Package: jinjar (via r-universe)

November 27, 2024

Title Template Engine Inspired by 'Jinja'

Version 0.3.1.9000

Description Template engine powered by the 'inja' C++ library. Users write a template document, using syntax inspired by the 'Jinja' Python package, and then render the final document by passing data from R. The template syntax supports features such as variables, loops, conditions and inheritance.

License MIT + file LICENSE

URL https://davidchall.github.io/jinjar/, https://github.com/davidchall/jinjar

BugReports https://github.com/davidchall/jinjar/issues

Imports cli, fs, jsonlite, rlang (>= 1.0.0)

Suggests knitr, rmarkdown, testthat

LinkingTo cpp11

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Config/pak/sysreqs make

Repository https://davidchall.r-universe.dev

RemoteUrl https://github.com/davidchall/jinjar

RemoteRef HEAD

RemoteSha 5827fdabdb7e7752c8b0f23a7436950b58c4c48e

Contents

`jinjar_config` *Configure the templating engine*

Description

Create an object to configure the templating engine behavior (e.g. customize the syntax). The default values have been chosen to match the Jinja defaults.

Usage

```
jinjar_config(
  loader = NULL,
  block_open = "{%",
  block_close = "%}",
  variable_open = "{{",
  variable_close = "}}",
  comment_open = "{#",
  comment_close = "#}",
  line_statement = NULL,
  trim_blocks = FALSE,
  lstrip_blocks = FALSE,
  ignore_missing_files = FALSE
)

default_config()
```

Arguments

`loader` How the engine discovers templates. Choices:

- `NULL` (default), disables search for templates.
- Path to template directory.
- A `loader` object.

`block_open`, `block_close`
The opening and closing delimiters for control blocks. Default: `"{%"` and `"%}"`.

`variable_open`, `variable_close`
The opening and closing delimiters for print statements. Default: `"{{"` and `"}}"`.

comment_open, comment_close
: The opening and closing delimiters for comments. Default: "{#" and "#}".

line_statement
: The prefix for an inline statement. If NULL (the default), inline statements are disabled.

trim_blocks
: Remove first newline after a block. Default: FALSE.

lstrip_blocks
: Remove inline whitespace before a block. Default: FALSE.

ignore_missing_files
: Ignore include or extends statements when the auxiliary template cannot be found. If FALSE (default), then an error is raised.

Value

A "jinjar_config" object.

Note

The equivalent Jinja class is Environment, but this term has special significance in R (see environment()).

Examples

```
jinjar_config()
```

---

loader *Template loaders*

Description

Loaders are responsible for exposing templates to the templating engine.

path_loader() loads templates from a directory in the file system.

package_loader() loads templates from a directory in an R package.

list_loader() loads templates from a named list.

Usage

```
path_loader(...)

package_loader(package, ...)

list_loader(x)
```

Arguments

...
: Strings specifying path components.

package
: Name of the package in which to search.

x
: Named list mapping template names to template sources.

Value

A "jinjar_loader" object.

See Also

The loader is an argument to jinjar_config().

Examples

```
path_loader(getwd())

package_loader("base", "demo")

list_loader(list(
  header = "Title: {{ title }}",
  content = "Hello {{ person }}!"
))
```

---

parse *Parse a template*

Description

Sometimes you want to render multiple copies of a template, using different sets of data variables. parse_template() returns an intermediate version of the template, so you can render() repeatedly without re-parsing the template syntax.

Usage

```
parse_template(.x, .config)

## S3 method for class 'character'
parse_template(.x, .config = default_config())

## S3 method for class 'fs_path'
parse_template(.x, .config = default_config())
```

Arguments

.x	The template. Choices: • A template string. • A path to a template file (use fs::path()).
.config	The engine configuration. The default matches Jinja defaults, but you can use jinjar_config() to customize things like syntax delimiters, whitespace control, and loading auxiliary templates.

Value

A "jinjar_template" object.

See Also

- render() to render the final document using data variables.
- print() for pretty printing.
- vignette("template-syntax") describes how to write templates.

Examples

```
x <- parse_template("Hello {{ name }}!")

render(x, name = "world")
```

---

print.jinjar_template *Print a template*

Description

Once a template has been parsed, it can be printed with color highlighting of the templating blocks.

Usage

```
## S3 method for class 'jinjar_template'
print(x, ..., n = 10)
```

Arguments

x	A parsed template (use parse_template()).
...	These dots are for future extensions and must be empty.
n	Number of lines to show. If Inf, will print all lines. Default: 10.

Examples

```
input <- '<!DOCTYPE html>
<html lang="en">
<head>
    <title>{{ title }}</title>
</head>
<body>
    <ul id="navigation">
    {% for item in navigation -%}
        <li><a href="{{ item.href }}">{{ item.caption }}</a></li>
    {% endfor -%}
    </ul>
{# a comment #}
```

```
</body>
</html>'

x <- parse_template(input)

print(x)

print(x, n = Inf)
```

---

render *Render a template*

Description

Data is passed to a template to render the final document.

Usage

```
render(.x, ...)

## S3 method for class 'character'
render(.x, ..., .config = default_config())

## S3 method for class 'fs_path'
render(.x, ..., .config = default_config())

## S3 method for class 'jinjar_template'
render(.x, ...)
```

Arguments

.x	The template. Choices: • A template string. • A path to a template file (use fs::path()). • A parsed template (use parse_template()).
...	<dynamic-dots> Data passed to the template. By default, a length-1 vector is passed as a scalar variable. Use I() to declare that a vector should be passed as an array variable. This preserves a length-1 vector as an array.
.config	The engine configuration. The default matches Jinja defaults, but you can use jinjar_config() to customize things like syntax delimiters, whitespace control, and loading auxiliary templates.

Value

String containing rendered template.

See Also

- parse_template() supports parsing a template once and rendering multiple times with different data variables.
- vignette("template-syntax") describes how to write templates.

Examples

```
# pass data as arguments
render("Hello {{ name }}!", name = "world")

# pass length-1 vector as array
render("Hello {{ name.0 }}!", name = I("world"))

# pass data programmatically
params <- list(name = "world")
render("Hello {{ name }}!", !!!params)

# render template file
## Not run:
render(fs::path("template.txt"), name = "world")

## End(Not run)
```

Index