

Introduction to mod_perl 2.0

February 11th, 2007

Frank Wiles
Revolution Systems

frank@revsys.com

So what IS mod_perl?

mod_perl is...

- An Apache module much like mod_cgi, mod_deflate, mod_speling, etc.

mod_perl is...

- An Apache module much like mod_cgi, mod_deflate, mod_speling, etc.
- An embedded Perl interpreter

mod_perl is...

- An Apache module much like mod_cgi, mod_deflate, mod_speling, etc.
- An embedded Perl interpreter
- A Perl interface to the Apache API

mod_perl is...

- An Apache module much like mod_cgi, mod_deflate, mod_speling, etc.
- An embedded Perl interpreter
- A Perl interface to the Apache API
- A way to *dramatically* increase the performance of existing/legacy CGIs

mod_perl is...

- An Apache module much like mod_cgi, mod_deflate, mod_speling, etc.
- An embedded Perl interpreter
- A Perl interface to the Apache API
- A way to *dramatically* increase the performance of existing/legacy CGIs
- IMHO the coolest thing on earth...

mod_perl is...

- An Apache module much like mod_cgi, mod_deflate, mod_speling, etc.
- An embedded Perl interpreter
- A Perl interface to the Apache API
- A way to *dramatically* increase the performance of existing/legacy CGI's
- IMHO the coolest thing on earth... (ok that might be reaching a bit)

You're probably asking yourself, "So what if Frank thinks it is cool, what can it do for me?"

Some examples of what mod_perl can do...

- Allow you to mass configure thousands of VirtualHosts from a database

Some examples of what mod_perl can do...

- Allow you to mass configure thousands of VirtualHosts from a database
- Fix nasty HTML emitted by legacy web applications

Some examples of what mod_perl can do...

- Allow you to mass configure thousands of VirtualHosts from a database
- Fix nasty HTML emitted by legacy web applications (basically filter the I/O of any other module)

Some examples of what mod_perl can do...

- Allow you to mass configure thousands of VirtualHosts from a database
- Fix nasty HTML emitted by legacy web applications (basically filter the I/O of any other module)
- Allow you to write a SMTP

Some examples of what mod_perl can do...

- Allow you to mass configure thousands of VirtualHosts from a database
- Fix nasty HTML emitted by legacy web applications (basically filter the I/O of any other module)
- Allow you to write a SMTP, FTP

Some examples of what mod_perl can do...

- Allow you to mass configure thousands of VirtualHosts from a database
- Fix nasty HTML emitted by legacy web applications (basically filter the I/O of any other module)
- Allow you to write a SMTP, FTP, POP3

Some examples of what mod_perl can do...

- Allow you to mass configure thousands of VirtualHosts from a database
- Fix nasty HTML emitted by legacy web applications (basically filter the I/O of any other module)
- Allow you to write a SMTP, FTP, POP3, IMAP

Some examples of what mod_perl can do...

- Allow you to mass configure thousands of VirtualHosts from a database
- Fix nasty HTML emitted by legacy web applications (basically filter the I/O of any other module)
- Allow you to write a SMTP, FTP, POP3, IMAP, Jabber

Some examples of what mod_perl can do...

- Allow you to mass configure thousands of VirtualHosts from a database
- Fix nasty HTML emitted by legacy web applications (basically filter the I/O of any other module)
- Allow you to write a SMTP, FTP, POP3, IMAP, Jabber, or whatever other protocol you desire

Some examples of what mod_perl can do...

- Allow you to mass configure thousands of VirtualHosts from a database
- Fix nasty HTML emitted by legacy web applications (basically filter the I/O of any other module)
- Allow you to write a SMTP, FTP, POP3, IMAP, Jabber, or whatever other protocol you desire while giving you the stable base of Apache to build upon

Or if you only care about
HTTP....

**mod_perl allows you to
change the fundamental
ways Apache works for
you today!**

Why would I want to change Apache?

- Tired of seeing “favicon.ico not found” show up in your logs?

Why would I want to change Apache?

- Tired of seeing “favicon.ico not found” show up in your logs? **Change Apache's default logging!**

Why would I want to change Apache?

- Tired of seeing “favicon.ico not found” show up in your logs? **Change Apache's default logging!**
- Annoyed that you have to drop everything because marketing wants a Redirect or RewriteRule added **RIGHT NOW**

Why would I want to change Apache?

- Tired of seeing “favicon.ico not found” show up in your logs? **Change Apache's default logging!**
- Annoyed that you have to drop everything because marketing wants a Redirect or RewriteRule added **RIGHT NOW. So build them an interface to do it themselves!**

Why would I want to change Apache?

- Tired of seeing “favicon.ico not found” show up in your logs? **Change Apache's default logging!**
- Annoyed that you have to drop everything because marketing wants a Redirect or RewriteRule added **RIGHT NOW. So build them an interface to do it themselves!**
- Want to get an E-mail on every 404 or 500 error?

Installing mod_perl

- You'll find most Linux distributions already have working mod_perl modules, so it is a simple matter of turning it on.
- If not, it's a simple matter of:
 - `tar -xvzf modperl-current.tar.gz`
 - `perl Makefile.PL MP_APXS=/path/to/your/apxs`
 - `make; make test; make install`

Installing mod_perl

- One notable exception, if you happen to be running Red Hat Enterprise Linux versions 3 or 4 there is a bit of an issue...

Speeding up your CGIs

- 99% of your existing CGIs can get a 2x-50x speed boost simply by running them through mod_perl

Speeding up your CGIs

- 99% of your existing CGIs can get a 2x-50x speed boost simply by running them through `mod_perl`
- This is accomplished by using the `ModPerl::Registry` or `ModPerl::RegistryPreFork` if using the Prefork MPM

Speeding up your CGIs

- 99% of your existing CGIs can get a 2x-50x speed boost simply by running them through `mod_perl`
- This is accomplished by using the module `ModPerl::Registry` or `ModPerl::RegistryPreFork` if using the Prefork MPM
- **SOME** CGIs misbehave and have to be run inside of `ModPerl::PerlRun` instead

How it works...

- Your typical Perl CGI goes through these phases:
 - fork a Perl interpreter
 - compile the Perl source
 - execute the actual Perl code

How it works...

- Your typical Perl CGI goes through these phases:
 - fork a Perl interpreter
 - compile the Perl source
 - execute the actual Perl code
- ModPerl::Registry reduces this to:
 - compile the Perl source, unless already cached
 - execute the Perl code

httpd.conf configuration

```
LoadModule perl_module modules/mod_perl.so
```

```
PerlModule ModPerl::Registry
```

```
Alias /perl/ /home/httpd/perl
```

```
<Location /perl>
```

```
    SetHandler perl-script
```

```
    PerlResponseHandler ModPerl::Registry
```

```
    Options +ExecCGI
```

```
</Location>
```

Performance Example

```
use Date::Manip;
use XML::Simple;

print "Content-type: application/xml\n\n";

my %output;

# Build our weird dates
my $today          = ParseDate("today");
my $next_friday    = ParseDate("next Friday");
my $three_biz_days = DateCalc( "today", "+3 business days +2 hours");
my $nine_biz_days  = DateCalc( "today", "+9 business days -1 hour");
my $tweleve_biz_days = DateCalc( "today", "-12 business days +3 hours");

$output{dates} = {};

my @array = (
    UnixDate($today, "%d-%m-%Y"),
    UnixDate($today, "%d-%m-%Y"),
    UnixDate($next_friday, "%d-%m-%Y"),
    UnixDate($three_biz_days, "%d-%m-%Y") );

$output{dates}{today}      = \@array;
$output{dates}{others}    = UnixDate($next_friday, "%d-%m-%Y");
$output{dates}{more}      = UnixDate( $nine_biz_days, "%d-%m-%Y");
$output{dates}{more2}     = UnixDate( $tweleve_biz_days, "%d-%m-%y");

# Build our XML
my $xs = XML::Simple->new;
my $xml = $xs->XMLout( \%output );
print $xml;
```

Performance Comparison

- As a CGI
 - 15.19 Requests Per Second
 - Mean time per request 329.065 ms

Performance Comparison

- As a CGI
 - 15.19 Requests Per Second
 - Mean time per request 329.065 ms
- Running under ModPerl::Registry
 - 151.23 Requests Per Second
 - Mean time per request 32.062 ms

Understanding the Apache Lifecycle

- Server life cycle handlers
 - PerlOpenLogsHandler
 - PerlPostConfigHandler
 - PerlChildInitHandler
 - PerlChildExitHandler

Understanding the Apache Lifecycle

- Protocol Handlers
 - PerlPreConnectionHandler
 - PerlProcessConnectionHandler

Understanding the Apache Lifecycle

- Filter related handlers
 - PerlInputFilterHandler
 - PerlOutputFilterHandler

Understanding the Apache Lifecycle

- PerlPostReadRequestHandler
- PerlTransHandler
- PerlMapToStorageHandler
- PerlHeaderParserHandler
- PerlAccessHandler
- PerlAuthenHandler
- PerlAuthzHandler
- PerlTypeHandler
- PerlFixupHandler
- PerlResponseHandler
- PerlLogHandler
- PerlCleanupHandler

Hello World!

```
package My::HelloWorld;

use strict;
use Apache2::RequestRec;
use Apache2::RequestIO;
use Apache2::Const -compile => qw( :common );

sub handler {
    my $r = shift;

    $r->content_type( 'text/html' );

    print qq{
        <html><body><h1>Hello World!</h1></body></html>
    };

    return( Apache2::Const::Ok );
}
1;
```

Configuring My::HelloWorld

```
<Perl>
    use Apache2::RequestRec;
    use Apache2::RequestUtil;
    use Apache2::RequestIO;
    use Apache2::Const -compile => qw( :common );

    use My::HelloWorld;
</Perl>

<Location /hello>
    SetHandler modperl
    PerlResponseHandler My::HelloWorld
</Location>
```

Configuring My::HelloWorld

```
<Perl>
    use Apache2::RequestRec;
    use Apache2::RequestUtil;
    use Apache2::RequestIO;
    use Apache2::Const -compile => qw( :common );

    use My::HelloWorld;
</Perl>

<Location /hello>
    SetHandler modperl
    PerlResponseHandler My::HelloWorld
</Location>
```

Configuring My::HelloWorld

```
<Perl>
    use Apache2::RequestRec;
    use Apache2::RequestUtil;
    use Apache2::RequestIO;
    use Apache2::Const -compile => qw( :common );

    use My::HelloWorld;
</Perl>
```

```
<Location /hello>
    SetHandler modperl
    PerlResponseHandler My::HelloWorld
</Location>
```

SetHandler perl-script is equivalent to:

```
SetHandler modperl
PerlOptions +SetupEnv +GlobalRequest
```

My::SillyAuth

```
package My::SillyAuth;

use strict;
use Apache2::Access;
use Apache2::RequestUtil;

use Apache2::Const -compile => qw( :common :http );

sub handler {
    my $r = shift;

    my ($status, $password) = $r->get_basic_auth_pw;

    # Allow the user in, if they follow our rules
    if( $r->user =~ /\d/ and $password =~ /\$/ ) {
        return Apache2::Const::Ok
    }
    else {
        $r->note_basic_auth_failure;
        return Apache2::Const::HTTP_UNAUTHORIZED;
    }
}

1;
```

Configuring My::SillyAuth

```
<Perl>
    use Apache2::Access;
    use Apache2::RequestUtil;

    use My::SillyAuth;
</Perl>

<Location /protected-area>
    SetHandler perl-script
    PerlAuthenHandler My::SillyAuth

    AuthType Basic
    AuthName "Silly Authentication Scheme"
    Require valid-user
</Location>
```

My::SpecialLogging

```
package My::SpecialLogging;

use Apache2::RequestRec;
use Apache2::Connection;
use Apache2::Const -compile => qw(OK DECLINED);
use Fcntl qw( :flock );

sub handler {
    my $r          = shift;
    my $log_path   = $r->dir_config('LogFile');

    return Apache2::Const::DECLINED
        if $r->uri =~ m|^/(admin|css|robots\.txt|favicon.ico)|

    open my $fh, ">>$log_path" or die "Cannot open log: $!";
    flock $fh, LOCK_EX;
    print $fh sprintf("%s [%s] %s",    scalar(localtime),
                                                                    $r->connection->remote_ip,
                                                                    $r->uri );

    close $fh;
    return Apache2::Const::OK;
}
1;
```

Configuring My::SpecialLogging

```
<Perl>
    use Apache2::RequestRec;
    use Apache2::Connection;
    use Fcntl qw( :flock );
    use Apache2::Const -compile => qw(OK DECLINED);

    use My::SpecialLogging;
</Perl>

<Location />
    SetHandler perl-script
    PerlSetVar LogFile /var/log/special.log
    PerlLogHandler My::SpecialLogging
</Location>
```

Configuring Apache with Perl

```
<Perl>
    use DBI;
    use MyHostingCompany::VHosts;

    $ServerAdmin = 'webmaster@mycompany.com';

    my $vhosts = MyHostingCompany::VHosts->new;

    foreach my $host ( $vhosts->all ) {

        $VirtualHost{$host} = {
            ServerName      => $host,
            ServerAlias     => "www.$host",
            DocumentRoot    => $host->doc_root,
        }

    }

</Perl>
```

I/O Filtering Modules

```
package My::LastModifiedFilter;

use base 'Apache2::Filter';      # Inherit from Apache2::Filter
use APR::Finfo;                  # For our file info
use APR::Table;                  # For $f->$r->headers_out->unset
use Apache2::Const -compile => qw(OK);

sub handler {
    my $f      = shift;          # Our filter object
    my $finfo  = $f->r->finfo;    # Our file information

    unless( $f->ctx ) {
        $f->r->headers_out->unset('Content-Length');
        $f->ctx(1);
    }

    while( $f->read(my $buffer, 1024) ) {
        if( $buffer =~ m|</BODY>|i ) {
            $f->print("Last modified: " .
                    scalar( localtime( $finfo->mtime ) ) .
                    "</BODY>" );
        }
        else { $f->print( $buffer ); }
    }
    return( Apache2::Const::OK );
}
1;
```

Using Custom Apache Directives

```
package My::DBIParameters;

use Apache2::CmdParms;
use Apache2::Module;
use Apache2::ServerUtil;

my @directives = ( {
    name      => 'DSN',
    func      => __PACKAGE__ . '::Single',
    errmsg    => "DSN 'dbi:Pg:dbuser=nobody'",
  } );

sub Single {
    my ($self, $param, $arg) = @_;
    $self->{$param->info} = $arg;
}

Apache2::Module::add( __PACKAGE__, \@directives );
1;
```

Using Custom Apache Directives

```
PerlLoadModule My::DBIParameters;
```

```
<Location /myapp>  
    SetHandler perl-script  
    DSN "dbi:Pg:dbname=myapp"  
    PerlResponseHandler MyApp::Something  
</Location>
```

Using Custom Apache Directives

```
PerlLoadModule My::DBIParameters;
```

```
<Location /myapp>  
    SetHandler perl-script  
    DSN "dbi:Pg:dbname=myapp"  
    PerlResponseHandler MyApp::Something  
</Location>
```

Then somewhere in `MyApp::Something...`

```
my %dbi_config = Apache2::Module::get_config(  
    'My::DBIParameters'  
    $r->server,  
    $r->per_dir_config  
);
```

Using Custom Apache Directives

```
PerlLoadModule My::DBIParameters;
```

```
<Location /myapp>  
    SetHandler perl-script  
    DSN "dbi:Pg:dbname=myapp"  
    PerlResponseHandler MyApp::Something  
</Location>
```

Then somewhere in `MyApp::Something...`

```
my %dbi_config = Apache2::Module::get_config(  
    'My::DBIParameters'  
    $r->server,  
    $r->per_dir_config  
);
```

`$dbi_config{DSN}` is now `'dbi:Pg:dbname=myapp'`

Making Directives Easier

```
package My::EasierDBI;
use base 'ModPerl::ParamBuilder'      # From CPAN
my $builder = ModPerl::ParamBuilder->new( __PACKAGE__ )
;

$builder->param( 'DSN' );
$builder->param( 'DBUser' );
$builder->param( 'DBPass' );

$builder->on_off( 'RaiseError' );

$builder->yes_no( 'AutoCommit' );

$builder->load;
1;
```

Making Directives Easier

Then in MyApp::Something we do:

```
use My::EasierDBI;  
my $params          = My::EasierDBI->new;  
my $config_ref     = $params->get_config;
```

Resources

- mod_perl site <http://perl.apache.org>
- Slides <http://www.revsys.com/talks>

Resources

- mod_perl site <http://perl.apache.org>
- Slides <http://www.revsys.com/talks>

Thanks!

Q&A