Package ‘pater’

December 5, 2025

Title Turn a URL Pathname into a Regular Expression

Version 1.0.0

Description R's implementation of the JavaScript library 'path-to-regexp',
it aims to provide R web frameworks features such as parameter handling
among other URL path utilities.

License MIT + file LICENSE

URL `https://github.com/JulioCollazos64/pater`

BugReports `https://github.com/JulioCollazos64/pater/issues`

Encoding UTF-8

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0), stringr

Config/testthat/edition 3

Imports utils

NeedsCompilation no

Author Julio Collazos [aut, cre],
path-to-regexp contributors [cph] (see
<`https://github.com/pillarjs/path-to-regexp/graphs/contributors`>)

Maintainer Julio Collazos <`amarullazo626@gmail.com`>

Repository CRAN

Date/Publication 2025-12-05 20:20:02 UTC

Contents

compile *Build a function for transforming parameters into a valid path*

Description

The output function will have one parameter in which you can specify the parameters using a named list.

Usage

```
compile(path, delimiter = "/", encode)
```

Arguments

`path`	A character vector or a `tokenData` object.
`delimiter`	A character vector of length 1. Specifies the delimiter for the path segments.
`encode`	Function to encode input strings. Defaults to utils::URLencode with the parameter reserved set to TRUE.

Value

A function.

Examples

```
toPath <- compile("/path/to/resource/:Id")
toPath(list(Id = "2"))

toPath <- compile("public/*files")
toPath(list(files = c("js", "hi.js")))
```

match *Build a function for matching pathnames against a pathname specification*

Description

Build a function for matching pathnames against a pathname specification

Usage

```
match(path, decode = NULL, delimiter = "/", ...)
```

Arguments

`path`	A character vector of length 1. A pathname specification.
`decode`	A function for decoding a string or FALSE to disable it.
`delimiter`	A character vector of length 1. Specifies the delimiter for the path segments.
`...`	Additional parameters for `pathToRegexp` or `parse`.

Value

A function.

Examples

```
path <- "/users/:userId/books/:bookId/*public"
fn <- match(path)
p <- fn("/users/User1/books/Id1/2/3")
p

path <- "/path/resource"
fn <- match(path)
fn("/resource/path")
```

`parse` *Decompose a pathname*

Description

This function decomposes a given pathname into meaningful tokens, see Details.

Usage

```
parse(path, encodePath = identity)
```

Arguments

`path`	A character vector of length 1.
`encodePath`	A function to encode characters, defaults to the `identity` function.

Details

The `parse` function returns a `tokenData` object which contains a list of tokens from the path you provided. A token is a meaningful portion of a given pathname, it can contain so called "parameters", which are placeholders that will be filled when a new HTTP request comes in, these parameters may or not expand a whole path segment, meaning that two parameters can share the same path segment, a token type can be any of the following:

- text: Represents a fixed path portion , e.g "/path/" or "/path/resource".
- param: Represent a dynamic path portion, e.g. "/path/:Id" or "/path/:from-:to". Must be named.
- wildcard: Similar to param but can "expand" itself into more than one path segment, e.g. "/public/*files". Must be named
- group: Represents an optional path portion, either an optional parameter or optional text, e.g. "/path{/:optional}" or "/user{s}".

`pater` uses this special syntax as some regex features are not available, e.g. "/users?" doesn't work, this was done in the original implementation for security reasons. If you want to specify one of these in your path use double backlash, "/users\?"

Value

An object of class `tokenData`.

Examples

```
# A "fixed" path
path <- "/path/resource"
parse(path)

# Parameters
path <- "/path/to/:resourceId"
parse(path)

# Wildcard
path <- "/path/*files"
parse(path)

# Group or "optional"
path <- "/path{s}"
parse(path)

# Error because of regex feature not supported
## Not run:
 path <- "/paths?"
 parse(path)

## End(Not run)

# Escape it
path <- "/paths\\?"
parse(path)
```

`pathToRegexp` *Build a regular expression for matching strings against pathnames*

Description

Build a regular expression for matching strings against pathnames

Usage

```
pathToRegexp(
  path,
  end = TRUE,
  sensitive = FALSE,
  trailing = TRUE,
  delimiter = "/",
  ...
)
```

Arguments

`path`	A character vector, TokenData, or a list of strings and TokenData objects.
`end`	A logical vector of length 1. Whether to add a construct to the regular expression to check for a complete end of string match. Defaults to TRUE.
`sensitive`	A logical vector of length 1. Whether resulting regex will be case sensitive. Defaults to FALSE.
`trailing`	A logical vector of length 1. Whether or not match trailing path. Defaults to TRUE.
`delimiter`	A character vector of length 1. Specifies the delimiter for the path segments. Defaults to "/"
`...`	Additional parameters for `parse`.

Value

A list with two elements: a regular expression and a list of keys.

Examples

```
path <- "/hello/world"
regex <- pathToRegexp(path)$pattern
grepl(regex,"/hello/world", perl = TRUE)
grepl(regex,"/hello/world/", perl = TRUE)

path <- "/hello/:world"
regex <- pathToRegexp(path)$pattern
grepl(regex, "/hello/world", perl = TRUE)
grepl(regex, "/hello/path", perl = TRUE)
```

```
# Taken from https://expressjs.com/en/guide/routing.html
path <- "/flights/:from-:to"
regex <- pathToRegexp(path)$pattern
grepl(regex, "/flights/a-b", perl = TRUE)
grepl(regex, "/flights/a-b/", perl = TRUE)

# Taken from https://expressjs.com/en/guide/routing.html
path <- "/users/:userId/books/:bookId"
regex <- pathToRegexp(path)$pattern
grepl(regex, "/users/1/books/2", perl = TRUE)
grepl(regex, "/users/1/books/2/", perl = TRUE)

path <- "/plantae/:genus.:species"
regex <- pathToRegexp(path)$pattern
grepl(regex, "/plantae/a.b", perl = TRUE)
grepl(regex, "/plantae/a.b/", perl = TRUE)

# Will match any route that starts with "/public/"
path <- "/public/*files"
regex <- pathToRegexp(path)$pattern
grepl(regex,"/public/format1", perl = TRUE)
grepl(regex,"/public/format2/format3", perl = TRUE)

# trailing
path <- "/user/:userId"
regex <- pathToRegexp(path, trailing = FALSE)$pattern
grepl(regex, "/user/1", perl = TRUE) # TRUE
grepl(regex, "/users/1/", perl = TRUE) # FALSE

# sensitive
path <- "/user"
regex <- pathToRegexp(path, sensitive = TRUE)$pattern
grepl(regex, "/user", perl = TRUE) # TRUE
grepl(regex, "/USER", perl = TRUE) # FALSE

# end
path <- "/users"
regex1 <- pathToRegexp(path, trailing = FALSE, end = FALSE)$pattern
regex2 <- pathToRegexp(path, trailing = FALSE, end = TRUE)$pattern
if(require("stringr")){
  str_extract("/users////", regex1) # "/users"
  str_extract("/users////", regex2) # NA
}
```

`stringifyTokens` *From `tokenData` object to a character vector*

Description

The inverse of the `parse()` function.

Usage

```
stringifyTokens(tokens)
```

Arguments

`tokens`	An object of class `tokenData`

Value

A character vector of length 1.

Examples

```
tokens <- parse("/path/to/resource/:Id")
path <- stringifyTokens(tokens)
identical(path, "/path/to/resource/:Id") # TRUE
```

Index