NAME

Catalyst::Manual::Actions - Catalyst Reusable Actions

DESCRIPTION

This section of the manual describes the reusable action system in Catalyst, how such actions work, descriptions of some existing ones, and how to write your own. Reusable actions are attributes on Catalyst methods that allow you to decorate your method with functions running before or after the method call. This can be used to implement commonly used action patterns, while still leaving you full freedom to customize them.

USING ACTIONS

This is pretty simple. Actions work just like the normal dispatch attributes you are used to, like Local or Private:

```
sub Hello :Local :ActionClass('SayBefore') {
      $c->res->output( 'Hello '.$c->stash->{what} );
}
```

In this example, we expect the SayBefore action to magically populate stash with something relevant before `Hello` is run. In the next section we'll show you how to implement it. If you want it in a namespace other than Catalyst::Action you can prefix the action name with a '+', for instance '+Foo::SayBefore', or if you just want it under your application namespace instead, use MyAction, like MyAction('SayBefore').

WRITING YOUR OWN ACTIONS

Implementing the action itself is almost as easy. Just use Catalyst::Action as a base class and decorate the `execute` call in the Action class:

```
package Catalyst::Action::MyAction;
use Moose;
use namespace::autoclean;
```

```
extends 'Catalyst::Action';

before 'execute' => sub {
  my ( $self, $controller, $c, $test ) = @_;
  $c->stash->{what} = 'world';
};

after 'execute' => sub {
    my ( $self, $controller, $c, $test ) = @_;
    $c->stash->{foo} = 'bar';
};

__PACKAGE__->meta->make_immutable;
```

Pretty simple, huh?

ACTION ROLES

You can only have one action class per action, which can be somewhat inflexible.

The solution to this is to use Catalyst::Controller::ActionRole, which would make the example above look like this:

```
package Catalyst::ActionRole::MyActionRole;
use Moose::Role;

before 'execute' => sub {
  my ( $self, $controller, $c, $test ) = @_;
  $c->stash->{what} = 'world';
};

after 'execute' => sub {
    my ( $self, $controller, $c, $test ) = @_;
    $c->stash->{foo} = 'bar';
};

1;
```

and this would be used in a controller like this:

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

sub foo : Does('MyActionRole') {
    my ($self, $c) = @_;
}

1;
```

EXAMPLE ACTIONS

Catalyst::Action::RenderView

This is meant to decorate end actions. It's similar in operation to Catalyst::Plugin::DefaultEnd, but allows you to decide on an action level rather than on an application level where it should be run.

Catalyst::Action::REST

Provides additional syntax for dispatching based upon the HTTP method of the request.

EXAMPLE ACTIONROLES

Catalyst::ActionRole::ACL

Provides ACLs for role membership by decorating your actions.

AUTHORS

Catalyst Contributors, see Catalyst.pm

COPYRIGHT