SRI / Mojolicious-9.39 / lib / Mojolicious.pm

```
package Mojolicious;
use Mojo::Base -base;

# "Fry: Shut up and take my money!"
use Carp ();
use Mojo::DynamicMethods -dispatch;
use Mojo::Exception;
use Mojo::Home;
use Mojo::Loader;
use Mojo::Log;
use Mojo::Server;
use Mojo::Util;
use Mojo::UserAgent;
use Mojolicious::Commands;
use Mojolicious::Controller;
use Mojolicious::Plugins;
use Mojolicious::Renderer;
use Mojolicious::Routes;
use Mojolicious::Sessions;
use Mojolicious::Static;
use Mojolicious::Types;
use Mojolicious::Validator;
use Scalar::Util ();

has commands         => sub { Mojolicious::Commands->new(app => shift) };
has controller_class => 'Mojolicious::Controller';
has exception_format => 'html';
has home             => sub { Mojo::Home->new->detect(ref shift) };
has log              => sub {
  my $self = shift;

  # Reduced log output outside of development mode
  my $log = Mojo::Log->new;
  return $log->level($ENV{MOJO_LOG_LEVEL}) if $ENV{MOJO_LOG_LEVEL};
  return $self->mode eq 'development' ? $log : $log->level('info');
};
has 'max_request_size';
has mode               => sub { $ENV{MOJO_MODE} || $ENV{PLACK_ENV} || 'development'
has moniker            => sub { Mojo::Util::decamelize ref shift };
has plugins            => sub { Mojolicious::Plugins->new };
has preload_namespaces => sub { [] };
has renderer           => sub { Mojolicious::Renderer->new };
has routes             => sub { Mojolicious::Routes->new };
has secrets            => sub {
  my $self = shift;

  # Warn developers about insecure default
  $self->log->trace('Your secret passphrase needs to be changed (see FAQ for more)'

  # Default to moniker
  return [$self->moniker];
};
has sessions  => sub { Mojolicious::Sessions->new };
has static    => sub { Mojolicious::Static->new };
has types     => sub { Mojolicious::Types->new };
has ua        => sub { Mojo::UserAgent->new };
has validator => sub { Mojolicious::Validator->new };

our $CODENAME = 'Waffle';
our $VERSION  = '9.39';

```

```
sub BUILD_DYNAMIC {
  my ($class, $method, $dyn_methods) = @_;

  return sub {
    my $self    = shift;
    my $dynamic = $dyn_methods->{$self->renderer}{$method};
    return $self->build_controller->$dynamic(@_) if $dynamic;
    my $package = ref $self;
    Carp::croak qq{Can't locate object method "$method" via package "$package"};
  };
}

sub build_controller {
  my ($self, $tx) = @_;

  # Embedded application
  my $stash = {};
  if ($tx && (my $sub = $tx->can('stash'))) { ($stash, $tx) = ($tx->$sub, $tx->tx) }

  # Build default controller
  my $defaults = $self->defaults;
  @$stash{keys %$defaults} = values %$defaults;
  my $c = $self->controller_class->new(app => $self, stash => $stash, tx => $tx);
  $c->{tx} ||= $self->build_tx;

  return $c;
}

sub build_tx {
  my $self = shift;

  my $tx  = Mojo::Transaction::HTTP->new;
  my $max = $self->max_request_size;
  $tx->req->max_message_size($max) if defined $max;
  $self->plugins->emit_hook(after_build_tx => $tx, $self);

  return $tx;
}

sub config   { Mojo::Util::_stash(config   => @_) }
sub defaults { Mojo::Util::_stash(defaults => @_) }

sub dispatch {
  my ($self, $c) = @_;

  my $plugins = $self->plugins->emit_hook(before_dispatch => $c);

  my $stash = $c->stash;
  return if $stash->{'mojo.rendered'};

  # Try to find a static file
  my $tx = $c->tx;
  $self->static->dispatch($c) and $plugins->emit_hook(after_static => $c) unless $tx

  # Start timer (ignore static files)
  $c->helpers->log->trace(sub {
    my $req    = $c->req;
    my $method = $req->method;
    my $path   = $req->url->path->to_abs_string;
    $c->helpers->timing->begin('mojo.timer');
    return qq{$method "$path"};
  }) unless $stash->{'mojo.static'};
```

```
124     # Routes
125     $plugins->emit_hook(before_routes => $c);
126     $c->helpers->reply->not_found unless $tx->res->code || $self->routes->dispatch($c
127   }
128
129   sub handler {
130     my $self = shift;
131
132     # Dispatcher has to be last in the chain
133     ++$self->{dispatch}
134       and $self->hook(around_action   => \&_action)
135       and $self->hook(around_dispatch => sub { $_[1]->app->dispatch($_[1]) })
136       unless $self->{dispatch};
137
138     # Process with chain
139     my $c = $self->build_controller(@_);
140     $self->plugins->emit_chain(around_dispatch => $c);
141
142     # Delayed response
143     $c->helpers->log->trace('Nothing has been rendered, expecting delayed response (s
144       unless $c->stash->{'mojo.rendered'};
145   }
146
147   sub helper { shift->renderer->add_helper(@_) }
148
149   sub hook { shift->plugins->on(@_) }
150
151   sub new {
152     my $self = shift->SUPER::new((ref $_[0] ? %{shift()} : @_), @Mojo::Server::ARGS_O
153
154     my $home = $self->home;
155     push @{$self->renderer->paths}, $home->child('templates')->to_string;
156     push @{$self->static->paths},   $home->child('public')->to_string;
157
158     # Default to controller and application namespace
159     my $controller = "@{[ref $self]}::Controller";
160     my $r          = $self->preload_namespaces([$controller])->routes->namespaces([$c
161
162     $self->plugin($_) for qw(HeaderCondition DefaultHelpers TagHelpers EPLRenderer EPL
163
164     # Exception handling should be first in chain
165     $self->hook(around_dispatch => \&_exception);
166
167     $self->startup;
168     $self->warmup;
169
170     return $self;
171   }
172
173   sub plugin {
174     my $self = shift;
175     $self->plugins->register_plugin(shift, $self, @_);
176   }
177
178   sub server { $_[0]->plugins->emit_hook(before_server_start => @_[1, 0]) }
179
180   sub start {
181     my $self = shift;
182     $_->warmup for $self->static, $self->renderer;
183     return $self->commands->run(@_ ? @_ : @ARGV);
184   }
185
```

```
186  sub startup { }
187
188  sub warmup { Mojo::Loader::load_classes $_ for @{shift->preload_namespaces} }
189
190  sub _action {
191    my ($next, $c, $action, $last) = @_;
192    my $val = $action->($c);
193    $val->catch(sub { $c->helpers->reply->exception(shift) }) if Scalar::Util::blesse
194    return $val;
195  }
196
197  sub _die { CORE::die ref $_[0] ? $_[0] : Mojo::Exception->new(shift)->trace }
198
199  sub _exception {
200    my ($next, $c) = @_;
201    local $SIG{__DIE__} = \&_die;
202    $c->helpers->reply->exception($@) unless eval { $next->(); 1 };
203  }
204
205  1;
206    Show 1014 lines of Pod
```