Package ‘Rook’

November 7, 2022

Title HTTP Web Server for R

Version 1.2

Date 2022-11-05

Description An HTTP web server for R with a documented API to interface between R and the server. The documentation contains the Rook specification and details for building and running Rook applications. To get started, be sure and read the 'Rook' help file first.

Encoding UTF-8

Depends R (>= 2.13.0)

Imports utils, tools, methods, brew

License GPL-2

LazyLoad yes

URL https://github.com/evanbiederstedt/rook

BugReports https://github.com/evanbiederstedt/rook/issues

Author Jeffrey Horner [aut], Evan Biederstedt [aut, cre]

Maintainer Evan Biederstedt <evan.biederstedt@gmail.com>

NeedsCompilation yes

Repository CRAN

Date/Publication 2022-11-07 08:50:19 UTC

R topics documented:

Rook-package *Rook: A web server interface and package for R*

Description

This help page defines the Rook specification. It borrows heavily from Ruby's Rack project: https://github.com/rack/rack.

After reading this document, read the Rhttpd help file as it will get you familiar with installing and running Rook applications. Then explore the example applications located in:

`system.file('exampleApps',package='Rook')`.

Rook applications

A Rook application is an R reference class object that implements a 'call' method or an R closure that takes exactly one argument, an environment, and returns a list with three named elements: `'status'`, `'headers'`, and `'body'`.

Hello World

Here is a basic Rook application as a closure that implements 'hello world':

```
function(env){
    body = paste('<h1>Hello World! This is Rook',env$rook.version,'.</h1>')
    list(
        status = 200L,
        headers = list(
            'Content-Type' = 'text/html'
        ),
        body = body
    )
}
```

And the equivalent reference class example:

```
setRefClass(
  'HelloWorld',
  methods = list(
    call = function(env){
      list(
        status = 200L,
        headers = list(
         'Content-Type' = 'text/html'
        ),
        body = paste('<h1>Hello World! This is Rook',env$rook.version,'.</h1>')
      )
    }
  )
)
```

The Environment

The environment argument is a true R environment object which the application is free to modify. It is required to contain the following variables:

REQUEST_METHOD The HTTP request method, such as "GET" or "POST". This cannot ever be an empty string, and so is always required.

SCRIPT_NAME The initial portion of the request URL's "path" that corresponds to the application object, so that the application knows its virtual "location". This may be an empty string, if the application corresponds to the "root" of the server.

PATH_INFO The remainder of the request URL's "path", designating the virtual "location" of the request's target within the application. This may be an empty string, if the request URL targets the application root and does not have a trailing slash. This value may be percent-encoded when I originating from a URL.

QUERY_STRING The portion of the request URL that follows the ?, if any. May be empty, but is always required!

SERVER_NAME, SERVER_PORT When combined with SCRIPT_NAME and PATH_INFO, these variables can be used to complete the URL. Note however that HTTP_HOST, if present, should be used in preference to SERVER_NAME for reconstructing the request URL. SERVER_NAME and SERVER_PORT can never be empty strings, and so are always required.

HTTP_ Variables Variables corresponding to the client-supplied HTTP request headers (i.e., variables whose names begin with HTTP_). The presence or absence of these variables should correspond with the presence or absence of the appropriate HTTP header in the request.

In addition, the environment must include the following Rook-specific variables:

rook.version This version of Rook.

rook.url_scheme 'http' or 'https', depending on the request URL.

rook.input See "The Input Stream" section.

rook.errors See "The Error Stream" section.

The Input Stream

The rook.input variable must contain an object created from a reference class that implements `read_lines()`, `read()`, and `rewind()`:

`read_lines(l=-1L)`: takes one argument, the number of lines to read. Includes partial ending line.

`read(l=-1L)`: takes one argument, the number of bytes to read. Returns a raw vector.

`rewind()`: Rewinds the input stream back to the beginning.

The Error Stream

The rook.error variable must contain an object created from a reference class that implements `flush()` and `cat()`:

`flush()`: called with no arguments and makes the error stream immediately appear.

`cat(...,sep=" ",fill=FALSE,labels=NULL)`: called with the same arguments as R's "`cat`" without the `file` and append `argument`.

The Response

Rook applications return a list with three named elements: `'status'`, `'headers'`, and `'body'`.

`'status'`: An HTTP status value as integer and must be greater than or equal to 100.

`'headers'`: A named list that contains only character values corresponding to valid HTTP headers.

`'body'`: Either a character or raw vector. If the character vector is named with value `'file'` then value of the vector is interpreted as the location of a file.

Author(s)

Jeffrey Horner <jeffrey.horner@gmail.com>

`App-class` *Class* `App`

Description

Abstract class from which `Middleware` and `Builder` inherit. Provides the app field.

`App` can also be used to instantiate reference classed applications wrapped around a function. See `Middleware` for an example.

Fields

`app`: A Rook application.

Methods

new(app=NULL): Creates a new App object. app is any Rook aware R object.

See Also

is_rookable, Builder, and Middleware.

Brewery-class *Class* Brewery

Description

A Middleware class for mapping URLs to a directory of files that are subsequently passed to brew. When a file is brewed, the two variables req (an object of class Request) and res (an object of class Response) are available for use.

Methods

new(url,root,...): url is a character string or regexp on which to match, root is the name of the directory where brew files reside. Named arguments can be passed in via ... and will be available within the scope of each brewed file.

See Also

Rhttpd, Builder, Redirect, and brew.

Examples

```
#
# This application runs any file found in tempdir() through brew.
#
s <- Rhttpd$new()
## Not run:
s$start(quiet=TRUE)

## End(Not run)
cat("<h1>Random Number: <%=rnorm(1)%></h1>",
    file=file.path(tempdir(),"index.html"))
s$add(name="random",
    app=Builder$new(
        Brewery$new(url="/",root=tempdir()),
        Redirect$new("/index.html")
    )
)
## Not run:
s$browse('random') # Opens a browser window to the app.

## End(Not run)
```

```
file.remove(file.path(tempdir(),"index.html"))
s$remove(all=TRUE)
rm(s)
```

Builder-class *Class* Builder

Description

A convenience object for combining various `Middleware` with a default application to create a more complex Rook application.

Methods

`new(...)`: Arguments can be any `Middleware` object while the last argument in the list must be a valid `Rook` application. That is, it will handle the incoming request without deferring to another application.

See Also

`Rhttpd`, `Static`, `Brewery`, and `Redirect`.

Examples

```
# The following is the Hmisc example. Explore the folder
# system.file('exampleApps/Hmisc',package='Rook') for more information.
s <- Rhttpd$new()
## Not run:
library(Hmisc)
dir.create(file.path(tempdir(),'plots'),showWarnings=FALSE)
s$add( name="Hmisc",
    app=Builder$new(
        Static$new(
            urls = c('/css','/images','/javascript'),
            root = system.file('exampleApps/Hmisc',package='Rook')
        ),
        Static$new(urls='/plots',root=tempdir()),
        Brewery$new(
            url='/brew',
            root= system.file('exampleApps/Hmisc',package='Rook'),
            imagepath=file.path(tempdir(),'plots'),
            imageurl='../plots/'
        ),
        Redirect$new('/brew/useR2007.rhtml')
    )
)
s$start(quiet=TRUE)
s$browse('Hmisc') # Opens a browser window to the application.
s$remove(all=TRUE)
```

```
s$stop()

## End(Not run)
```

File-class *Class* File

Description

A Rook application that serves static files from a root directory, according to the path info of the Rook request.

Methods

`new(root)`: `root` is the name of the directory from where to serve files.

See Also

Rhttpd.

Examples

```
# This example serves all your files in /etc (on UNIX and Mac only).
#
# Note that when you open the application, you will see the word
# 'Forbidden'. "File" doesn't serve directories, so you must amend the
# url in the location bar with the file you want to view. Try adding /passwd.

s <- Rhttpd$new()
## Not run:
s$start(quiet=TRUE)

## End(Not run)
s$add(name="etc",app=File$new('/etc'))
## Not run:
s$browse('etc') # Opens a browser window to the app.

## End(Not run)
s$remove(all=TRUE)
rm(s)
```

is_rookable *Test for Rookable applications*

Description

A convenience function for testing whether or not objects are either a function or reference class as defined by the Rook specification for applications.

Usage

```
is_rookable(app)
```

Arguments

app	Any R object.

Value

Logical determining whether or not argument is Rookable. Not vectorized.

See Also

`Rook`.

Middleware-class *Class* `Middleware`

Description

An abstract class for building Rook Middleware applications. `Middleware` applications either handle the incoming web request or hand off the request to the Rook app defined in the field of the same name.

Methods

`set_app(app)`: `app` is a `Rook` application that will handle the request if this Middleware app does not.

See Also

The following classes implement Middleware: `Brewery` and `Static`.

Examples

```
# Middleware applications are typically instantiated in the argument list of
# Builder$new(), but here is stand-alone example.
#
# Once your browser loads the app, you will see something like this in
# your location bar: http://127.0.0.1:28649/custom/middle. Add '/foo'
# onto the end of that and reload.

setRefClass(
    'FooBar',
    contains = 'Middleware',
    methods = list(
        initialize = function(...){
            # app to defer to.
            callSuper(app=App$new(function(env){
                res <- Response$new()
                res$write("<h1>I'm the deferred app.</h1>")
                res$finish()
            }))
        },
        call = function(env){
            req <- Request$new(env)
            res <- Response$new()
            if (length(grep('foo',req$path_info()))){
                res$write("<h1>I'm the middleware app.</h1>")
                return(res$finish())
            } else {
                app$call(env)
            }
        }
    )
)
s <- Rhttpd$new()
## Not run:
s$start(quiet=TRUE)

## End(Not run)
s$add(name="middle",app=getRefClass('FooBar')$new())
## Not run:
s$browse('middle') # Opens a browser window to the app.

## End(Not run)
s$remove(all=TRUE)
rm(s)
```

`Mime-class` *Class* `Mime` *and object* `Mime`

Description

A convenience object for determining the MIME type of a file name.

Methods

`file_extname(fname=NULL)`: Returns the file extensions for the given file.

`mime_type(ext=NULL, fallback='application/octet-stream')`: Returns the MIME type given the file extension. Be sure to include the dot character in `ext`. If no match is found, then the fallback MIME type is returned.

Examples

```
Mime$file_extname('foo.png')
Mime$mime_type('.png')
```

Multipart-class — *Class* `Multipart` *and object* `Multipart`

Description

A convenience object for parsing multipart/form-data POST payloads.

Methods

`parse(env)`: Returns parsed POST payload as a named list. `env` is an environment created by `Rhttpd` and conforms to the Rook specification.

See Also

`Rhttpd`, `Request`, and `Response`.

Examples

```
s <- Rhttpd$new()
## Not run:
s$start(quiet=TRUE)

## End(Not run)
s$add(name="multi",
    app=function(env){
        req <- Request$new(env)
        res <- Response$new()
        res$write('<form enctype="multipart/form-data" method=POST>')
        res$write('Upload a file: <input type=file name=fileUpload>')
        res$write('<input type=submit></form><br>')
        post <- Multipart$parse(env)
        if (length(post)){
            poststr <- paste(capture.output(str(post),file=NULL),collapse='\n')
            res$write(c('<pre>',poststr,'</pre>'))
        }
        res$finish()
    }
)
```

```
## Not run:
s$browse('multi') # Opens a browser window to the app.

## End(Not run)
s$remove(all=TRUE)
rm(s)
```

Redirect-class   *Class* Redirect

Description

A Rook application whose only role is to return an HTTP redirect header to the given url.

Methods

new(url): Returns a Rook object. url is a character string whose value is a full or relative url to which the browser is redirected.

See Also

See Brewery for an example.

Request-class   *Class* Request

Description

A convenience class for working with a Rook environment. Be sure to see the example at the end of this help file.

Methods

parseable_data(): Returns a boolean value determining if the POST payload is parseable.

url(): Returns url as a character string containing the scheme, host, port, and possibly the GET query string if supplied.

request_method(): Returns the HTTP method as a character string, e.g. 'GET', 'POST', etc.

GET(): Returns a named list containing the variables parsed from the query string.

post(): Returns TRUE if the current request method is 'POST', FALSE otherwise.

new(env): Instantiates a new Request object for the given Rook environment.

media_type(): Returns the media type for the current request as a character string.

query_string(): Returns the unparsed query string.

fullpath(): Returns the same string as url() but without the scheme, host, and port.

`referer()` **or** `referrer()`: Returns the referring url.

`cookies()`: Returns any cookies in the request as a named list.

`content_charset()`: Returns the content charset as a character string.

`head()`: Returns TRUE if the HTTP method is 'HEAD', FALSE otherwise.

`accept_encoding()`: Returns the accept encoding header as a character string.

`content_length()`: Returns content length header value as a string.

`form_data()`: Returns TRUE if there's form data, e.g. POST data with the request, FALSE otherwise.

`xhr()`: Returns the x-requested-with header value as a character string.

`params()`: Returns the combination of `POST()` and `GET()` in one named list.

`media_type_params()`: Returns any media type parameters from the content type as a named list.

`user_agent()`: Returns the user-agent header value as a character string.

`put()`: Returns TRUE if the current request is a 'PUT'.

`get()`: Returns TRUE if the current request is a 'GET'.

`path()`: Returns a character string like `fullpath()` but without the query string.

`body()`: Returns the 'rook.input' object from the environment. See `RhttpdInputStream` for more information.

`port()`: Returns the server port as an integer.e

`host_with_port()`: Returns the host and port as a character string separated by ':'.

`scheme()`: Returns the scheme, e.g. 'http' or 'https', as a character string.

`ip()`: Returns the remote IP address as a character string.

`options()`: Returns TRUE if the current request is 'OPTIONS'.

`to_url(url, ...)`: Concatenates the script name with the `url` argument along with any named parameters passed via `...` .

`host()`: Returns the server host as a character string.

`POST()`: Returns a named list containing the variables parsed from the POST payload.

`trace()`: Returns TRUE if the current request is 'TRACE'.

`script_name(s=NULL)`: Returns the script name of the application, e.g. '/custom/multi'. Also, if `s` is not NULL, sets the script name to `s`.

`content_type()`: Returns the content-type header value as a character string.

`delete()`: Returns TRUE if the current request is 'DELETE'.

`path_info(s=NULL)`: Returns the portion of the url after the script name as a character string. If `s` is not NULL, sets the path info to `s`.

See Also

`Rhttpd` and `Response`.

Examples

```
#
# The following example prints out the result of each method.
#
ls_str <- function(s) paste(capture.output(str(s),file=NULL),collapse='\n')
s <- Rhttpd$new()
## Not run:
s$start(quiet=TRUE)

## End(Not run)
s$add(name="request",
    app=function(env){
        req <- Request$new(env)
        res <- Response$new()
        res$set_cookie('imacookie','42')
        action <- req$to_url('/foo',bar=1,baz='three')
        res$write('<form enctype="multipart/form-data" method=POST action="')
        res$write(action)
        res$write('">')
        res$write('Upload a file: <input type=file name=fileUpload>')
        res$write('<input type=submit></form><br><pre>')
        res$write(c('parseable_data: ',req$parseable_data(),'\n'))
        res$write(c('url: ',req$url(),'\n'))
        res$write(c('request_method: ',req$request_method(),'\n'))
        res$write(c('GET: ',ls_str(req$GET()),'\n'))
        res$write(c('post: ',req$post(),'\n'))
        res$write(c('media_type: ',req$media_type(),'\n'))
        res$write(c('query_string: ',req$query_string(),'\n'))
        res$write(c('fullpath: ',req$fullpath(),'\n'))
        res$write(c('referer: ',req$referer(),'\n'))
        res$write(c('cookies: ',ls_str(req$cookies()),'\n'))
        res$write(c('content_charset: ',req$content_charset(),'\n'))
        res$write(c('head: ',req$head(),'\n'))
        res$write(c('accept_encoding: ',req$accept_encoding(),'\n'))
        res$write(c('content_length: ',req$content_length(),'\n'))
        res$write(c('form_data: ',req$form_data(),'\n'))
        res$write(c('xhr: ',req$xhr(),'\n'))
        res$write(c('params: ',ls_str(req$params()),'\n'))
        res$write(c('media_type_params:\n',ls_str(req$media_type_params()),'\n'))
        res$write(c('user_agent: ',req$user_agent(),'\n'))
        res$write(c('put: ',req$put(),'\n'))
        res$write(c('get: ',req$get(),'\n'))
        res$write(c('path: ',req$path(),'\n'))
        res$write(c('body: ',ls_str(req$body()),'\n'))
        res$write(c('port: ',req$port(),'\n'))
        res$write(c('host_with_port: ',req$host_with_port(),'\n'))
        res$write(c('scheme: ',req$scheme(),'\n'))
        res$write(c('ip: ',req$ip(),'\n'))
        res$write(c('options: ',req$options(),'\n'))
        res$write(c('to_url: ',req$to_url('foo',bar=1,baz='two'),'\n'))
        res$write(c('host: ',req$host(),'\n'))
        res$write(c('POST: ',ls_str(req$POST()),'\n'))
```

```
        res$write(c('trace: ',req$trace(),'\n'))
        res$write(c('script_name: ',req$script_name(),'\n'))
        res$write(c('content_type: ',req$content_type(),'\n'))
        res$write(c('delete: ',req$delete(),'\n'))
        res$write(c('path_info: ',req$path_info(),'\n'))
        res$write(c('\nRac env: ',ls_str(as.list(env)),'\n'))
        res$finish()
    }
)
## Not run:
s$browse('request') # Opens a browser window to the app.

## End(Not run)
s$remove(all=TRUE)
rm(s)
```

Response-class *Class* Response

Description

A convenience class for creating Rook responses.

Methods

`header(key, value)`**:** Sets an HTTP header for the response. Both `key` and `value` must be character strings. If `value` is missing, then the `header` value is returned.

`redirect(target, status=302)`**:** Sets up an HTTP redirect to the `target` url.

`write(str)`**:** Takes a character vector and appends it to the response body.

`new(body='', status=200, headers=list())`**:** Create a new Response object. `body` is a character vector, `status` is an HTTP status value. `headers` is a named list.

`set_cookie(key, value)`**:** Sets an HTTP cookie for the response. Both `key` and `value` must be character strings.

`delete_cookie(key, value)`**:** Sends appropriate HTTP header to delete the associated cookie on the client. `key` and `value` must be character strings.

`finish()`**:** Returns the response according to the Rook specification.

See Also

`Rhttpd` and `Request`.

Examples

```
s <- Rhttpd$new()
## Not run:
s$start(quiet=TRUE)

## End(Not run)
s$add(name="response",
    app=function(env){
        req <- Request$new(env)
        res <- Response$new()
        res$write('hello')
        res$finish()
    }
)
## Not run:
s$browse('response') # Opens a browser window to the app.

## End(Not run)
s$remove(all=TRUE)
rm(s)
```

`Rhttpd-class` *Class* `Rhttpd`

Description

`Rhttpd` is a convenience class for installing and running Rook applications. It hides the details of starting and stopping the server and adding and removing Rook applications from the server.

Users starts by creating one `Rhttpd` object, then adding applications to it, and then starting the server (see the section "Examples" for a typical session). There are no restrictions on creating more than one server object, but know that it only manages the applications that are added to it and not others.

Applications can be added and removed regardless of whether or not the server is running. Stopping the server does not remove any applications. Adding an application with the same name as one already installed simply overwrites the one installed. If the server is started with no applications installed, it will install the application named `RookTestApp` located in:

`system.file('exampleApps/RookTestApp.R',package='Rook')`.

Also, see `browseURL` to learn how to get R to automatically launch your favorite web browser.

NOTE: This version of Rook can only listen on the loopback device.

Methods

`open(x)` **or** `browse(x)`**:** Calls `browseURL` on the installed Rook application designated by `x`. `x` is either an integer or a character string. See the output of `print()`.

`print()` **or** `show()`**:** Lists the installed Rook applications.

`remove(app,all=FALSE)`: Removes the application known to the server. app can be an RhttpdApp object previously added, the name of the application as a character string, or an index as a numeric or integer value. See the output of `print()`.

`full_url(i)`: Returns the absolute url to the application for the given index.

`start(listen='127.0.0.1', port=getOption('help.ports'), quiet=FALSE)`: Starts the server on the loopback device and port. `listen` is always character string. Note that if there are no applications added to the object prior to starting, then the RookTestApp located in `system.file('exampleApps/RookTestApp.R',package='Rook')` is automatically added.

`new()`: Create a new `Rhttpd` object.

`launch(...)`: Combines the steps of starting the server, creating an RhttpdApp object, adding it to the server, and opening the app in the browser. `...` argument is passed to `RhttpdApp$new()`.

`debug()`: Returns the integer value provided by `getOption('Rhttpd_debug')` or 0 if the option is NULL.

`stop()`: Stops the server.

`add(app=NULL,name=NULL)`: Adds a new Rook application to the server. app can be an RhttpdApp object or any Rook application. name is a character string and is ignored if app is an RhttpdApp object.

See Also

RhttpdApp

Examples

```
# Create an Rhttpd object and start the internal web server. Note that
# if there are no applications added, then the default RookTest app in
# system.file('exampleApps/RookTestApp.R',package='Rook') is automatically
# added.

s <- Rhttpd$new()
## Not run:
s$start(quiet=TRUE)
s$browse(1)

## End(Not run)
s$print()

# Be sure to install the Hmisc package before installing and running
# this application. You will want to; it's a pretty good one.
# s$add(
#    app=system.file('exampleApps/Hmisc/config.R',package='Rook'),
#    name='hmisc')

s$add(
    app=system.file('exampleApps/helloworld.R',package='Rook'),
    name='hello'
)
s$add(
```

```
    app=system.file('exampleApps/helloworldref.R',package='Rook'),
    name='helloref'
)
s$add(
    app=system.file('exampleApps/summary.R',package='Rook'),
    name='summary'
)

s$print()

#  Stops the server but doesn't uninstall the app
## Not run:
s$stop()

## End(Not run)
s$remove(all=TRUE)
rm(s)
```

`RhttpdApp-class` *Class* `RhttpdApp`

Description

Creates a Rook application ready to add to an `Rhttpd` server.

Details

The internal web server allows dispatching to user-defined closures located in tools:::.httpd.handlers.env. For instance, if a handler named 'foo' is placed there, then the url path to that handler is /custom/foo.

`RhttpdApp` along with `Rhttpd` hide these details by allowing a user to create application objects specifying only their name and the application. There is currently a limit of 63 characters or less for application names.

NOTE: When a file is given as the value of the `app` argument to `new()`, it is monitored for timestamp changes. If a change occurs in the modification time as returned by `file.info`, then the file is sourced prior to handling subsequent requests.

Methods

`new(app, name)`: Creates an object of class `RhttpdApp`. Argument `app` can be any `Rook` aware object or it can be a location to a file whose source creates a Rook aware object. That object must be named either `'app'` or the value of `name`. `name` is a character vector.

See Also

`Rhttpd`.

Examples

```
s <- Rhttpd$new()
s$add(RhttpdApp$new(
    name='summary',
    app=system.file('exampleApps/summary.R',package='Rook')
))
## Not run:
s$start(quiet=TRUE)
s$browse(1)

## End(Not run)
s$remove(all=TRUE)

#  Stops the server but doesn't uninstall the app
## Not run:
s$stop()

## End(Not run)
s$remove(all=TRUE)
rm(s)
```

RhttpdErrorStream-class   *Class* `RhttpdErrorStream`

Description

An internal class used by `Rhttpd`.

Examples

```
showClass("RhttpdErrorStream")
```

RhttpdInputStream-class   *Class* `RhttpdInputStream`

Description

An internal class used by `Rhttpd`.

Examples

```
showClass("RhttpdInputStream")
```

`Server` *Rook Server Object*

Description

`Server` is an object exported by Rook that has no value to the user. It is mainly used by web servers for their convenience. To see an example of how it may be used, see rApache.R in the inst/servers directory.

`Static-class` *Class* `Static`

Description

A `Middleware` class for serving static files from a root directory given a set of url paths.

Methods

`new(urls, root)`: Creates a new object. `urls` is a character vector whose elements must start with a `'/'`. `root` is a length 1 character vector whose value must be a valid directory.

See Also

See `Builder` for an example.

`suspend_console` *Suspend the R console*

Description

Calls Sys.sleep in a never-ending while loop to mimic suspension of the R console.

Usage

```
suspend_console()
```

Value

No value is ever returned.

See Also

`Rook`.

URLMap-class *Class* URLMap

Description

A Rook application that maps url paths to other Rook applications.

Methods

new(...): Creates a Rook application. All arguments must be Rook applications and named as in the example.

See Also

Rhttpd.

Examples

```
s <- Rhttpd$new()
s$add(
    name="pingpong",
    app=Rook::URLMap$new(
        '/ping' = function(env){
            req <- Rook::Request$new(env)
            res <- Rook::Response$new()
            res$write(sprintf('<h1><a href="%s">Pong</a></h1>',req$to_url("/pong")))
            res$finish()
        },
        '/pong' = function(env){
            req <- Rook::Request$new(env)
            res <- Rook::Response$new()
            res$write(sprintf('<h1><a href="%s">Ping</a></h1>',req$to_url("/ping")))
            res$finish()
        },
        '/?' = function(env){
            req <- Rook::Request$new(env)
            res <- Rook::Response$new()
            res$redirect(req$to_url('/pong'))
            res$finish()
        }
    )
)
## Not run:
s$start(quiet=TRUE)
s$browse('pingpong')

## End(Not run)
s$remove('pingpong')
## Not run:
s$stop()
```

```
## End(Not run)
rm(s)
```

`Utils-class` *Class* `Utils`

Description

A convenience object for working with various aspects of web requests and responses.

Methods

`bytesize(string=NULL)`: Returns size in bytes for `string`, a character vector.

`unescape(s=NULL)`: returns the url decoded value of the character vector `s`. Also replaces the `'+'` character with a space.

`status_code(status=NULL)`: returns integer value for the given HTTP `status`, which can either be numeric or or a character vector describing the status. Returns `as.integer(500)` if `status` is NULL.

`escape_html(string=NULL)`: replaces `"&"`, `"<"`, `">"`, `"'"`, and `'"'` with entity equivalents.

`raw.match(needle=NULL, haystack=NULL, all=TRUE)`: returns index position of `needle` in `haystack`. All matched indexes are returned by default. `needle` is either a raw vector or character string. `haystack` is a raw vector.

`parse_query(qs=NULL, d=DEFAULT_SEP)`: Creates a named list from the the query string `qs`. `d` is the separator value and defaults to `'[&;] *'`.

`rfc2822(ts=NULL)`: Formats `ts` in RFC2822 time. `ts` must be a `POSIXt` object.

`escape(s=NULL)`: Transforms any non-printable characters found in `s` to their percent-encoded equivalents.

`build_query(params=NULL)`: Creates a query string from the named list given in `params`.

`timezero()`: Returns a `POSIXct` object set to UNIX epoch.

`set_cookie_header(header, key, value, expires, path, domain, secure, httpOnly)`: Sets an HTTP cookie header in the environment `header`. All arguments except `expires` are length 1 character vectors, while `expires` must be a `POSIXct` object.

`delete_cookie_header(header, key, value, expires, path, domain, secure, httpOnly)`: Deletes the HTTP cookie header.

See Also

`Multipart`.

Examples

```
Utils$bytesize('foo')
Utils$escape('foo bar')
Utils$unescape('foo+bar')
Utils$escape_html('foo <bar>')
Utils$escape('foo <bar>')
Utils$escape('foo\n<bar>')
Utils$status_code('OK')
Utils$status_code('Found')
Utils$status_code('Not Found')
x <- Utils$parse_query('foo=1&bar=baz')
x
Utils$rfc2822(Sys.time())
Utils$timezero()
Utils$build_query(x)
rm(x)
```

Index