

Advanced Perl Techniques Day 2

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

Resources

- Slides available on-line
 - <http://mag-sol.com/train/public/2012-02/ukuug>
- Also see Slideshare
 - <http://www.slideshare.net/davorg/slideshows>
- Get Satisfaction
 - <http://getsatisfaction.com/magnum>

What We Will Cover

- Modern Core Perl
 - What's new in Perl 5.10, 5.12 & 5.14
- Advanced Testing
- Database access with DBIx::Class
- Handling Exceptions

What We Will Cover

- Profiling and Benchmarking
- Object oriented programming with Moose
- MVC Frameworks
 - Catalyst
- PSGI and Plack

Benchmarking & Profiling



Benchmarking

- Ensure that your program is fast enough
- But how fast is fast enough?
- *premature optimization is the root of all evil*
 - Donald Knuth
 - paraphrasing Tony Hoare
- Don't optimise until you know what to optimise

Benchmark.pm

- Standard Perl module for benchmarking
- Simple usage
- use Benchmark;
my %methods = (
 method1 => sub { ... },
 method2 => sub { ... },
);
timethese(10_000, \%methods);
- Times 10,000 iterations of each method

Benchmark.pm Output

- Benchmark: timing 10000 iterations of method1, method2...
method1: 6 wallclock secs \
(2.12 usr + 3.47 sys = 5.59 CPU) \
@ 1788.91/s (n=10000)
method2: 3 wallclock secs \
(0.85 usr + 1.70 sys = 2.55 CPU) \
@ 3921.57/s (n=10000)

Timed Benchmarks

- Passing `timethese` a positive number runs each piece of code a certain number of times
- Passing `timethese` a negative number runs each piece of code for a certain number of seconds

Timed Benchmarks

- use Benchmark;
my %methods = (
 method1 => sub { ... },
 method2 => sub { ... },
);

Run for 10,000(!) seconds
timethese(-10_000, \%methods);

Comparing Performance

- Use `cmpthese` to get a tabular output
- Optional export
- use Benchmark 'cmpthese';

```
my %methods = (  
    method1 => sub { ... },  
    method2 => sub { ... },  
);  
cmpthese(10_000, \%methods);
```

cmpthese Output

- | | Rate | method1 | method2 |
|---------|-----------|---------|---------|
| method1 | 2831802/s | -- | -61% |
| method2 | 7208959/s | 155% | -- |
- method2 is 61% slower than method1
- Can also pass negative number to cmpthese

Benchmarking is Hard

- Very easy to produce lots of numbers
- Harder to ensure that the numbers are meaningful
- Compare code fragments that do the same thing

Bad Benchmarking

- use Benchmark qw{ timethese };
timethese(1_000, {
 Ordinary => sub {
 my @results = sort { -M \$a <=> -M \$b }
 glob "/bin/*";
 },
 Schwartzian => sub {
 map \$_->[0],
 sort { \$a->[1] <=> \$b->[1] }
 map [\$_, -M], glob "/bin/*";
 },
});

What to Benchmark

- Profile your code
- See which parts it is worth working on
- Look for code that
 - Takes a long time to run, or
 - Is called many times, or
 - Both

Devel::DProf

- Devel::DProf is the standard Perl profiling tool
- Included with Perl distribution
- Uses Perl debugger hooks
- `perl -d:DProf your_program`
- Produces a data file called `tmon.out`
- Command line program `dprofpp` to view results

Sample Output

- ```
$ perl -d:DProf ./invoice.pl 244
$ dprofpp
Total Elapsed Time = 1.173152 Seconds
 User+System Time = 0.963152 Seconds
Exclusive Times
%Time ExclSec CumulS #Calls sec/call Csec/c Name
6.02 0.058 0.067 482 0.0001 0.0001 Params::Validate::_validate
5.09 0.049 0.114 7 0.0070 0.0163 Class::DBI::Loader::mysql::BEGIN
4.15 0.040 0.050 10 0.0040 0.0050 Template::Parser::BEGIN
4.15 0.040 0.166 7 0.0057 0.0237 DateTime::Locale::BEGIN
4.05 0.039 0.094 43 0.0009 0.0022 base::import
3.74 0.036 0.094 449 0.0001 0.0002 DateTime::Locale::_register
3.11 0.030 0.280 4 0.0074 0.0700 DateTime::Format::MySQL::BEGIN
2.91 0.028 0.028 170 0.0002 0.0002 Lingua::EN::Inflect::_PL_noun
2.70 0.026 0.040 1 0.0262 0.0401 Template::Parser::_parse
2.49 0.024 0.024 1113 0.0000 0.0000 Class::Data::Inheritable::__ANON__
2.08 0.020 0.020 12 0.0017 0.0017 DBD::mysql::db::_login
2.08 0.020 0.020 4 0.0050 0.0050 Template::Stash::BEGIN
2.08 0.020 0.099 9 0.0022 0.0110 Template::Config::load
2.08 0.020 0.067 9 0.0022 0.0074 Template::BEGIN
2.08 0.020 0.039 4 0.0049 0.0097 Lingua::EN::Inflect::Number::BEGIN
```

# Devel::NYTProf

- New profiling module
- Based on work from the New York Times
- Enhanced by Tim Bunce
- Pretty HTML output
  - “borrowed” from Devel::Cover
- Far more flexible
- Far more powerful

# Using NYTProf

- Similar to Devel::DProf
- `$ perl -d:NYTProf ./invoice.pl 244`
- Writes `nytprof.out`
- `$ nytprofhtml`
- Or
- `$ nytprofcsv`

# Using NYTProf

- (Demo)

# Conclusions

- Don't optimise until you know you need to optimise
- Don't optimise until you know what to optimise
- Use profiling to find out what is worth optimising
- Use benchmarking to compare different solutions

# More Information

- perldoc Benchmark
- perldoc Devel::DProf
- perldoc Devel::NYTProf
- Chapters 5 and 6 of *Mastering Perl*

# Benchmarking Examples

- Profile the code that you have been given
- Where do we need to make improvements
- Suggest some alternative implementations
- Benchmark your suggestions



# Moose



# Moose

- *A complete modern object system for Perl 5*
- Based on experiments with Perl 6 object model
- Built on top of Class::MOP
  - MOP - Meta Object Protocol
  - Set of abstractions for components of an object system
  - Classes, Objects, Methods, Attributes
- An example might help

# Moose Example

- ```
package Point;
use Moose;

has 'x' => (isa => 'Int',
            is  => 'ro');
has 'y' => (isa => 'Int',
            is  => 'rw');

sub clear {
    my $self = shift;
    $self->{x} = 0;
    $self->y(0);
}
```

Understanding Moose

- There's a lot going on here
- use Moose
 - Loads Moose environment
 - Makes our class a subclass of Moose::Object
 - Turns on strict and warnings

Creating Attributes

- `has 'x' => (isa => 'Int',
 is => 'ro')`
 - Creates an attribute called 'x'
 - Constrained to be an integer
 - Read-only accessor
- `has 'y' => (isa => 'Int',
 is => 'rw')`

Defining Methods

- ```
sub clear {
 my $self = shift;
 $self->{x} = 0;
 $self->y(0);
}
```
- Standard method syntax
- Uses generated method to set y
- Direct hash access for x

# Subclassing

- ```
package Point3D;  
use Moose;  
  
extends 'Point';  
  
has 'z' => (isa => 'Int');  
  
after 'clear' => sub {  
    my $self = shift;  
    $self->{z} = 0;  
};
```

Subclasses

- extends 'Point'
 - Similar to use base
 - Overwrites @ISA instead of appending
- has 'z' => (isa = 'Int')
 - Adds new attribute 'z'
 - No accessor function - private attribute

Extending Methods

- ```
after 'clear' => sub {
 my $self = shift;
 $self->{z} = 0;
};
```
- New clear method for subclass
- Called after method for superclass
- Cleaner than `$self->SUPER::clear()`

# Creating Objects

- Moose classes are used just like any other Perl class
- `$point = Point->new(x => 1, y => 2);`
- `$p3d = Point3D->new(x => 1,  
y => 2,  
z => 3);`

# More About Attributes

- Use the `has` keyword to define your class's attributes
- `has 'first_name' => ( is => 'rw' );`
- Use `is` to define `rw` or `ro`
- Omitting `is` gives an attribute with no accessors

# Getting & Setting

- By default each attribute creates a method of the same name.
- Used for both getting and setting the attribute
- `$dave->first_name( 'Dave' );`
- `say $dave->first_name;`

# Change Accessor Name

- Change accessor names using reader and writer
- ```
has 'name' => (  
  is => 'rw',  
  reader => 'get_name',  
  writer => 'set_name',  
);
```
- See also `MooseX::FollowPBP`

Required Attributes

- By default Moose class attributes are optional
- Change this with `required`

Required Attributes

- has 'name' => (
 is => 'ro',
 required => 1,
);
- Forces constructor to expect a name
- Although that name could be undef

Attribute Defaults

- Set a default value for an attribute with default
- `has 'size' => (
 is => 'rw',
 default => 'medium',
);`

Attribute Defaults

- Can use a subroutine reference
- has 'size' => (
 is => 'rw',
 default => \&rand_size,
);

More Attribute Properties

- `lazy`
 - Only populate attribute when queried
- `trigger`
 - Subroutine called after the attribute is set
- `isa`
 - Set the type of an attribute
- Many more

More Moose

- Many more options
- Support for concepts like delegation and roles
- Powerful plugin support
 - MooseX::*
- Lots of work going on in this area

Moose Examples

- Write a Moose class that models a person
- Write code which uses that class
- Try out various attribute properties

Catalyst



MVC Frameworks

- MVC frameworks are a popular way to write applications
 - Particularly web applications

M, V and C

- Model
 - Data storage & data access
- View
 - Data presentation layer
- Controller
 - Business logic to glue it all together

MVC Examples

- Ruby on Rails
- Django (Python)
- Struts (Java)
- CakePHP
- Many examples in most languages
- Perl has many options

MVC in Perl

- Maypole
 - The original Perl MVC framework
- CGI::Application
 - Simple MVC for CGI programming
- Jifty
 - Developed and used by Best Practical
- Catalyst
 - Currently the popular choice

Newer MVC in Perl

- Dancer
- Squatting
- Mojolicious

Catalyst

- MVC framework in Perl
- Building on other heavily-used tools
- Model uses DBIx::Class
- View uses Template Toolkit
- These are just defaults
- Can use anything you want

Simple Catalyst App

- Assume we already have model
 - CD database from DBIx::Class section
- Use `catalyst.pl` to create project
- ```
$ catalyst.pl CD
created "CD"
created "CD/script"
created "CD/lib"
created "CD/root"
... many more ...
```

# What Just Happened?

- Catalyst just generated a lot of useful stuff for us
- Test web servers
  - Standalone and FastCGI
- Configuration files
- Test stubs
- Helpers for creating models, views and controllers

# A Working Application

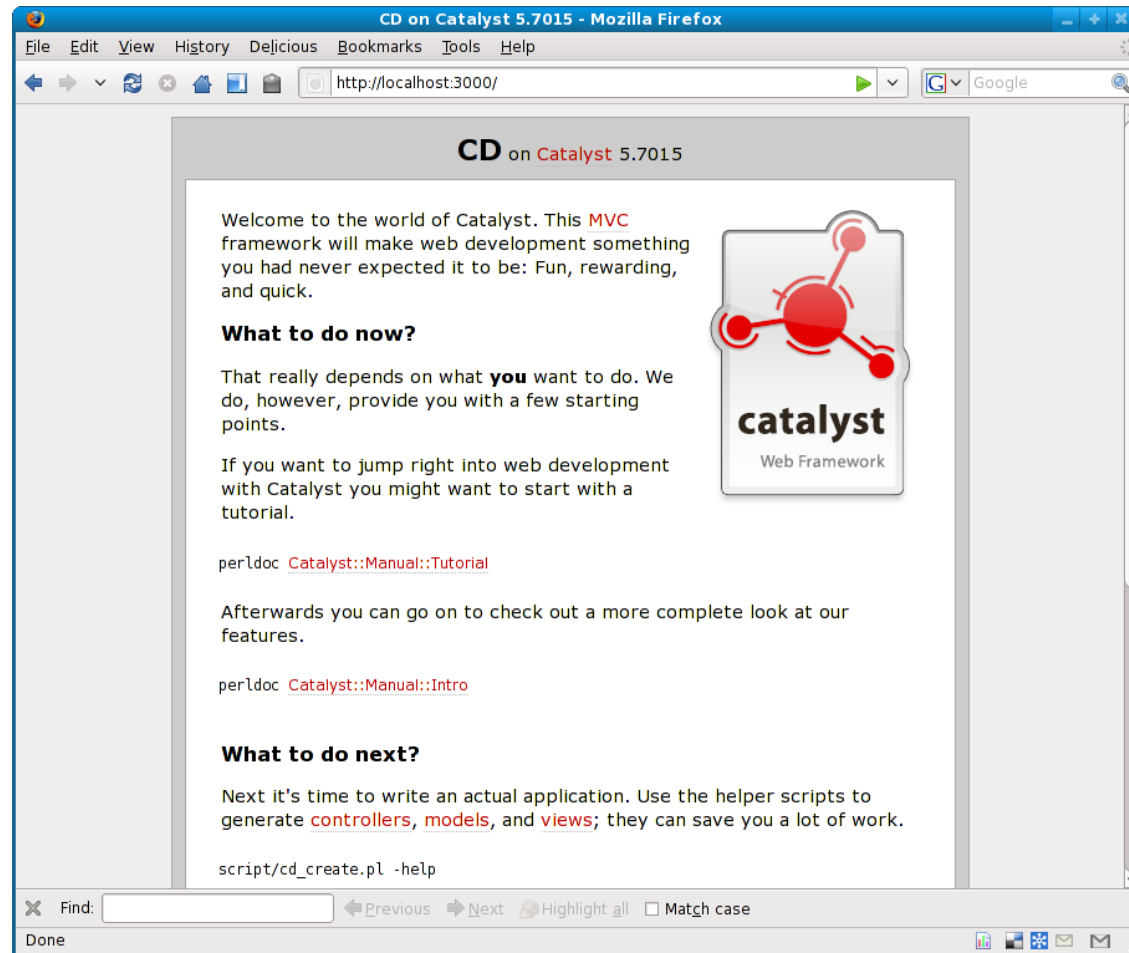
- We already have a working application
- `$ CD/script/cd_server.pl`

... lots of output

```
[info] CD powered by Catalyst 5.7015
You can connect to your server at
http://localhost:3000
```

- Of course, it doesn't do much yet

# Simple Catalyst App



# Next Steps

- Use various helper programs to create models and views for your application
- Write controller code to tie it all together
- Many plugins to handle various parts of the process
  - Authentication
  - URL resolution
  - Session handling
  - etc...



# Create a View

- ```
$ script/cd_create.pl view Default TT  
exists  
"/home/dave/training/cdlib/CD/script/../lib/CD/V  
iew"  
exists  
"/home/dave/training/cdlib/CD/script/../t"  
created  
"/home/dave/training/cdlib/CD/script/../lib/CD/V  
iew/Default.pm"  
created  
"/home/dave/training/cdlib/CD/script/../t/view_D  
efault.t"
```

Remove Default Message

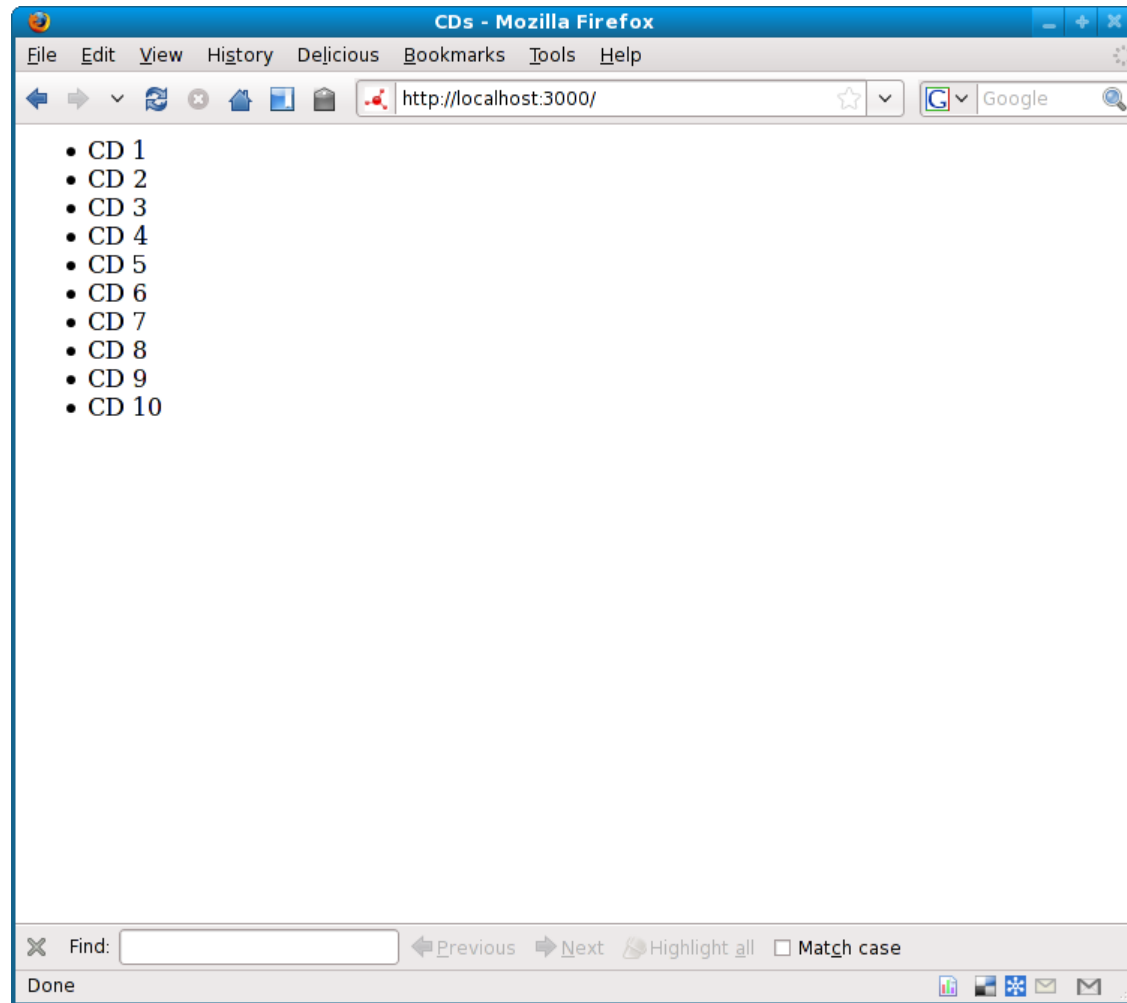
- In lib/CD/Controller/Root.pm
- ```
sub index :Path :Args(0) {
 my ($self, $c) = @_;

 # Hello World
 $c->response_body($c->welcome_message);
}
```
- Remove response\_body line
- Default behaviour is to render index.tt
- Need to create that

# index.tt

- root/index.tt
- `<html>`
  - `<head>`
    - `<title>CDs</title>`
  - `</head>`
  - `<body>`
    - `<ul>`
      - `[% FOREACH cd IN [ 1 .. 10 ] %]`
        - `<li>CD [% cd %]</li>`
    - `[% END %]`
  - `</ul>`
  - `</body>`
- `</html>`

# New Front Page



# Adding Data

- Of course that's hard-coded data
- Need to add a model class
- And then more views
- And some controllers
- There's a lot to do
- I recommend working through a tutorial

# Easier Catalyst

- A lot of web applications do similar things
- Given a database
- Produce screens to edit the data
- Surely most of this can be automated
- It's called Catalyst::Plugin::AutoCRUD
- (Demo)

# Cat::Plugin::AutoCRUD

- Does a lot of work
- On the fly
- For every request
- No security on table updates
- So it's not right for every project
- Very impressive though

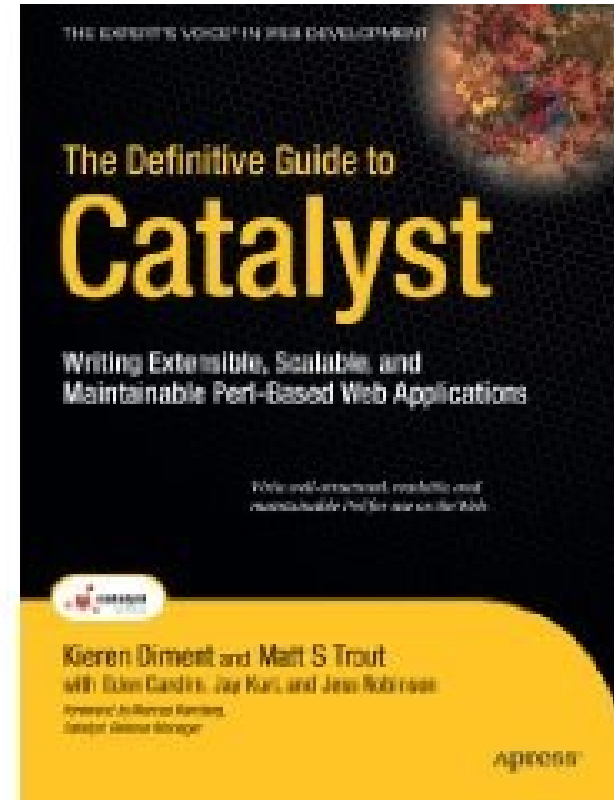
# Conclusions

- There's a lot to bear in mind when writing a web app
- Using the right framework can help
- Catalyst is the most popular Perl framework
- As powerful as any other framework
  - In any language
- Lots of work still going on
- Large team, active development



# Recommended Book

- The Definitive Guide to Catalyst
  - Kieren Diment
  - Matt S Trout



# Catalyst Examples

- Set up a basic Catalyst project based on the CD database from yesterday
- We will then work together to add some simple controllers and views to the application
- Add `Catalyst::Plugin::AutoCRUD` to your application

# 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 (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

# 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' ],  
  ];  
};
```

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/`

Plack::Request

- Plack::Request turns the environment into a request object
- use Plack::Request;
use Data::Dumper;

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

Plack::Response

- Plack::Response builds a PSGI response object

- ```
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;
}
```

# Middleware

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

# 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: Tue, 20 Jul 2010 20:25:52 GMT
Server: HTTP::Server::PSGI
Content-Length: 11
Content-Type: text/plain
Client-Date: Tue, 20 Jul 2010 20:25:52 GMT
Client-Peer: 127.0.0.1:5000
Client-Response-Num: 1
X-Runtime: 0.000050

Middleware Example

- \$ HEAD http://localhost:5000
200 OK
Date: Tue, 20 Jul 2010 20:25:52 GMT
Server: HTTP::Server::PSGI
Content-Length: 11
Content-Type: text/plain
Client-Date: Tue, 20 Jul 2010 20:25:52 GMT
Client-Peer: 127.0.0.1:5000
Client-Response-Num: 1
X-Runtime: 0.000050

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;
```

Framework Support

- Many modern Perl applications already support PSGI
- Catalyst, CGI::Application, Dancer, Jifty, Mason, Maypole, Mojolicious, Squatting, Web::Simple
- Many more

Catalyst Support

- Catalyst::Engine::PSGI
- use MyApp;

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

- Also Catalyst::Helper::PSGI
- script/myapp_create.pl PSGI

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

Plack::Handler::Apache2

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

# PSGI/Plack Summary

- PSGI is a specification
- Plack is an implementation
- PSGI makes your life easier
- Most of the frameworks and server you use already support PSGI
- No excuse not to use it

# Further Information

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

# PSGI/Plack Examples

- Try out some of the simple PSGI examples from the previous slides
- Use plackup to run the applications

# That's all folks

- Any questions?

