Package 'beakr'

May 7, 2026

Type Package

Title A Minimalist Web Framework for R

Version 0.4.4

Author Hans Martin [aut],
Jonathan Callahan [aut, cre]

Maintainer Jonathan Callahan <jonathan.s.callahan@gmail.com>

Description A minimalist web framework for developing application programming interfaces in R that provides a flexible framework for handling common HTTP-requests, errors, logging, and an ability to integrate any R code as server middle-ware.

License GPL-3

URL https://github.com/MazamaScience/beakr

BugReports https://github.com/MazamaScience/beakr/issues

ByteCompile TRUE

Depends R (>= 3.1.0)

Imports R6, base64enc, httpuv, jsonlite, magrittr, mime, stringr, webutils

Suggests knitr, testthat, rmarkdown

Encoding UTF-8

RoxygenNote 7.3.1

NeedsCompilation no

Repository CRAN

Date/Publication 2025-09-05 16:10:08 UTC

Contents

`Beakr` *Beakr Application Class*

Description

A ‘Beakr‘ object defines a web server instance using the [‘httpuv‘] package. It provides the main entry point for creating and starting a Beakr application, wrapping a [‘Router‘] and exposing lifecycle methods.

Format

An [‘R6::R6Class‘] generator for ‘Beakr‘ objects.

Public fields

`name` Application name. If ‘NULL‘, a random name is set in ‘$initialize()‘.

`router` The ‘Router‘ instance used to handle requests.

`server` The underlying ‘httpuv‘ server object (once started).

Methods

Public methods:

- `Beakr$appDefinition()`
- `Beakr$new()`
- `Beakr$start()`
- `Beakr$print()`

- Beakr$clone()

Method `appDefinition()`**:** Build the application definition passed to **httpuv** (request & WS handlers).

Usage:

```
Beakr$appDefinition()
```

Method `new()`**:** Initialize the app: create a 'Router' and assign a random 'name' if missing.

Usage:

```
Beakr$new()
```

Method `start()`**:** Start the HTTP server via **httpuv**.

Usage:

```
Beakr$start(host, port, daemon)
```

Arguments:

`host` Hostname or IP to bind.

`port` Integer port to listen on.

`daemon` If 'TRUE', run in background with 'httpuv::startServer()'; otherwise run foreground with 'httpuv::runServer()'.

Method `print()`**:** Print a one-line summary (name, state, host, port, #middlewares).

Usage:

```
Beakr$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

[Router], [Middleware], [httpuv::startServer], [httpuv::runServer]

`cors` *Allow Cross-Origin-Requests*

Description

Allow Cross-Origin Resource Sharing headers as described in MDN Web Docs. Cross-origin resource sharing is a mechanism that allows restricted resources on a web page to be requested from another domain(origin) outside the domain from which the first resource was served.

Usage

```
cors(
  beakr,
  path = NULL,
  methods = c("GET", "POST", "PUT", "DELETE", "OPTIONS", "PATCH"),
  origin = "*",
  credentials = NULL,
  headers = NULL,
  maxAge = NULL,
  expose = NULL
)
```

Arguments

beakr	Beakr instance object.
path	String representing a path for which to specify a CORS policy. Default NULL applies a single policy for all URL routes.
methods	A vector of the request methods to allow. i.e Access-Control-Allow-Methods parameter, e.g GET, POST.
origin	A vector of the request origin(s) for which resource sharing is enabled. i.e Access-Control-Allow-Origin response header parameter.
credentials	A boolean to enable/disable credentialed requests. i.e Access-Control-Allow-Credentials response header parameter.
headers	A vector of the allowed headers. i.e Access-Control-Allow-Headers response header parameter.
maxAge	The max age, in seconds. i.e Access-Control-Max-Age response header parameter.
expose	The headers to expose. i.e Access-Control-Expose-Headers response header parameter.

Value

A Beakr instance with CORS enabled

Note

You can verify that CORS is enabled by using the Chrome browser and opening up the Developer Tools. The "Network" tab allows you to inspect response headers and see where the Cross-Origin policy is specified.

If you run the example in the console, be sure to stopServer(bekar) when you are done.

See Also

Request, Response, Error

Examples

```
library(beakr)

# Create an new beakr instance
beakr <- newBeakr()

# beakr pipeline
beakr %>%

  # Enable CORS
  cors() %>%

  # Respond to GET requests at the "/hi" route
  httpGET(path = "/hi", function(req, res, err) {
    print("Hello, World!")
  }) %>%

  # Respond to GET requests at the "/bye" route
  httpGET(path = "/bye", function(req, res, err) {
    print("Farewell, my friends.")
  }) %>%

  # Start the server on port 25118
  listen(host = "127.0.0.1", port = 25118, daemon = TRUE)

# ------------------------------------------------------------
# POINT YOUR BROWSER AT:
# * http://127.0.0.1:25118/hi
# * http://127.0.0.1:25118/bye
#
# THEN, STOP THE SERVER WITH stopServer(beakr)
# ------------------------------------------------------------

# Stop the beakr instance server
stopServer(beakr)
```

`decorate` *Decorate a function for use in a web service*

Description

The `decorate()` function can be used to prepare a function for easy use in a beakr pipeline.

Decorating a function associates the specified function and its parameters with `req`, `res`, and `err` objects and assigns a content-type to the response object. This prepares a standard R function to be used in `Beakr` instances and accept requests.

Usage

```
decorate(FUN, content_type = "text/html", strict = FALSE)
```

Arguments

FUN	Function to decorate.
content_type	HTTP "content-type" of the function output. (*e.g.* "text/plain", "text/html" or other mime type)
strict	Boolean, requiring strict parameter matching.

Value

A *decorated* middleware function.

Examples

```
library(beakr)

# Create an new Beakr instance
beakr <- newBeakr()

# Create simple hello and goodbye function
hello <- function(name) { paste0("Hello, ", name, "!") }
goodbye <- function(text = "Adios") { paste0(text, ", dear friend.") }

# Create a web service from these functions
beakr %>%

  httpGET(path = "/hello", decorate(hello)) %>%

  httpGET(path = "/goodbye", decorate(goodbye)) %>%

  handleErrors() %>%

  listen(host = '127.0.0.1', port = 25118, daemon = TRUE)

# ------------------------------------------------------------
# POINT YOUR BROWSER AT:
# * http://127.0.0.1:25118/hello?name=Honeydew
# * http://127.0.0.1:25118/goodbye?text=Sionara
#
# THEN, STOP THE SERVER WITH stopServer(beakr)
# ------------------------------------------------------------

# Stop the beakr instance server
stopServer(beakr)
```

Error *Error Class*

Description

An ‘Error‘ object tracks and reports errors that occur during request handling or middleware execution in a [‘Router‘]. Errors are collected in a list, and the ‘occurred‘ active binding indicates whether any errors have been set.

Format

An [‘R6::R6Class‘] generator for ‘Error‘ objects.

Public fields

`errors` Character vector of recorded error messages.

Active bindings

`occurred` Logical; ‘TRUE‘ if any errors have been recorded.

Methods

Public methods:

- `Error$set()`
- `Error$clone()`

Method `set()`**:** Append an error message to ‘errors‘.

Usage:

```
Error$set(err)
```

Arguments:

`err` Error message (coerced to character).

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

[Middleware], [Router], [handleErrors]

handleErrors *Error-handling middleware*

Description

This default error handler should be added at the end of the beakr pipeline, right before `listen()`. Errors generated by any previous step will be returned within a JSON wrapper.

Usage

```
handleErrors(beakr = NULL, FUN = jsonError)
```

Arguments

beakr	Beakr instance
FUN	a function to handle the error response

Value

A `Beakr` instance with added middleware.

Note

If you run the example in the console, be sure to `stopServer(bekar)` when you are done.

Examples

```
library(beakr)

# Create an new beakr instance
beakr <- newBeakr()

# beakr pipeline
beakr %>%

  # Respond to GET requests at the "/hi" route
  httpGET(path = "/hi", function(req, res, err) {
    print("Hello, World!")
  }) %>%

  # Respond to GET requests at the "/bye" route
  httpGET(path = "/bye", function(req, res, err) {
    print("Farewell, my friends.")
  }) %>%

  handleErrors() %>%

  # Start the server on port 25118
  listen(host = "127.0.0.1", port = 25118, daemon = TRUE)
```

```
# ------------------------------------------------------------
# POINT YOUR BROWSER AT:
# * http://127.0.0.1:25118/NOT_A_ROUTE
#
# THEN, STOP THE SERVER WITH stopServer(beakr)
# ------------------------------------------------------------

# Stop the beakr instance server
stopServer(beakr)
```

`httpDELETE` *DELETE-binding middleware*

Description

Routes HTTP DELETE requests to the specified path with the specified callback functions or middleware.

Usage

```
httpDELETE(beakr, path = NULL, FUN = NULL)
```

Arguments

`beakr`	`Beakr` instance or `NULL`.
`path`	String representing a path for which the middleware function is invoked.
`FUN`	Middleware function to be invoked.

Value

A `Beakr` instance with added middleware.

Examples

```
## Not run:
library(beakr)

# Create an new Beakr instance
beakr <- newBeakr()

# Create a simple beakr pipeline
beakr %>%
  httpDELETE("/", function(req, res, err) {
    return("Successful DELETE request!\n")
  }) %>%
  listen(host = '127.0.0.1', port = 25118, daemon = TRUE)
```

```
# ------------------------------------------------------------
# IN A TERMINAL:
#  curl -X DELETE http://127.0.0.1:25118/
# > Successful DELETE request!
# ------------------------------------------------------------

# Stop the beakr instance server
stopServer(beakr)

## End(Not run)
```

`httpGET` *GET-binding middleware*

Description

Routes HTTP GET requests to the specified path with the specified callback functions or middleware.

Usage

```
httpGET(beakr, path = NULL, FUN = NULL)
```

Arguments

`beakr`	`Beakr` instance or `NULL`.
`path`	String representing a path for which the middleware function is invoked.
`FUN`	Middleware function to be invoked.

Value

A `Beakr` instance with added middleware.

Examples

```
## Not run:
library(beakr)

# Create an new Beakr instance
beakr <- newBeakr()

# Create a simple beakr pipeline
beakr %>%
  httpGET("/", function(req, res, err) {
    return("Successful GET request!\n")
  }) %>%
  listen(host = '127.0.0.1', port = 25118, daemon = TRUE)

# ------------------------------------------------------------
```

```
# IN A TERMINAL:
#  curl -X GET http://127.0.0.1:25118/
# > Successful GET request!
# ------------------------------------------------------------

# Stop the beakr instance server
stopServer(beakr)

## End(Not run)
```

`httpPOST` *POST-binding middleware*

Description

Routes HTTP POST requests to the specified path with the specified callback functions or middleware.

Usage

```
httpPOST(beakr, path = NULL, FUN = NULL)
```

Arguments

`beakr`	`Beakr` instance or `NULL`.
`path`	String representing a path for which the middleware function is invoked.
`FUN`	Middleware function to be invoked.

Value

A `Beakr` instance with added middleware.

Examples

```
## Not run:
library(beakr)

# Create an new Beakr instance
beakr <- newBeakr()

# Create a simple beakr pipeline
beakr %>%
  httpPOST("/", function(req, res, err) {
    return("Successful POST request!\n")
  }) %>%
  listen(host = '127.0.0.1', port = 25118, daemon = TRUE)

# ------------------------------------------------------------
# IN A TERMINAL:
```

```
#  curl -X POST http://127.0.0.1:25118/
# > Successful POST request!
# ------------------------------------------------------------

# Stop the beakr instance server
stopServer(beakr)

## End(Not run)
```

`httpPUT` *PUT-binding middleware*

Description

Routes HTTP PUT requests to the specified path with the specified callback functions or middleware.

Usage

```
httpPUT(beakr, path = NULL, FUN = NULL)
```

Arguments

`beakr`	`Beakr` instance or `NULL`.
`path`	String representing a path for which the middleware function is invoked.
`FUN`	Middleware function to be invoked.

Value

A `Beakr` instance with added middleware.

Examples

```
## Not run:
library(beakr)

# Create an new Beakr instance
beakr <- newBeakr()

# Create a simple beakr pipeline
beakr %>%
  httpPUT("/", function(req, res, err) {
    return("Successful PUT request!\n")
  }) %>%
  listen(host = '127.0.0.1', port = 25118, daemon = TRUE)

# ------------------------------------------------------------
# IN A TERMINAL:
#  curl -X PUT http://127.0.0.1:25118/
```

```
# > Successful PUT request!
# ------------------------------------------------------------

# Stop the beakr instance server
stopServer(beakr)

## End(Not run)
```

`jsonError` *JSON error function*

Description

This function is used to add a JSON error response to the `res` object. It is called by the `handleErrors()` utility function.

Usage

```
jsonError(req, res, err)
```

Arguments

`req`	`Request` object.
`res`	`Response` object.
`err`	`Error` Error object.

Value

The incoming `res` object is modified.

See Also

Request, Response, Error

`listen` *Listen for connections on a Beakr instance*

Description

Binds and listens for connections at the specified host and port.

Usage

```
listen(
  beakr = NULL,
  host = "127.0.0.1",
  port = 25118,
  daemon = FALSE,
  verbose = FALSE
)
```

Arguments

beakr	Beakr instance.
host	String that is a valid IPv4 or IPv6 address to listen on. Defaults to the local host ("127.0.0.1").
port	Number or integer that indicates the port to listen on. Default is a port opened on 25118.
daemon	Logical specifying whether the server should be run in the background.
verbose	Logical specifying whether to print out details of the Beakr instance now running. This should only be used when running a beaker app interactively, not in production.

Details

`listen()` binds the specified host and port and listens for connections on a thread. The thread handles incoming requests. when it receives an HTTP request, it will schedule a call to the user-defined middleware and handle the request.

If `daemon = TRUE`, `listen()` binds the specified port and listens for connections on a thread running in the background.

See the **httpuv** package for more details.

Value

A Beakr instance with an active server.

Note

The default port number 25118 was generated using:

```
> match(c("b","e","a","k","r"), letters) %% 10
[1] 2 5 1 1 8
```

Examples

```
library(beakr)

# Create an new Beakr instance
beakr <- newBeakr()
```

```
# beakr pipeline
beakr %>%

  httpGET("/", function(req, res, err) {
    return("Successful GET request!\n")
  }) %>%

  listen(daemon = TRUE)     # run in the background

# Stop the server
stopServer(beakr)
```

`Listener` *Listener Class*

Description

A 'Listener' object represents an event handler within a ['Router']. Each listener pairs an 'event' type (e.g., '"start"', '"error"', '"finish"') with a function 'FUN' to execute when that event is triggered.

Format

An ['R6::R6Class'] generator for 'Listener' objects.

Public fields

`FUN` Handler function to execute when 'event' is triggered.

`event` Event name (e.g., '"start"', '"error"', '"finish"').

Methods

Public methods:

- `Listener$new()`
- `Listener$clone()`

Method `new()`**:** Construct a listener by setting its 'event' and handler 'FUN'.

Usage:

```
Listener$new(event, FUN, ...)
```

Arguments:

`event` Event name string.

`FUN` Function to call when the event occurs.

`...` Ignored; accepted for flexibility.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

[Router], [Error]

`listServers` *List all servers*

Description

Lists all `Beakr` servers currently running (and any other servers created with the **httpuv** package). This function is included to encourage experimentation so that users who create multiple `Beakr` instances can quickly find and stop them all.

See `httpuv::listServers` for details.

Usage

```
listServers()
```

Value

None

Examples

```
library(beakr)

beakr1 <- newBeakr()
beakr2 <- newBeakr()
beakr1 %>% listen(daemon = TRUE, port = 1234, verbose = TRUE)
beakr2 %>% listen(daemon = TRUE, port = 4321, verbose = TRUE)
length(listServers())
stopAllServers()
length(listServers())
```

`Middleware` *Middleware Class*

Description

A ‘Middleware‘ object wraps a handler function with associated metadata (‘path‘, ‘method‘, ‘protocol‘). Middleware functions have access to the request (‘req‘), response (‘res‘), and error (‘err‘) objects during the request–response cycle via the [‘Router‘].

Format

An [‘R6::R6Class‘] generator for ‘Middleware‘ objects.

Public fields

`path` Path this middleware matches, or ‘NULL‘ for all paths.

`FUN` Handler function executed when matched.

`method` HTTP method to match (e.g., ‘"GET"‘), or ‘NULL‘ for any.

`protocol` Protocol string: ‘"http"‘ or ‘"websocket"‘.

Methods

Public methods:

- `Middleware$new()`
- `Middleware$clone()`

Method `new()`**:** Initialize middleware with handler, path, method, and protocol selection.

Usage:

```
Middleware$new(FUN, path, method, websocket)
```

Arguments:

`FUN` Handler function (e.g., ‘(req, res, err)‘ for HTTP).

`path` Route path to match, or ‘NULL‘ for all.

`method` HTTP method to match, or ‘NULL‘ for any.

`websocket` If ‘TRUE‘, set ‘protocol = "websocket"‘; otherwise ‘"http"‘.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

[Router], [Request], [Response], [Error]

`newBeakr` *Create a new Beakr instance*

Description

Create a `Beakr` instance by calling the top-level `newBeakr()` function. If `name` is not supplied, a random name will be assigned.

This `Beakr` instance will then begin a pipeline of separate middleware steps for routing, serving files and handling errors. The pipeline will end with the `listen()` function.

Usage

```
newBeakr(name = NULL)
```

Arguments

name	Optional name assigned to the Beakr instance.

Value

A new and empty Beakr instance.

Examples

```
library(beakr)

# Create an new beakr instance
beakr <- newBeakr()

# beakr pipeline of hanldlers
beakr %>%

  httpGET(path = "/route_A", function(res, req, err) {
    print("This is route 'A'.")
  }) %>%

  httpGET(path = "/route_B", function(res, req, err) {
    print("This is route 'B'.")
  }) %>%

  handleErrors() %>%

  listen(host = '127.0.0.1', port = 25118, daemon = TRUE)

# ------------------------------------------------------------
# POINT YOUR BROWSER AT:
# * http://127.0.0.1:25118/route_A
# * http://127.0.0.1:25118/route_B
#
# THEN, STOP THE SERVER WITH stopServer(beakr)
# ------------------------------------------------------------

# Stop the beakr instance server
stopServer(beakr)
```

Request *Request Class*

Description

A 'Request' object represents an incoming HTTP request. It stores query string, parameters, body, headers, method, and protocol information. By convention, the request object is named 'req' (with the corresponding response named 'res').

Format

An [‘R6::R6Class‘] generator for ‘Request‘ objects.

Public fields

`parameters` Named list of route parameters.

`headers` Named list of request headers.

`path` The request path.

`method` HTTP method (e.g., ‘"GET"‘, ‘"POST"‘).

`raw` The raw request object as received.

`type` Content type (e.g., ‘"text/html"‘, ‘"application/json"‘).

`body` Raw request body as a single string.

`protocol` Protocol string (‘"http"‘ or ‘"websocket"‘).

Methods

Public methods:

- `Request$attach()`
- `Request$getHeader()`
- `Request$setHeader()`
- `Request$addParameters()`
- `Request$new()`
- `Request$clone()`

Method `attach()`**:** Attach a parameter key-value to ‘parameters‘.

Usage:

```
Request$attach(key, value)
```

Arguments:

`key` Parameter name.

`value` Parameter value.

Method `getHeader()`**:** Get the value of a request header.

Usage:

```
Request$getHeader(key)
```

Arguments:

`key` Header name (case-insensitive).

Method `setHeader()`**:** Set or overwrite a request header.

Usage:

```
Request$setHeader(key, value)
```

Arguments:

`key` Header name.

`value` Header value.

Method `addParameters()`**:** Merge a named list of parameters into ‘parameters‘.

Usage:

```
Request$addParameters(named_list)
```

Arguments:

`named_list` Named list of key-value pairs.

Method `new()`**:** Parse fields from the raw request and populate the object.

Usage:

```
Request$new(req)
```

Arguments:

`req` Raw request object.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

[Response]

`Response` *Response Class*

Description

A ‘Response‘ object represents the HTTP response that a ‘Beakr‘ app sends when handling a request. By convention, the response object is named ‘res‘ (with the corresponding request named ‘req‘).

Format

An [‘R6::R6Class‘] generator for ‘Response‘ objects.

Public fields

`headers` A named list of HTTP response headers. Defaults to ‘list("Content-Type" = "text/html")‘.

`status` An integer HTTP status code. Defaults to ‘200L‘.

`body` The response body. May be ‘NULL‘, character, raw, JSON, or base64.

Methods

Public methods:

- Response$setHeader()
- Response$setContentType()
- Response$setStatus()
- Response$setBody()
- Response$redirect()
- Response$json()
- Response$text()
- Response$structured()
- Response$plot()
- Response$clone()

Method `setHeader()`**:** Set a header key-value pair (e.g., '"Content-Type" = "text/html"').

Usage:

```
Response$setHeader(key, value)
```

Arguments:

`key` Header name.

`value` Header value.

Method `setContentType()`**:** Set the response 'Content-Type'.

Usage:

```
Response$setContentType(type)
```

Arguments:

`type` MIME type string.

Method `setStatus()`**:** Set the HTTP status code.

Usage:

```
Response$setStatus(status)
```

Arguments:

`status` Integer HTTP status code.

Method `setBody()`**:** Set the response body, respecting the current 'Content-Type'.

Usage:

```
Response$setBody(body)
```

Arguments:

`body` Body content, type depends on 'Content-Type'.

Method `redirect()`**:** Redirect the client by setting status 302 and 'Location' header.

Usage:

```
Response$redirect(url)
```

Arguments:

`url` The URL to redirect to.

Method `json()`**:** Convert ‘txt‘ to JSON and set content type to ‘"application/json"‘.

Usage:

```
Response$json(txt, auto_unbox = TRUE)
```

Arguments:

`txt` Content to convert to JSON.

`auto_unbox` Logical; whether to simplify length-1 vectors.

Method `text()`**:** Set the response body as plain text and content type ‘"text/html"‘.

Usage:

```
Response$text(txt)
```

Arguments:

`txt` Content to include as plain text.

Method `structured()`**:** Return a structured response depending on ‘protocol‘.

Usage:

```
Response$structured(protocol)
```

Arguments:

`protocol` ‘"http"‘ or ‘"websocket"‘.

Method `plot()`**:** Render a plot to PNG (optionally base64-encode) and set as response body.

Usage:

```
Response$plot(plot_object, base64 = TRUE, ...)
```

Arguments:

`plot_object` A plot object to render.

`base64` Logical; if ‘TRUE‘, encode image as base64.

`...` Passed to [graphics::png()].

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

[Router], [Request], [Error]

`Router` *Router Class*

Description

'Router' coordinates HTTP/WebSocket routing and middleware execution. After instantiation, you can register middleware and event listeners; then call '$invoke()' to run the request/response cycle.

Format

An ['R6::R6Class'] generator for 'Router' objects.

Public fields

`middleware` List of middleware entries.

`listeners` List of listeners (event handlers).

Methods

Public methods:

- `Router$addMiddleware()`
- `Router$addListener()`
- `Router$processEvent()`
- `Router$invoke()`
- `Router$clone()`

Method `addMiddleware()`**:** Append middleware entry/entries to 'middleware'.

Usage:

```
Router$addMiddleware(middleware)
```

Arguments:

`middleware` A middleware object/function or list of them.

Method `addListener()`**:** Append a listener to 'listeners'.

Usage:

```
Router$addListener(listener)
```

Arguments:

`listener` A listener object with fields like 'event' and 'FUN'.

Method `processEvent()`**:** Dispatch an event to all matching listeners.

Usage:

```
Router$processEvent(event, ...)
```

Arguments:

`event` Event name (e.g., '"start"', '"error"', '"finish"').

... Additional arguments forwarded to each listener 'FUN'.

Method `invoke()`**:** Run the routing/middleware pipeline and return a structured response.

Usage:

```
Router$invoke(req, websocket_msg = NULL, websocket_binary = NULL)
```

Arguments:

`req` Raw request object or 'Request' instance.

`websocket_msg` Optional WebSocket text message.

`websocket_binary` Optional WebSocket binary payload (raw).

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.

See Also

[Response], [Request], [Error]

`serveStaticFiles` *File-serving middleware*

Description

Binds to GET requests that aren't handled by specified paths. The result is to return files that are found on the host machine at the requested path. Binary file types like `.png`, `.gif` or `.pdf` are returned as raw bytes. All others are returned as characters.

Mime types are guessed using the **mime** package. The `rawTypesPattern` parameter is used to match mime types that should be returned as raw bytes.

Usage

```
serveStaticFiles(
  beakr = NULL,
  urlPath = NULL,
  rootPath = getwd(),
  rawTypesPattern = "image|json|octet|pdf|video",
  verbose = FALSE
)
```

Arguments

beakr	Beakr instance or NULL.
urlPath	String representing the URL directory underneath which static file paths will appear.
rootPath	String representing the absolute path used as the root directory when searching for files on host machine. Defaults to the directory in which the script is running.
rawTypesPattern	String pattern identifying mime types to be returned as raw bytes.
verbose	Boolean to show a verbose static file information.

Details

All files to be served in this manner must exist underneath the host machine directory specified with rootPath. The directory structure underneath rootPath will be mapped onto URLs underneath urlPath. This helps when deploying web services at preordained URLs.

The example below presents files underneath host machine directory hostDir/ to be accessed at URLS under test/.

Value

A Beakr instance with added middleware.

Note

If you run the example in the console, be sure to stopServer(bekar) when you are done.

Examples

```
library(beakr)

# Create a .txt file in temp directory
hostDir <- tempdir()
file <- paste0(hostDir, "/my_file.txt")
cat("I am a text file.", file = file)

# Create an new beakr instance
beakr <- newBeakr()

# beakr pipeline
beakr %>%

  # Respond to GET requests at the "/hi" route
  httpGET(path = "/hi", function(req, res, err) {
    print("Hello, World!")
  }) %>%

  # Respond to GET requests at the "/bye" route
  httpGET(path = "/bye", function(req, res, err) {
    print("Farewell, my friends.")
```

```
 }) %>%

 # Host the directory of static files
 serveStaticFiles("/test", hostDir, verbose = TRUE) %>%

 # Start the server on port 25118
 listen(host = "127.0.0.1", port = 25118, daemon = TRUE)

# ------------------------------------------------------------
# POINT YOUR BROWSER AT:
# * http://127.0.0.1:25118/test/my_file.txt
#
# THEN, STOP THE SERVER WITH stopServer(beakr)
# ------------------------------------------------------------

# Stop the beakr instance server
stopServer(beakr)
```

`stopAllServers` *Stop all servers*

Description

Stops all `Beakr` servers currently running (and any other servers created with the **httpuv** package). This function is included to encourage experimentation so that users who create multiple `Beakr` instances can quickly find and stop them all.

See `httpuv::stopAllServers` for details.

Usage

```
stopAllServers()
```

Value

None

Examples

```
library(beakr)

beakr1 <- newBeakr()
beakr2 <- newBeakr()
beakr1 %>% listen(daemon = TRUE, port = 1234, verbose = TRUE)
beakr2 %>% listen(daemon = TRUE, port = 4321, verbose = TRUE)
length(listServers())
stopAllServers()
length(listServers())
```

stopServer *Stop a beakr instance server*

Description

Stops the server associated with a Beakr instance, closing all open connections and unbinding the port.

Usage

```
stopServer(beakr = NULL, verbose = FALSE)
```

Arguments

beakr	Beakr instance.
verbose	Logical specifying whether to print out details of the Beakr instance just stopped.

Value

None

Examples

```
library(beakr)

beakr <- newBeakr()

# beakr pipeline
beakr %>%

  handleErrors() %>%

  listen(daemon = TRUE, verbose = TRUE)

stopServer(beakr, verbose = TRUE)
```

Index