Package: routing (via r-universe)

May 12, 2026

Title 'Express.js' Like Routing for R Web Frameworks

Version 1.1.0

Description It aims to provide R web frameworks a routing mechanism of HTTP requests inspired by the battle tested 'Express.js' web framework.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE, r6 = TRUE)

RoxygenNote 7.3.3

Imports httpcode, pater, promises, R6, utils

Suggests curl, httpuv, later, testthat (>= 3.0.0), yyjsonr

Config/testthat/edition 3

URL https://github.com/JulioCollazos64/routing

BugReports https://github.com/JulioCollazos64/routing/issues

Repository https://juliocollazos64.r-universe.dev

Date/Publication 2026-05-12 01:21:00 UTC

RemoteUrl https://github.com/juliocollazos64/routing

RemoteRef HEAD

RemoteSha 8cc5010c5b4db5683929ce40673be5964fadac0c

Contents

finalHandler *Final Handler*

Description

Returns a `function(err)` to be used as the `callback` argument of Router`$handle()`. It is invoked when the router exhausts its stack without sending a response, or when an unhandled occurs.

Behaviour:

- **No error** and no response sent – writes a `404 Not Found` HTML response. The body message is `"Cannot <METHOD> <PATH_INFO>"`.
- **Error** – derives the HTTP status from `err$status` (must be a numeric 4xx or 5xx value); falls back to `res$status`, and then to `500`. The body contains the standard HTTP phrase for the status code.

In all cases the response is an HTML document and includes the security headers `Content-Security-Policy: default-s` and `X-Content-Type-Options: nosniff`. Pre-existing `Content-Encoding`, `Content-Language`, and `Content-Range` headers are removed.

Usage

```
finalHandler(req, res, options)
```

Arguments

`req`	(`environment`) Rook request environment.
`res`	(`Response`) Response object.
`options`	(`list`) Reserved for future use; currently ignored.

Value

A `function(err)` that sends the final HTTP response.

Router *Router*

Description

A port of the pillarjs/router package for R. Maintains an ordered stack of layers; each layer pairs a path pattern with a middleware function, a nested `Router`, or a `Route` object. Middleware and nested routers are added via `$use()`; routes are added via `$route()` or the HTTP-verb shortcuts. When a request arrives, the stack is walked in order and each matching layer is invoked until the request is handled or the stack is exhausted.

Important details:

- `forward()` **instead of** `next()` – `next` is a reserved word in R. `forward("route")` and `forward("router")` work identically to their Express counterparts.
- `forward` **is implicit** – handlers do not need to declare `forward` as an argument to call it; it is automatically injected into the handler's formals if absent. `function(req, res) { forward() }` works just as well as `function(req, res, forward) { forward() }`.
- `forward` **is auto-called** – if a handler returns without calling `forward()` or sending a response, `forward()` is called automatically.
- **Error handlers take three arguments** – write error handlers as `(err, req, res)`; `forward` is injected automatically as the fourth argument.

HTTP verb shortcuts:
`$get()`, `$post()`, `$put()`, `$delete()`, etc. (one per HTTP verb) and `$all()` are convenience wrappers with signature `(path, ...)` equivalent to `router$route(path)$<verb>(...)`. They return `self` invisibly for chaining.

Methods

Public methods:

- `Router$new()`
- `Router$handle()`
- `Router$use()`
- `Router$route()`
- `Router$param()`
- `Router$static()`
- `Router$getStack()`
- `Router$clone()`

Method `new()`**:** Creates a new `Router`.

Usage:

```
Router$new(caseSensitive = FALSE, mergeParams = FALSE, strict = FALSE)
```

Arguments:

`caseSensitive (logical(1))`
: When `TRUE`, path matching is case-sensitive (`/Foo` does not match `/foo`). Default `FALSE`.

`mergeParams (logical(1))`
: When `TRUE`, `req$params` from a parent router are merged with those of this router instead of being replaced. Default `FALSE`.

`strict (logical(1))`
: When `TRUE`, trailing slashes are significant (`/foo/` does not match `/foo`). Default `FALSE`.

Returns: A new `Router` object.

Method `handle()`**:** Dispatches a request through the router's layer stack. Normally called by a server or a parent router rather than directly.

Usage:

```
Router$handle(req, res = NULL, callback = NULL)
```

Arguments:

`req` (`environment`)
: Rook request environment.

`res` (`Response`)
: Response object.

`callback` (`function`)
: Called when no layer matched or an unhandled error occurred.

Method `use()`**:** Mounts one or more middleware handlers, optionally scoped to a path prefix. A Router may be passed and will be wrapped automatically. Functions whose first parameter is named `err` are treated as error handlers.

Usage:

```
Router$use(...)
```

Arguments:

`...` (`function | list`)
: An optional leading `character(1)` path prefix, followed by one or more handler functions, nested lists of functions, or another Router.

Returns: `self` invisibly.

Method `route()`**:** Creates a new `Route` for `path` and appends it to the stack.

Usage:

```
Router$route(path)
```

Arguments:

`path` (`character(1)`)
: Path pattern.

Returns: The new `Route` invisibly.

Method `param()`**:** Registers a callback triggered whenever a named route parameter is present in a matched route. The callback runs before the route handler.

The callback signature is `function(req, res, value, name)`:

- `value` – the captured value of the parameter.
- `name` – the parameter name (character).

Usage:

```
Router$param(name, fn)
```

Arguments:

`name` (`character(1)`)
: Name of the route parameter to watch.

`fn` (`function`)
: Callback with signature `function(req, res, value, name)`.

Returns: `self` invisibly.

Examples:

```
router <- Router$new()

router$param("id", function(req, res, value, name) {
  user <- findUser(value)
  req$user <- list(id = value, name = user$name)
  forward()
})

router$get("/user/:id", function(req, res) {
  res$send(paste("user:", req$user$name))
})
```

Method `static()`**:** Registers a directory of static files to be served at a given URL prefix. Static files are served directly by httpuv's background I/O thread without invoking R.

Usage:

```
Router$static(root, url, ...)
```

Arguments:

`root` (`character(1)`)
: Local filesystem path to the directory containing the files to serve.

`url` (`character(1)`)
: URL prefix at which the files will be available (e.g. `"/static"`).

`...` Additional arguments passed to `httpuv::staticPath()`.

Returns: `self` invisibly.

Method `getStack()`**:** Returns the internal layer stack.

Usage:

```
Router$getStack()
```

Returns: `list` of `Layer` objects.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Router$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
router <- Router$new()

router$get(
  "/get",
```

```
  function(req, res) {
    res$send("Hello there!")
  }
)$post(
  "/post",
  function(req, res) {
    res$send("Goodbye!")
  }
)

router$get(
  "/hello",
  function(req, res) {
    forward()
  },
  function(req, res) {
    res$send("Hello!")
  }
)

router$post("/bye", function(req, res) {
  res$send("Bye!")
})

router$route("/hi")$get(function(req, res) {
  res$send("handling a GET request!")
})$post(function(req, res) {
  res$send("handling a POST request!")
})

router$get(
  c("/path", "/another-path"),
  function(req, res) {
    res$send("Hola!")
  }
)

## ------------------------------------------------
## Method `Router$param`
## ------------------------------------------------

router <- Router$new()

router$param("id", function(req, res, value, name) {
  user <- findUser(value)
  req$user <- list(id = value, name = user$name)
  forward()
})

router$get("/user/:id", function(req, res) {
  res$send(paste("user:", req$user$name))
})
```

Index