Package 'webmockr'

August 9, 2019

Title Stubbing and Setting Expectations on 'HTTP' Requests

Description Stubbing and setting expectations on 'HTTP' requests. Includes tools for stubbing 'HTTP' requests, including expected request conditions and response conditions. Match on 'HTTP' method, query parameters, request body, headers and more. Can be used for unit tests or outside of a testing context.

Version 0.4.0

License MIT + file LICENSE

URL https://github.com/ropensci/webmockr (devel)
https://ropenscilabs.github.io/http-testing-book/ (user manual)

BugReports https://github.com/ropensci/webmockr/issues

LazyData true

Encoding UTF-8

Imports curl, jsonlite, magrittr (>= 1.5), R6 (>= 2.1.3), urltools (>= 1.6.0), fauxpas, crul (>= 0.7.0)

Suggests roxygen2 (>= 6.1.1), testthat, xml2, vcr, httr

RoxygenNote 6.1.1

X-schema.org-applicationCategory Web

X-schema.org-keywords http, https, API, web-services, curl, mock, mocking, fakeweb, http-mocking, testing, testing-tools, tdd

X-schema.org-isPartOf https://ropensci.org

NeedsCompilation no

Author Scott Chamberlain [aut, cre] (<https://orcid.org/0000-0003-1444-9135>), rOpenSci [fnd] (https://ropensci.org)

Maintainer Scott Chamberlain <myrmecocystus+r@gmail.com>

Repository CRAN

Date/Publication 2019-08-09 21:50:02 UTC

R topics documented:

`webmockr-package` *Stubbing and setting expectations on HTTP requests*

Description

Stubbing and setting expectations on HTTP requests

Features

- Stubbing HTTP requests at low http client lib level
- Setting and verifying expectations on HTTP requests
- Matching requests based on method, URI, headers and body
- Supports multiple HTTP libraries, including **crul** and **httr**
- Integration with HTTP test caching library **vcr**

Author(s)

Scott Chamberlain <myrmecocystus+r@gmail.com>

Examples

```
library(webmockr)
stub_request("get", "https://httpbin.org/get")
stub_request("post", "https://httpbin.org/post")
stub_registry()
```

---

build_crul_request *Build a crul request*

Description

Build a crul request

Usage

```
build_crul_request(x)
```

Arguments

x	an unexecuted crul request object

Value

a crul request

---

build_crul_response *Build a crul response*

Description

Build a crul response

Usage

```
build_crul_response(req, resp)
```

Arguments

req	a request
resp	a response

Value

a crul response

build_httr_request *Build a httr request*

Description

Build a httr request

Usage

```
build_httr_request(x)
```

Arguments

x	an unexecuted httr request object

Value

a httr request

build_httr_response *Build a httr response*

Description

Build a httr response

Usage

```
build_httr_response(req, resp)
```

Arguments

req	a request
resp	a response

Value

a httr response

CrulAdapter *crul library adapter*

Description

crul library adapter

Details

Methods

`enable()` Enable the adapter

`disable()` Disable the adapter

`build_crul_request(x)` Build a crul RequestSignature x: crul request parts (list)

`build_crul_response(req, resp)` Build a crul response req: a crul request (list) resp: a crul response ()

`handle_request()` All logic for handling a request req: a crul request (list)

`remove_crul_stubs()` Remove all crul stubs

This adapter modifies **crul** to allow mocking HTTP requests

See Also

Other http_lib_adapters: `HttrAdapter`

enable *Enable or disable webmockr*

Description

Enable or disable webmockr

Usage

```
enable(adapter = NULL, options = list())

enabled(adapter = "crul")

disable(adapter = NULL, options = list())
```

Arguments

`adapter`	(character) the adapter name, 'crul' or 'httr'. one or the other. if none given, we attempt to enable both adapters
`options`	list of options - ignored for now.

Details

`enable()` enables **webmockr** for all adapters. `disable()` disables **webmockr** for all adapters. `enabled()` answers whether **webmockr** is enabled for a given adapter

Value

`enable()` and `disable()` invisibly returns booleans for each adapter, as a result of running enable or disable, respectively, on each HttpLibAdapaterRegistry object. `enabled` returns a single boolean

---

HashCounter *hash with counter, to store requests, and count each time it is used*

Description

hash with counter, to store requests, and count each time it is used

Details

Methods

`put(key)` Register a request by it's key - key: a character string of the request, serialized from CrulAdapter or other adapter

`get(key)` Get a request by key - key: a character string of the request, serialized from CrulAdapter or other adapter

See Also

Other request-registry: `RequestRegistry`, `request_registry`

Examples

```
x <- HashCounter$new()
x$put("foo bar")
x$put("foo bar")
x$put("hello world")
x$put("hello world")
x$put("hello world")
x$hash
```

HttpLibAdapaterRegistry *http lib adapter registry*

Description

http lib adapter registry

Details

Methods

`register(x)` Register an http library adapter x: an http lib adapter, e.g., CrulAdapter return: nothing, registers the library adapter

Examples

```
x <- HttpLibAdapaterRegistry$new()
x$register(CrulAdapter$new())
x
x$adapters
x$adapters[[1]]$name
```

HttrAdapter *httr library adapter*

Description

httr library adapter

Details

Methods

`enable()` Enable the adapter

`disable()` Disable the adapter

`build_httr_request(x)` Build a httr RequestSignature x: httr request parts (list)

`build_httr_response(req, resp)` Build a httr response req: a httr request (list) resp: a httr response ()

`handle_request()` All logic for handling a request req: a httr request (list)

`remove_httr_stubs()` Remove all httr stubs

This adapter modifies **httr** to allow mocking HTTP requests

See Also

Other http_lib_adapters: CrulAdapter

Examples

```
## Not run:
if (requireNamespace("httr", quietly = TRUE)) {
library(httr)

# normal httr request, works fine
real <- GET("https://httpbin.org/get")
real

# with webmockr
library(webmockr)
## turn on httr mocking
httr_mock()
## now this request isn't allowed
# GET("https://httpbin.org/get")
## stub the request
stub_request('get', uri = 'https://httpbin.org/get') %>%
  wi_th(
    headers = list('Accept' = 'application/json, text/xml, application/xml, */*')
  ) %>%
  to_return(status = 418, body = "I'm a teapot!", headers = list(a = 5))
## now the request succeeds and returns a mocked response
(res <- GET("https://httpbin.org/get"))
res$status_code
rawToChar(res$content)

# allow real requests while webmockr is loaded
webmockr_allow_net_connect()
webmockr_net_connect_allowed()
GET("https://httpbin.org/get?animal=chicken")
webmockr_disable_net_connect()
webmockr_net_connect_allowed()
# GET("https://httpbin.org/get?animal=chicken")
}

## End(Not run)
```

---

httr_mock *Turn on httr mocking*

Description

Turn on httr mocking

Usage

```
httr_mock(on = TRUE)
```

Arguments

on	(logical) set to TRUE to turn on, and FALSE to turn off. default: TRUE

Value

silently sets a callback that routes httr request through webmockr

---

remove_request_stub *Remove a request stub*

Description

Remove a request stub

Usage

```
remove_request_stub(stub)
```

Arguments

stub	a request stub, of class StubbedRequest

Value

logical, TRUE if removed, FALSE if not removed

See Also

Other stub-registry: StubRegistry, stub_registry_clear, stub_registry

Examples

```
(x <- stub_request("get", "https://httpbin.org/get"))
stub_registry()
remove_request_stub(x)
stub_registry()
```

`RequestPattern` *RequestPattern class*

Description

RequestPattern class

Arguments

`method`	the HTTP method (any, head, options, get, post, put, patch, trace, or delete). "any" matches any HTTP method. required.
`uri`	(character) request URI. required or uri_regex
`uri_regex`	(character) request URI as regex. required or uri
`query`	(list) query parameters, optional
`body`	(list) body request, optional
`headers`	(list) headers, optional

Details

Methods

`matches(request_signature)` Test if request_signature matches a pattern - request_signature: a request signature

`to_s()` Print pattern for easy human consumption

See Also

pattern classes for HTTP method MethodPattern, headers HeadersPattern, body BodyPattern, and URI/URL UriPattern

Examples

```
## Not run:
(x <- RequestPattern$new(method = "get", uri = "https://httpbin.org/get"))
x$body_pattern
x$headers_pattern
x$method_pattern
x$uri_pattern
x$to_s()

# make a request signature
rs <- RequestSignature$new(method = "get", uri = "https://httpbin.org/get")

# check if it matches
x$matches(rs)

# regex uri
```

```
(x <- RequestPattern$new(method = "get", uri_regex = ".+ossref.org"))
x$uri_pattern
x$uri_pattern$to_s()
x$to_s()

# uri with query parameters
(x <- RequestPattern$new(
    method = "get", uri = "https://httpbin.org/get",
    query = list(foo = "bar")
))
x$to_s()

# just headers (via setting method=any & uri_regex=.+)
headers <- list(
  'User-Agent' = 'Apple',
  'Accept-Encoding' = 'gzip, deflate',
  'Accept' = 'application/json, text/xml, application/xml, */*')
x <- RequestPattern$new(
   method = "any",
   uri_regex = ".+",
   headers = headers)
x$to_s()
rs <- RequestSignature$new(method = "any", uri = "http://foo.bar",
  options = list(headers = headers))
rs
x$matches(rs)

## End(Not run)
```

`RequestRegistry` *Request registry*

Description

Request registry

Details

Methods

`register_request(request)` Register a request - request: a character string of the request, serialized from CrulAdapter or other adapter

`reset()` Reset the registry to no registered requests

See Also

Other request-registry: `HashCounter`, `request_registry`

Examples

```
x <- RequestRegistry$new()
x$register_request(request = "GET http://scottchamberlain.info")
x$register_request(request = "GET http://scottchamberlain.info")
x$register_request(request = "POST https://httpbin.org/post")
# print method to list requests
x

# hashes, and number of times each requested
x$request_signatures$hash

# reset the request registry
x$reset()
```

RequestSignature *General purpose request signature builder*

Description

General purpose request signature builder

Arguments

`method`	the HTTP method (any, head, options, get, post, put, patch, trace, or delete). "any" matches any HTTP method. required.
`uri`	(character) request URI. required.
`options`	(list) options. optional. See Details.

Details

Methods

`to_s()` Request signature to a string return: a character string representation of the request signature

options

- body - body as a named list
- headers - headers as a named list
- proxies - proxies as a named list
- auth - authentication details, as a named list

Examples

```
# make request signature
x <- RequestSignature$new(method = "get", uri = "https:/httpbin.org/get")
# method
x$method
# uri
x$uri
# request signature to string
x$to_s()

# headers
w <- RequestSignature$new(
  method = "get",
  uri = "https:/httpbin.org/get",
  options = list(headers = list(`User-Agent` = "foobar", stuff = "things"))
)
w
w$headers
w$to_s()

# headers and body
bb <- RequestSignature$new(
  method = "get",
  uri = "https:/httpbin.org/get",
  options = list(
    headers = list(`User-Agent` = "foobar", stuff = "things"),
    body = list(a = "tables")
  )
)
bb
bb$headers
bb$body
bb$to_s()
```

---

request_registry *List requests in the request registry*

Description

List requests in the request registry

Usage

```
request_registry()
```

Value

an object of class `RequestRegistry`, print method gives the requests in the registry and the number of times each one has been performed

See Also

Other request-registry: HashCounter, RequestRegistry

Examples

```
webmockr::enable()
stub_request("get", "https://httpbin.org/get") %>%
  to_return(body = "success!", status = 200)

# nothing in the request registry
request_registry()

# make the request
z <- crul::HttpClient$new(url = "https://httpbin.org")$get("get")

# check the request registry - the request was made 1 time
request_registry()

# do the request again
z <- crul::HttpClient$new(url = "https://httpbin.org")$get("get")

# check the request registry - now it's been made 2 times, yay!
request_registry()
```

Response *Response class*

Description

Response class

Arguments

options	(list) a list of options

Details

Methods

`set_request_headers(headers)` set request headers - headers: a list of key-value pair headers

`get_request_headers()` get request headers

`set_response_headers(headers)` set response headers - headers: a list of key-value pair headers

`get_response_headers()` get response headers

`set_body(body)` - body: must be a string

`get_body()` get body

`set_status()` - body: must be an integer status code

`get_status()` get status code

`set_exception()` set exception

`get_exception()` get exception

Examples

```
## Not run:
(x <- Response$new())

x$set_url("https://httpbin.org/get")
x

x$set_request_headers(list('Content-Type' = "application/json"))
x
x$request_headers

x$set_response_headers(list('Host' = "httpbin.org"))
x
x$response_headers

x$set_status(404)
x
x$get_status()

x$set_body("hello world")
x
x$get_body()

x$set_exception("exception")
x
x$get_exception()

## End(Not run)
```

`StubbedRequest` *StubbedRequest class*

Description

StubbedRequest class

Arguments

`method`	the HTTP method (any, head, get, post, put, patch, or delete). "any" matches any HTTP method. required.
`uri`	(character) request URI. either this or `uri_regex` required
`uri_regex`	(character) request URI as regex. either this or `uri` required

Details

Methods

`with(query, body, headers)` Set expectations for what's given in HTTP request

- query (list) request query params, as a named list. optional
- body (list) request body, as a named list. optional
- headers (list) request headers as a named list. optional.

`to_return(status, body, headers)` Set expectations for what's returned in HTTP response

- status (numeric) an HTTP status code
- body (list) response body, as a list. optional
- headers (list) named list, response headers. optional.

`to_s()` Response as a string

See Also

`stub_request()`

Examples

```
## Not run:
x <- StubbedRequest$new(method = "get", uri = "api.crossref.org")
x$method
x$uri
x$with(headers = list('User-Agent' = 'R', apple = "good"))
x$to_return(status = 200, body = "foobar", headers = list(a = 5))
x
x$to_s()

# uri_regex
(x <- StubbedRequest$new(method = "get", uri_regex = ".+ossref.org"))
x$method
x$uri
x$to_s()

(x <- StubbedRequest$new(method = "get", uri_regex = ".+ossref.org"))
x$to_s()
x$timeout <- TRUE
x$to_s()

## End(Not run)
```

StubRegistry *Stub registry*

Description

Stub registry

Details

Methods

`register_stub(stub)` Register a stub - stub: an object of class StubbedRequest

`find_stubbed_request(req)` Find a stubbed request - req: an object of class RequestSignature

`response_for_request(request_signature)` Find a stubbed request - request_signature: an object of class RequestSignature

`request_stub_for(request_signature)` Find a stubbed request - request_signature: an object of class RequestSignature

`remove_request_stub(stub)` Remove a stubbed request by matching request signature - stub: an object of class StubbedRequest

`remove_all_request_stubs()` Remove all request stubs

`is_registered(x)` Find a stubbed request - x: an object of class RequestSignature

See Also

Other stub-registry: `remove_request_stub`, `stub_registry_clear`, `stub_registry`

Examples

```
## Not run:
# Make a stub
stub1 <- StubbedRequest$new(method = "get", uri = "api.crossref.org")
stub1$with(headers = list('User-Agent' = 'R'))
stub1$to_return(status = 200, body = "foobar", headers = list())
stub1

# Make another stub
stub2 <- StubbedRequest$new(method = "get", uri = "api.crossref.org")
stub2

# Put both stubs in the stub registry
reg <- StubRegistry$new()
reg$register_stub(stub = stub1)
reg$register_stub(stub = stub2)
reg
reg$request_stubs

## End(Not run)
```

---

stub_registry *List stubs in the stub registry*

Description

List stubs in the stub registry

Usage

```
stub_registry()
```

Value

an object of class `StubRegistry`, print method gives the stubs in the registry

See Also

Other stub-registry: `StubRegistry`, `remove_request_stub`, `stub_registry_clear`

Examples

```
# make a stub
stub_request("get", "https://httpbin.org/get") %>%
  to_return(body = "success!", status = 200)

# check the stub registry, there should be one in there
stub_registry()

# make another stub
stub_request("get", "https://httpbin.org/get") %>%
  to_return(body = "woopsy", status = 404)

# check the stub registry, now there are two there
stub_registry()

# to clear the stub registry
stub_registry_clear()
```

`stub_registry_clear` *Clear the stub registry*

Description

Clear all stubs

Usage

```
stub_registry_clear()
```

Value

nothing, well technically an empty list invisibly, but it's not anything useful

See Also

Other stub-registry: `StubRegistry`, `remove_request_stub`, `stub_registry`

Examples

```
(x <- stub_request("get", "https://httpbin.org/get"))
stub_registry()
stub_registry_clear()
stub_registry()
```

`stub_request` *Stub an http request*

Description

Stub an http request

Usage

```
stub_request(method = "get", uri = NULL, uri_regex = NULL)
```

Arguments

`method`	(character) HTTP method, one of "get", "post", "put", "patch", "head", "delete", "options" - or the special "any" (for any method)
`uri`	(character) The request uri. Can be a full uri, partial, or a regular expression to match many incantations of a uri. required.
`uri_regex`	(character) A URI represented as regex. See examples

Details

Internally, this calls StubbedRequest which handles the logic

See `stub_registry()` for listing stubs, `stub_registry_clear()` for removing all stubs and `remove_request_stub()` for removing specific stubs

If multiple stubs match the same request, we use the first stub. So if you want to use a stub that was created after an earlier one that matches, remove the earlier one(s).

Value

an object of class `StubbedRequest`, with print method describing the stub.

See Also

`wi_th()`, `to_return()`, `to_timeout()`, `to_raise()`

Examples

```
## Not run:
# basic stubbing
stub_request("get", "https://httpbin.org/get")
stub_request("post", "https://httpbin.org/post")

# any method, use "any"
stub_request("any", "https://httpbin.org/get")

# list stubs
stub_registry()

# request headers
stub_request("get", "https://httpbin.org/get") %>%
   wi_th(headers = list('User-Agent' = 'R'))

# request body
stub_request("post", "https://httpbin.org/post") %>%
   wi_th(body = list(foo = 'bar'))
stub_registry()
library(crul)
x <- crul::HttpClient$new(url = "https://httpbin.org")
crul::mock()
x$post('post', body = list(foo = 'bar'))

# add expectation with to_return
stub_request("get", "https://httpbin.org/get") %>%
  wi_th(
    query = list(hello = "world"),
    headers = list('User-Agent' = 'R')) %>%
  to_return(status = 200, body = "stuff", headers = list(a = 5))

# list stubs again
stub_registry()

# regex
stub_request("get", uri_regex = ".+ample\\..")

# set stub an expectation to timeout
stub_request("get", "https://httpbin.org/get") %>% to_timeout()
x <- crul::HttpClient$new(url = "https://httpbin.org")
res <- x$get('get')

# raise exception
library(fauxpas)
stub_request("get", "https://httpbin.org/get") %>% to_raise(HTTPAccepted)
stub_request("get", "https://httpbin.org/get") %>% to_raise(HTTPAccepted, HTTPGone)

x <- crul::HttpClient$new(url = "https://httpbin.org")
stub_request("get", "https://httpbin.org/get") %>% to_raise(HTTPBadGateway)
crul::mock()
x$get('get')
```

```
# pass options to .list to avoid NSE
z <- stub_request("get", "https://httpbin.org/get")
wi_th(z, .list = list(query = list(foo = "bar")))

# just body
stub_request("any", uri_regex = ".+") %>%
   wi_th(body = list(foo = 'bar'))
library(crul)
x <- crul::HttpClient$new(url = "https://httpbin.org")
crul::mock()
x$post('post', body = list(foo = 'bar'))
x$put('put', body = list(foo = 'bar'))

# just headers
headers <- list(
  'Accept-Encoding' = 'gzip, deflate',
  'Accept' = 'application/json, text/xml, application/xml, */*')
stub_request("any", uri_regex = ".+") %>% wi_th(headers = headers)
library(crul)
x <- crul::HttpClient$new(url = "https://httpbin.org", headers = headers)
crul::mock()
x$post('post')
x$put('put', body = list(foo = 'bar'))
x$get('put', query = list(stuff = 3423234L))

# clear all stubs
stub_registry()
stub_registry_clear()

## End(Not run)
```

`to_raise` *Set raise error condition*

Description

Set raise error condition

Usage

```
to_raise(.data, ...)
```

Arguments

`.data`	input. Anything that can be coerced to a `StubbedRequest` class object
`...`	One or more HTTP exceptions from the **fauxpas** package. Run `grep("HTTP*",getNamespaceExports("` `= TRUE)` for a list of possible exceptions

Details

The behavior in the future will be:

When multiple exceptions are passed, the first is used on the first mock, the second on the second mock, and so on. Subsequent mocks use the last exception

But for now, only the first exception is used until we get that fixed

Value

an object of class `StubbedRequest`, with print method describing the stub

Note

see examples in `stub_request()`

`to_return` *Expectation for what's returned from a stubbed request*

Description

Set response status code, response body, and/or response headers

Usage

```
to_return(.data, ..., .list = list())
```

Arguments

`.data`	input. Anything that can be coerced to a `StubbedRequest` class object
`...`	Comma separated list of named variables. accepts the following: `status`, `body`, `headers`. See Details for more.
`.list`	named list, has to be one of 'status', 'body', and/or 'headers'. An alternative to passing in via `....` Don't pass the same thing to both, e.g. don't pass 'status' to `...`, and also 'status' to this parameter

Details

Values for status, body, and headers:

- status: (numeric/integer) three digit status code
- body: various, including character string, list, raw, numeric, etc
- headers: (list) a named list

response headers are returned with all lowercase names and the values are all of type character. if numeric/integer values are given (e.g., `to_return(headers = list(a = 10))`), we'll coerce any numeric/integer values to character.

Value

an object of class `StubbedRequest`, with print method describing the stub

Note

see more examples in `stub_request()`

Examples

```
# first, make a stub object
(req <- stub_request("post", "https://httpbin.org/post"))

# add status, body and/or headers
to_return(req, status = 200)
to_return(req, body = "stuff")
to_return(req, body = list(a = list(b = "world")))
to_return(req, headers = list(a = 5))
to_return(req, status = 200, body = "stuff", headers = list(a = 5))

# .list - pass in a named list instead
to_return(req, .list = list(body = list(foo = "bar")))
```

`to_timeout` *Set timeout as an expected return on a match*

Description

Set timeout as an expected return on a match

Usage

```
to_timeout(.data)
```

Arguments

`.data`	input. Anything that can be coerced to a `StubbedRequest` class object

Value

an object of class `StubbedRequest`, with print method describing the stub

Note

see examples in `stub_request()`

webmockr-defunct *Defunct functions in* **webmockr**

Description

- `webmockr_enable()`: Function removed, see `enable()`
- `webmockr_disable()`: Function removed, see `disable()`
- to_return_: Only `to_return()` is available now
- wi_th_: Only `wi_th()` is available now

webmockr_configure *webmockr configuration*

Description

webmockr configuration

Usage

```
webmockr_configure(allow_net_connect = FALSE, allow_localhost = FALSE,
  allow = NULL, net_http_connect_on_start = FALSE,
  show_stubbing_instructions = FALSE, query_values_notation = FALSE,
  show_body_diff = FALSE)

webmockr_configure_reset()

webmockr_configuration()

webmockr_allow_net_connect()

webmockr_disable_net_connect(allow = NULL)

webmockr_net_connect_allowed(uri = NULL)
```

Arguments

`allow_net_connect`	(logical) Default: `FALSE`
`allow_localhost`	(logical) Default: `FALSE`
`allow`	(character) one or more URI/URL to allow (and by extension all others are not allowed)
`net_http_connect_on_start`	(logical) Default: `FALSE`. ignored for now

show_stubbing_instructions
: (logical) Default: FALSE. ignored for now

query_values_notation
: (logical) Default: FALSE. ignored for now

show_body_diff
: (logical) Default: FALSE. ignored for now

uri
: (character) a URI/URL as a character string - to determine whether or not it is allowed

Examples

```
## Not run:
webmockr_configure()
webmockr_configure(
 allow_localhost = TRUE
)
webmockr_configuration()
webmockr_configure_reset()

webmockr_allow_net_connect()
webmockr_net_connect_allowed()

# disable net connect for any URIs
webmockr_disable_net_connect()
### gives NULL with no URI passed
webmockr_net_connect_allowed()
# disable net connect EXCEPT FOR given URIs
webmockr_disable_net_connect(allow = "google.com")
### is a specific URI allowed?
webmockr_net_connect_allowed("google.com")

## End(Not run)
```

---

wi_th *Set additional parts of a stubbed request*

Description

Set query params, request body, and/or request headers

Usage

```
wi_th(.data, ..., .list = list())
```

Arguments

.data
: input. Anything that can be coerced to a StubbedRequest class object

...
: Comma separated list of named variables. accepts the following: query, body, headers.

.list	named list, has to be one of 'query', 'body', and/or 'headers'. An alternative to passing in via .... Don't pass the same thing to both, e.g. don't pass 'query' to ..., and also 'query' to this parameter

Details

with is a function in the base package, so we went with wi_th

Values for query, body, and headers:

- query: (list) a named list
- body: various, including character string, list, raw, numeric, etc
- headers: (list) a named list

Value

an object of class StubbedRequest, with print method describing the stub

Note

see more examples in stub_request()

Examples

```
# first, make a stub object
req <- stub_request("post", "https://httpbin.org/post")

# add body
# list
wi_th(req, body = list(foo = "bar"))
# string
wi_th(req, body = '{"foo": "bar"}')
# raw
wi_th(req, body = charToRaw('{"foo": "bar"}'))
# numeric
wi_th(req, body = 5)

# add query - has to be a named list
wi_th(req, query = list(foo = "bar"))

# add headers - has to be a named list
wi_th(req, headers = list(foo = "bar"))
wi_th(req, headers = list(`User-Agent` = "webmockr/v1", hello="world"))


# .list - pass in a named list instead
wi_th(req, .list = list(body = list(foo = "bar")))
```

Index