mod_perl Best Practices

Philippe M. Chiasson
gozer@ectoplasm.org

Overview

- mod_perl is many things
  - Perl glue to APR
  - Perl glue to Apache/httpd
  - More Perl'ish APIs where needed
  - Pure Perl convenience modules
  - NOT JUST CGI ACCELERATION!

Best Practices

- There is no single answer
- Recommended practices
- From the experiences of mod_perl users
- Take your pick
- Use them when it makes sense
- Take some, leave some

mod_perl 1.x

- The original project, by Doug M.
- Mainly hand-written
- Glue code added on demand
- 400 tests
- a few hundred methods

mod_perl 2.x

- A complete rewrite
- Mostly auto-generated code
- Glue code complete from the start
- 2500+ tests
- A much larger API exposed

A demo

```
ScriptAlias /cgi @DocumentRoot@
```

Hello World!

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

use CGI qw(:standard);

my $q = new CGI;

print header;
print start_html("Simple Demo");
print h1("Hello World!");
print end_html;
```

Measure everything

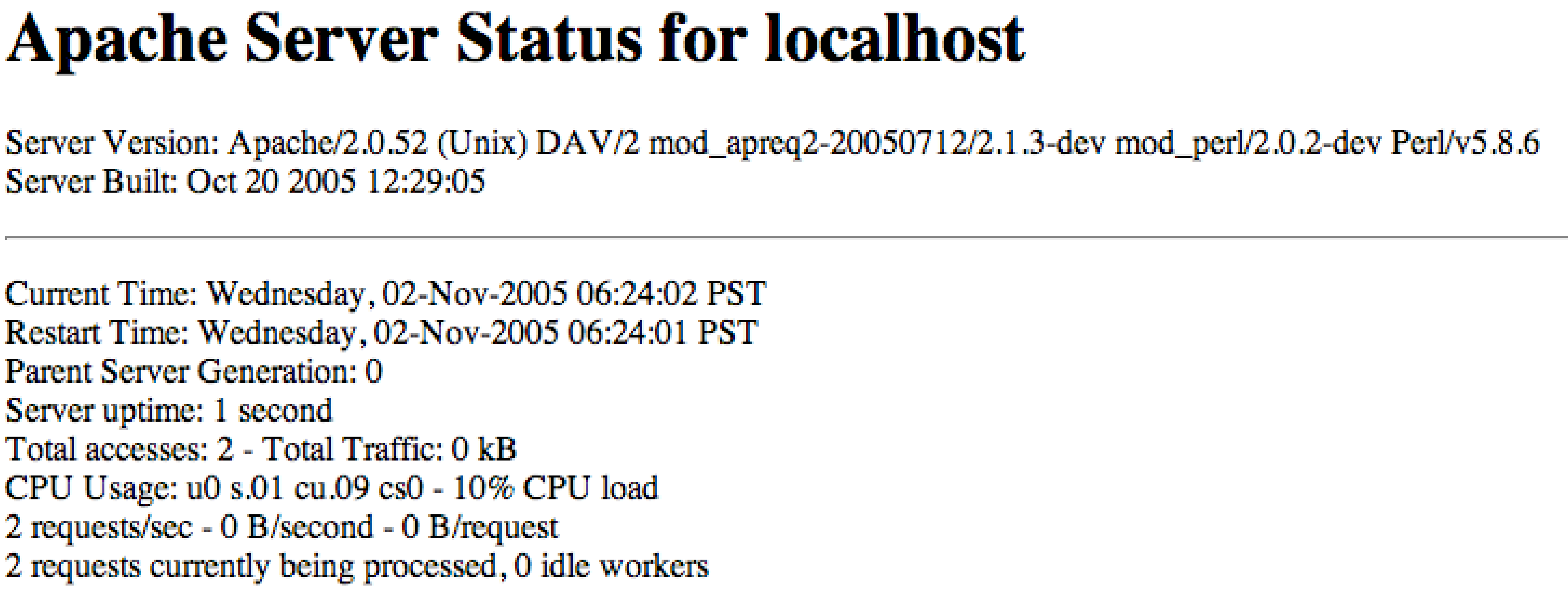
- ab - Apache Bench
- mod_status
- Apache::Status

ab - Apache Bench

```
+ ab -c1 -n50 http://127.0.0.1:8529/cgi/demo.pl
Requests per second:    4.91 [#/sec] (mean)
Time per request:       203.684 [ms] (mean)
Time per request:       203.684 [ms] (mean, across all concurrent requ
Transfer rate:          2.75 [Kbytes/sec] received
```

mod_status

```
ExtendedStatus On
<Location /server/status>
  SetHandler server-status
</Location>
```



Apache Server Status for localhost

Server Version: Apache/2.0.52 (Unix) DAV/2 mod_apreq2-20050712/2.1.3-dev mod_perl/2.0.2-dev Perl/v5.8.6
Server Built: Oct 20 2005 12:29:05

Current Time: Wednesday, 02-Nov-2005 06:24:02 PST
Restart Time: Wednesday, 02-Nov-2005 06:24:01 PST
Parent Server Generation: 0
Server uptime: 1 second
Total accesses: 2 - Total Traffic: 0 kB
CPU Usage: u0 s.01 cu.09 cs0 - 10% CPU load
2 requests/sec - 0 B/second - 0 B/request
2 requests currently being processed, 0 idle workers

Apache2::Status

- Ships with mod_perl
- Invaluable tools to inspect a running server
- Enable it

Apache2::Status

```
PerlModule Apache2::Status
<Location /perl-status>
  SetHandler perl-script
  PerlHandler Apache2::Status
  PerlSetVar StatusOptionsAll On
</Location>
```

ModPerl::PerlRun

```
Alias /perlrun @DocumentRoot@

<Location /perlrun>
  SetHandler perl-script
  PerlHandler ModPerl::PerlRun
  Options +ExecCGI
</Location>
```

ModPerl::Registry

```
Alias /registry @DocumentRoot@

<Location /registry>
  SetHandler perl-script
  PerlHandler ModPerl::Registry
  Options +ExecCGI
</Location>
```

Avoid ModPerl::PerlRun

- Significant overhead
- Closest to CGI
- Only until something else can work
- Last resort

Prefer ModPerl::Registry

- Little overhead
- Lots of caching
- Best, quickest approach

Preload modules

- Preloading means more shared memory
- Avoid lazy loading
- Preload everything early, explicitely

Use Handlers

- ModPerl::(Registry|PerlRun) == easy
  - Ports CGI-like scripts in no time
  - Carries overhead and side-effects
- When writing from scratch, go handlers

Demo Handler

```
use Apache2::RequestIO ();
use Apache2::Request();
use Apache2::Const -compile => qw(OK);

use CGI qw(:standard);

sub handler {
    my $q = new CGI;
    print header;
    print start_html("Simple Demo");
    print h1("Hello World!");
    print p(“mod_perl: $ENV{MOD_PERL}”);
    print end_html;

    Apache2::Const::OK;
}
```

ModPerl::MethodLookup

- mod_perl 2.0 has LOTS of methods
- mod_perl 2.0 has LOTS of classes
- Allows finer grained control

ModPerl::MethodLookup

- Comes with mod_perl
- Knows of every single class, method, structs
- Designed to provide easy, handy shell aliases

ModPerl::MethodLookup

```
$> alias mp2mod perl -MModPerl::MethodLookup -e print_module
$> alias mp2met perl -MModPerl::MethodLookup -e print_method
$> alias mp2obj perl -MModPerl::MethodLookup -e print_object
```

A better Demo

Precompute when possible

- mod_perl is a persistent environment
- Makes a great candidate for heavy caching
- Precompute whenever the benefit is worth it
- Benchmarks of course

Globals

- Don’t use Globals!
  - if you can change them, ever
- Keep globals read-only if you need them
- Use them sparsingly

Hash::Utils::lock_hash

- If using globals, Hash::Utils (part of Perl) can help
- Hash::Utils::lock_hash and unlock_hash
- Use it and it will save your skin at least once

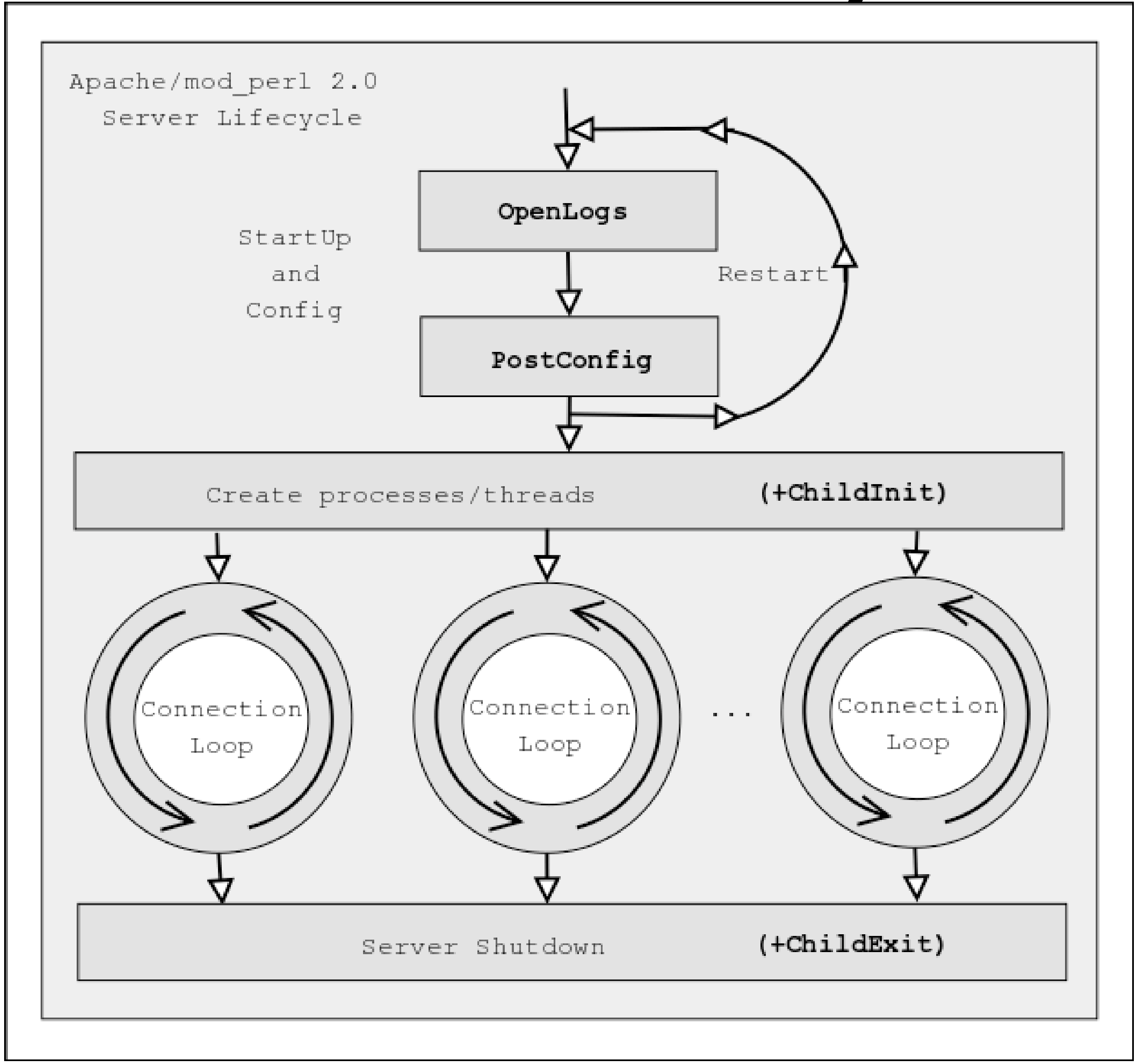
Use request phases

- There are tons of handler phases available
- Find and use the right ones for the job
- Spreading logic across multiple phases makes for more modular logic
- Apache/httpd's model is old and tested, use it

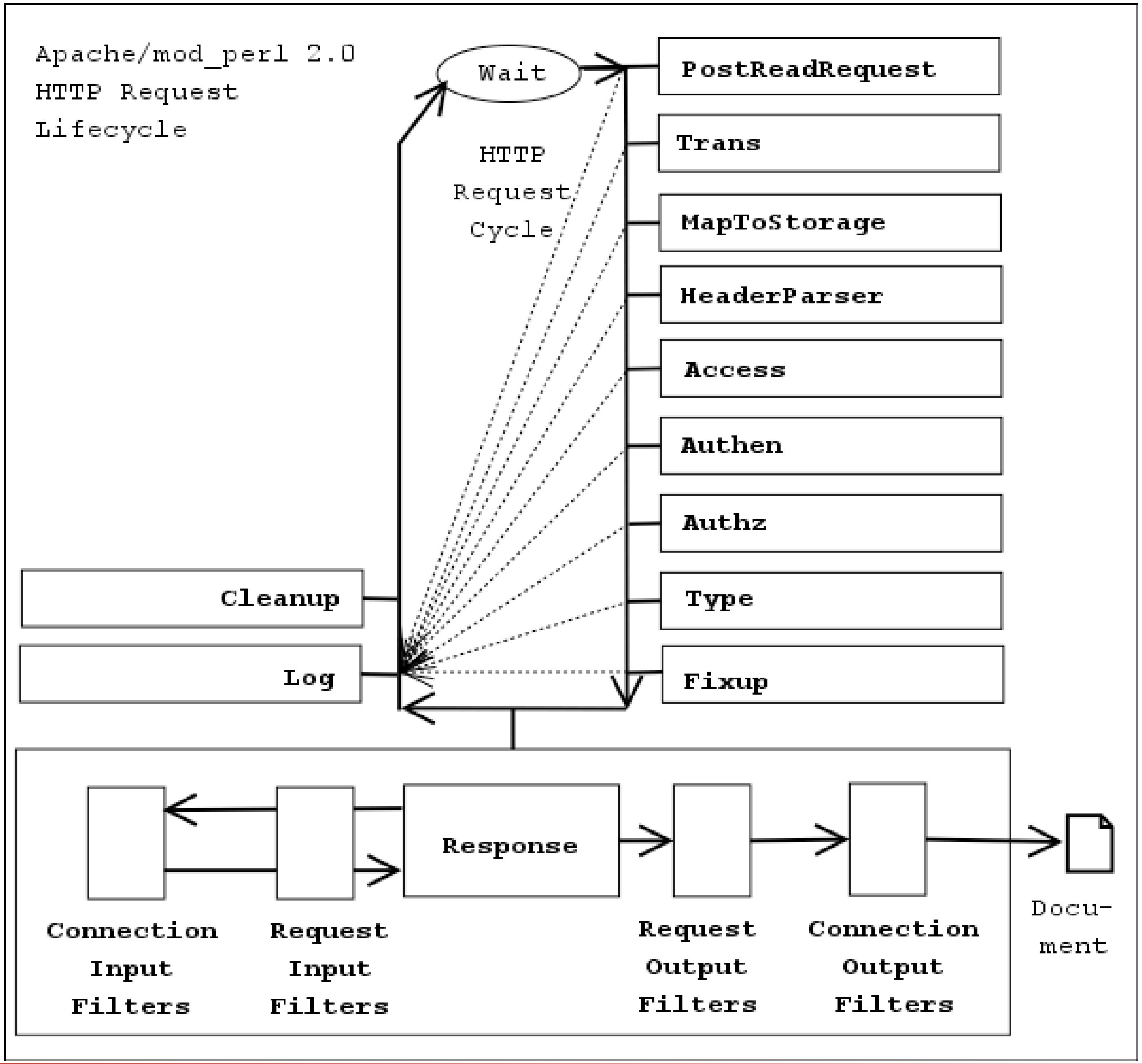
Use request phases

- Server life cycle
  - `PerlOpenLogsHandler`
  - `PerlPostConfigHandler`
  - `PerlChildInitHandler`
  - `PerlChildExitHandler`
- Protocols
  - `PerlPreConnectionHandler`
  - `PerlProcessConnectionHandler`
- Filters
  - `PerlInputFilterHandler`
  - `PerlOutputFilterHandler`
- HTTP Protocol
  - `PerlPostReadRequestHandler`
  - `PerlTransHandler`
  - `PerlMapToStorageHandler`
  - `PerlInitHandler`
  - `PerlHeaderParserHandler`
  - `PerlAccessHandler`
  - `PerlAuthenHandler`
  - `PerlAuthzHandler`
  - `PerlTypeHandler`
  - `PerlFixupHandler`
  - `PerlResponseHandler`
  - `PerlLogHandler`
  - `PerlCleanupHandler`

Server Life-Cycle



HTTP Handlers



Logging

- You have access to Apache/httpd's error_log
- Should be used for errors/warnings/info
- Otherwise, use APR::PerlIO

Logging

- $r->log->info(“msg”);
- $r->log->error(“msg”);

Logging

- Don’t clutter the error_log
- Create your own log files
- APR::PerlIO

APR::PerlIO

- Apache/httpd can be multi-threaded
- thread-safety is important
- APR::PerlIO hides it all
- great for log files

APR::PerlIO

```
use APR::PerlIO ();
open my $logfh, ">:APR", $logfile, $pool or die $!;
```

$(r|c)->p?notes

- Sometimes, it's needed to pass data between different phases
- Don't use globals or external entities
- You have access to notes

Do the right thing

- Keep your module code working outside mod_perl
- When it makes sense
- Be smart and use $ENV{MOD_PERL} to do the right thing

Hook into Apache2::Status

```
use Apache2::Module;
Apache2::Status->menu_item(
    'WebGUI' => 'Status for WebGUI Demo quote DB',
    \&status,
) if Apache2::Module::loaded('Apache2::Status');

sub status {
    my ($r, $q) = @_;
    my @status;
    my $stats = __PACKAGE__->author_cache_stats();
    push @status, "$stats entries in the author cache";
    return \@status;
}
```

use libapreq2

- Apache HTTP Request Library
- Specialized library
  - Parsing POST body
  - Parsing query strings
  - Uploaded files
  - Cookies

use libapreq2

```
$> wget http://httpd.apache.org/dist/httpd/libapreq/libapreq2-2.06
dev.tar.gz
$> tar zxvf
$> cd libapreq-*/
$> perl Makefile.PL --with-apache2-apxs=`which apxs`
$> make
$> make test
$> make install
```

Apache2::Request

```
use Apache2::Request;
$req = Apache2::Request->new($r);
@foo = $req->param("foo");
$bar = $req->args("bar");
```

Apache2::Cookie

```
use Apache2::Cookie;

$jar = Apache2::Cookie::Jar->new($r);

$incoming_cookie = $jar->cookies("WebGUI-Quotes");
$outgoing_cookie = Apache2::Cookie->new($r,
                      -name   => "WebGUI-Quotes",
                       -value => $quote_id );
$outgoing_cookie->bake;
```

Apache Config Directives

- Apache configuration's framework
  - Hiearchical
  - HTTP Centric
  - Comfortable
- Use it where it makes sense

Perl(Set|Add)Var

```
PerlSetVar Quotes_DSN “dbi:mysql:quotes”

<Location /quotes>
  SetHandler perl-script
  PerlHandler WebGUI::Quotes
</Location>

<Location /quotes-dev>
  SetHandler perl-script
  PerlHandler WebGUI::Quotes
  PerlSetVar Quotes_DSN “dbi:mysql:quotes-dev”
</Location>


sub handler {
  my $r = shift;
  my $dsn = $r->dir_config(‘Quotes_DSN’);
  DBI->connect($dsn);
```

Dynamic Configuration

- Handler configuration grows complex
- Often, configuration is tied to code
- Hide it, handle it dynamically

$s->add_config()

```
use Apache2::ServerUtil ();
my $conf = <<'EOC';
<Location /quote>
    SetHandler perl-script
    PerlResponseHandler WebGUI::Quote
  </Location>
  EOC
  Apache2::ServerUtil->server->add_config([split /\n/, $conf]);
```

Apache-Test

- It's own Apache projects
- Spawned from mod_perl
- Web testing framework
- Pure & simple magic
- Use it early, use it often

Thank you!

More info

- *mod_perl User’s mailing-list*
  - http://perl.apache.org/maillist/modperl.html
  - <modperl@perl.apache.org>
- *mod_perl Developer's Cookbook*
  - http://www.modperlcookbook.org/
- Practical mod_perl
  - http://www.modperlbook.org/
- mod_perl at the ASF
  - http://perl.apache.org/

Thank You!

Slides and bonus material:

http://gozer.ectoplasm.org/talk/