NAME

Dancer::Template::Abstract − abstract class for Dancer's template engines

VERSION

version 1.3140

DESCRIPTION

This class is provided as a base class for each template engine. Any template engine must inherit from it and provide a set of methods described below.

TEMPLATE TOKENS

By default Dancer injects some tokens (or variables) to templates. The available tokens are:

`perl_version`
: The current running Perl version.

`dancer_version`
: The current running Dancer version.

`settings`
: Hash to access current application settings.

`request`
: Hash to access your current request.

`params`
: Hash to access your request parameters.

`vars`
: Hash to access your defined variables (using `vars`).

`session`
: Hash to access your session (if you have session enabled)

INTERFACE

init()

The template engine can overload this method if some initialization stuff has to be done before the template engine is used.

The base class provides a plain *init()* method that only returns true.

default_tmpl_ext()

Template class that inherits this class should override this method to return a default template extension, example: for Template::Toolkit it returns ''tt'' and for HTML::Mason it returns ''mason''. So when you call `template 'index';` in your dispatch code, Dancer will look for a file 'index.tt' or 'index.mason' based on the template you use.

Note 1: when returning the extension string, please do not add a dot in front of the extension as Dancer will do that.

Note 2: for backwards compatibility abstract class returns ''tt'' instead of throwing an exception 'method not implemented'.

User would be able to change the default extension using the `<extension>` configuration variable on the template configuration. For example, for the default (`Simple`) engine:

```
template: "simple"
engines:
  simple:
    extension: 'tmpl'
```

view($view)

The default behavior of this method is to return the path of the given view, appending the default template extension (either the value of the `extension` setting in the configuration, or the value returned by `default_tmpl_ext`) if it is not present in the view name given and no layout template with that exact name existed. (In other words, given a layout name `main`, if `main` exists in the layouts dir, it will be used; if not, `main.tmpl` (where `tmpl` is the value of the `extension` setting, or the value returned by `default_tmpl_ext`) will be looked for.)

view_exists($view_path)

By default, Dancer::Template::Abstract checks to see if it can find the view file calling `view_exists($path_to_file)`. If not, it will generate a nice error message for the user.

If you are using extending Dancer::Template::Abstract to use a template system with multiple document roots (like Text::XSlate or Template), you can override this method to always return true, and therefore skip this check.

layout($layout, `$tokens`, `$content`)

The default behavior of this method is to merge a content with a layout. The layout file is looked for with similar logic as per `view` − an exact match first, then attempting to append the default template extension, if the view name given did not already end with it.

render($self, `$template`, `$tokens`)

This method must be implemented by the template engine. Given a template and a set of tokens, it returns a processed string.

If `$template` is a reference, it's assumed to be a reference to a string that contains the template itself. If it's not a reference, it's assumed to be the path to template file, as a string. The render method will then have to open it and read its content (Dancer::FileUtils::read_file_content does that job).

This method's return value must be a string which is the result of the interpolation of `$tokens` in `$template`.

If an error occurs, the method should trigger an exception with `die()`.

Examples :

```
# with a template as a file
$content = $engine->render('/my/template.txt', { var => 42 };

# with a template as a scalar
my $template = "here is <% var %>";
$content = $engine->render(\$template, { var => 42 });
```

AUTHOR

This module has been written by Alexis Sukrieh, see Dancer for details.

AUTHOR

Dancer Core Developers

COPYRIGHT AND LICENSE