NAME

Catalyst::Manual::CatalystAndMoose - How Catalyst 5.8+ and Moose relate

DESCRIPTION

Since version 5.8, the core of Catalyst is based on Moose. Although the developers went through great lengths to allow for a seamless transition, there are still a few things to keep in mind when trying to exploit the power of Moose in your Catalyst application.

This document provides you with a short overview of common caveats and best practices for using Moose-based classes within Catalyst.

THE CONTEXT CLASS

A Moose-ified version of the context class should look like this:

```
    package MyApp;
    use Moose;
    use namespace::autoclean;
    use Catalyst (
        # your roles and plugins
    );
    extends 'Catalyst';

    # If you want to use method modifiers to adjust the setup process, (e.g. setup_finalize)
    # they must be here, before the call to setup (advanced users only)

    $app->config( name => 'MyApp' );
    $app->setup;

    # method modifiers generally must be created after setup because otherwise they will
    # conflict with plugin overrides

    after 'finalize' => sub {
        my $c = shift;
        $c->log->info( 'done!' );
    }
```

You should also be aware that roles in `$c->setup` are applied after the last plugin with all the benefits of using a single with() statement in an ordinary Moose class.

Your class is automatically made immutable at the end of the current file.

CAVEAT: Using roles in `$c->setup` was implemented in Catalyst version 5.80004. In prior versions you might get away with

```
    after 'setup_plugins' => sub{ with(
        # your roles
    )};

    $app->setup(
        # your plugins
    );
```

but this is discouraged and you should upgrade to 5.80004 anyway, because it fixes a few important regressions against 5.71

CAVEAT: Using roles in `$c->setup` will not currently allow you to pass parameters to roles, or perform conflict resolution. Conflict detection still works as expected.

ACCESSORS

Most of the request-specific attributes like `$c->stash`, `$c->request` and `$c->response` have been converted to Moose attributes but without type constraints, attribute helpers or builder methods. This ensures that Catalyst 5.8 is fully backwards compatible to applications using the published API of Catalyst 5.7 but slightly limits the gains that could be had by wielding the full power of Moose attributes.

Most of the accessors to information gathered during compile time (such as configuration) are managed by `Catalyst::ClassData`, which is a Moose-aware version of Class::Data::Inheritable but not compatible with MooseX::ClassAttribute.

ROLES AND METHOD MODIFIERS

Since the release of Catalyst version 5.8, the only reason for creating a Catalyst extension as a plugin is to provide backward compatibility to applications still using version 5.7.

If backward compatibility is of no concern to you, you could as easily rewrite your plugins as roles and enjoy all the benefits of automatic method re-dispatching of `before` and `after` method modifiers, naming conflict detection and generally cleaner code.

NOTE

Plugins and roles should never use

```
    after 'setup' => sub { ... } # wrong
```

(or any other method of hooking the setup method) but rely on

```
    after 'setup_finalize' => sub { ... } # this will work
```

to run their own setup code if needed. If they need to influence the setup process itself, they can modify `setup_dispatcher()`, `setup_engine()`, `setup_stats()`, `setup_components()` and `setup_actions()`, but this should be done with due consideration and as late as possible.

CONTROLLERS

To activate Catalyst's action attributes, Moose-ified controller classes need to extend Catalyst::Controller at compile time, before the actions themselves are declared:

```
    package Catalyst::Controller::Root;
    use Moose;
    use namespace::autoclean;

    BEGIN { extends 'Catalyst::Controller'; }
```

Controller Roles

It is possible to use roles to apply method modifiers on controller actions from 5.80003 onwards, or use modifiers in your controller classes themselves. For example

```
  package MyApp::Controller::Foo;
  use Moose;
  use namespace::autoclean;
  BEGIN { extends 'Catalyst::Controller' };

  sub foo : Local {
      my ($self, $c) = @_;
      $c->res->body('Hello ');
  }
  after foo => sub {
      my ($self, $c) = @_;
      $c->res->body($c->res->body . 'World');
  };
```

It is possible to have action methods with attributes inside Moose roles, using MooseX::MethodAttributes, example:

```
  package MyApp::ControllerRole;
  use MooseX::MethodAttributes::Role;
  use namespace::autoclean;

  sub foo : Local {
      my ($self, $c) = @_;
      ...
  }

  package MyApp::Controller::Foo;
  use Moose;
  use namespace::autoclean;
  BEGIN { extends 'Catalyst::Controller' };

  with 'MyApp::ControllerRole';
```

AUTHORS

Catalyst Contributors, see Catalyst.pm

COPYRIGHT

syntax highlighting: no syntax highlighting ▼