::CGIBin

mount /cgi-bin as PSGI applications
Plack::App::FCGIDispatcher

Connect to FCGI daemon
(even in Ruby, Python, C)
Plack::App::JSP

Runs JavaScript PSGI apps :)

Testimonials

a.k.a. self-promotion
“I love this! It’s exactly the right answer to what I was looking for.” - Benjamin Trott
“Wow, this is nothing short of awesome.”
- Stevan Little
“Plack is so simple, easy and powerful.”
- Jesse Vincent
“Plack rocks. Miyagawa rocks.”
- Piers Cawley
“Is he on too many drugs or too few? Either way let’s make sure that never changes.”
- Matt S. Trout
Real World
Plack/PSGI
You should join!
> cpanm Plack
> cpanm Task::Plack
Cloud?
(Heroku, GAE)
Sunaba
http://sunaba.plackperl.org/
Sunaba
Sunaba runs your Plack/PSGI apps in the cloud.

```perl
my $app = sub {
  my $env = shift;
  [ 200, [ "Content-Type", "text/plain" ], [ "Hello World" ] ];
};
```

Sunaba is an experimental service powered by Plack, Twiggy and Tatsumaki. Oh, and it's open source running on a VPS box of Tatsuriko Mysigawa. Sandbox per environment is powered by Dan Rogal's levv Ajax API and all of its restrictions apply. Services can be interrupted, shutdown or blocked at any time at their own will. NO WARRANTY. Use at your own risk.
Runs on dankogai’s Sandbox
You can even try:

```
system("rm -fr /");
while (1) {
    
}
```
Summary

- PSGI is an interface, Plack is the code.
- We have many (pretty fast) PSGI servers.
- We have adapters and tools for most web frameworks.
- Use it!
Remember:
Plack is < 1y old
Lots of (exciting) things going on.
http://github.com/miyagawa/Plack
http://plackperl.org/
irc://irc.perl.org/#plack
Questions?
Thank you!

Slides: http://slideshare.net/miyagawa