NAME
DESCRIPTION
Initialization
Request Lifecycle
AUTHORS
COPYRIGHT

NAME

Catalyst::Manual::Internals - Catalyst Internals

DESCRIPTION

This document provides a brief overview of the internals of Catalyst. As Catalyst is still developing rapidly, details may become out of date: please treat this as a guide, and look at the source for the last word.

The coverage is split into initialization and request lifecycle.

Initialization

Catalyst initializes itself in two stages:

1. When the Catalyst module is imported in the main application module, it stores any options.
2. When `__PACKAGE__->setup` is called, it evaluates any options stored (`-Debug`), and makes the application inherit from Catalyst (if that hasn't already been done with an explicit `use base 'Catalyst';` or `extends 'Catalyst';`. Any specified plugins are then loaded, the application module is made to inherit from the plugin classes. It also sets up a default log object and ensures that the application module inherits from `Catalyst` and from the selected specialized Engine module.
3. Catalyst automatically loads all components it finds in the `$class::Controller`, `$class::C`, `$class::Model`, `$class::M`, `$class::View` and `$class::V` namespaces (using `Module::Pluggable`). As each is loaded, if it has a COMPONENT method then this method will be called, and passed that component's configuration. It then returns an instance of the component, which becomes the `$self` when methods in that component are called later.
4. Each controller has it's `register_actions` method called. At this point, the subroutine attributes are retrieved from the MooseX::MethodAttributes::Role::Meta::Map, parsed, and used to build instances of Catalyst::Action, which are then registered with the dispatcher.

Request Lifecycle

For each request Catalyst builds a *context* object, which includes information about the request, and then searches the action table for matching actions.

The handling of a request can be divided into three stages: preparation of the context, processing of the request, and finalization of the response. These are the steps of a Catalyst request in detail; every step can be overloaded to extend Catalyst.

```
handle_request
  prepare
    prepare_request
    prepare_connection
```

```
      prepare_query_parameters
      prepare_headers
      prepare_cookies
      prepare_path
      prepare_body (unless parse_on_demand)
        prepare_body_parameters
        prepare_parameters
        prepare_uploads
      prepare_action
    dispatch
    finalize
      finalize_uploads
      finalize_error (if one happened)
      finalize_headers
        finalize_cookies
      finalize_body
```

These steps are normally overloaded from engine classes, and may also be extended by plugins. For more on extending Catalyst, see Catalyst::Manual::ExtendingCatalyst.

The engine class populates the Catalyst request object with information from the underlying layer (PSGI) during the prepare phase, then push the generated response information down to the underlying layer during the finalize phase.

AUTHORS

Catalyst Contributors, see Catalyst.pm

COPYRIGHT