Package 'fauxpas'

October 13, 2022

Title HTTP Error Helpers

Description HTTP error helpers. Methods included for general purpose HTTP error handling, as well as individual methods for every HTTP status code, both via status code numbers as well as their descriptive names. Supports ability to adjust behavior to stop, message or warning. Includes ability to use custom whisker template to have any configuration of status code, short description, and verbose message. Currently supports integration with 'crul', 'curl', and 'httr'.

Version 0.5.0

License MIT + file LICENSE

URL https://docs.ropensci.org/fauxpas, https://github.com/ropensci/fauxpas

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

VignetteBuilder knitr

Encoding UTF-8

Language en-US

Imports R6 (>= 2.1.2), httpcode (>= 0.3.0), whisker

Suggests crul (>= 0.5.0), curl (>= 2.2), httr (>= 1.2.0), testthat, knitr, rmarkdown

RoxygenNote 7.1.0

X-schema.org-applicationCategory Web

X-schema.org-keywords http, https, API, web-services, curl, errors, error

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

NeedsCompilation no

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

Maintainer Scott Chamberlain <myrmecocystus@gmail.com>

Repository CRAN

Date/Publication 2020-04-13 16:10:02 UTC

R topics documented:

`fauxpas-package` *fauxpas*

Description

HTTP Error Helpers

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>

`Error` *Error class*

Description

Error class

Arguments

`behavior`	Behavior of the error. default: auto. See Details
`message_template`	A message template. optional. use whisker templating. names to use are: reason and status. use in template like `{{reason}}` and `{{status}}`. Note that `{{message}}` that is used in `message_template_verbose` will be ignored here.
`call.`	(logical) indicating if the call should become part of the error message. Default: `FALSE`
`message_template_verbose`	A verbose message template. optional. use whisker templating. names to use are: reason, status, message. use in template like `{{reason}}`, `{{status}}`, and `{{message}}`. Note that this is ignored here, but is used in the `HTTP*` methods (e.g. `HTTPBadRequest`)
`muffle`	(logical) whether to not respond when status codes in 1xx-3xx series. Default: `FALSE`

Details

Methods

- `do(response, mssg)`
  Execute condition, whether it be message, warning, or error.
  - response: is any response from **crul**, **curl**, or **httr** Execute condition, whether it be message, warning, error, or your own custom function. This method uses `message_template_verbose`, and uses it's default value.
  - mssg: character string message to include in call. ignored if template does not have a `message` entry
- `set_behavior(behavior)`
  Set behavior, same as setting behavior on initializing with `$new()`

behavior parameter options

- stop - use `stop`
- warning - use `warning`
- message - use `message`
- auto - toggle between `stop` and `message` depending on the HTTP status code series. Defaults will be:
  - 1xx: `message`
  - 2xx: `message`
  - 3xx: `message`
  - 4xx: `stop`
  - 5xx: `stop`

Of course, you can always override the defaults.

See Also

`http`, `Error-Classes`

Examples

```
Error$new()
# reset behavior
(z <- Error$new())
z$set_behavior("warning")
z

if (requireNamespace("crul")) {
 library("crul")
 res <- HttpClient$new("https://httpbin.org/status/418")$get()

 # stop
 (x <- Error$new(behavior = "stop"))
 ## Not run: x$do(res)
```

```
 # warn
 (x <- Error$new(behavior = "warning"))
 x$do(res)

 # do vs. do_verbose
 x <- HTTPRequestURITooLong$new(behavior = "stop")
 res <- HttpClient$new("https://httpbin.org/status/414")$get()
 ## Not run:
 http414(res)
 ## with template
 http414(res, message_template = "{{status}}\n  --> {{reason}}")
 x$do(res)
 x$do_verbose(res)

## End(Not run)

 # service unavailable
 x <- HTTPServiceUnavailable$new(behavior = "stop")
 res <- HttpClient$new("https://httpbin.org/status/503")$get()
 ## Not run:
 x$do(res)
 x$do_verbose(res)

## End(Not run)

 # message template
 y <- Error$new(message_template = "{{reason}} ..... {{status}}")
 res <- HttpClient$new("https://httpbin.org/status/418")$get()
 ## Not run:
 y$do(res)

## End(Not run)

 yy <- Error$new(message_template = "{{status}}\n  --> {{reason}}")
 yy$message_template
 ## Not run:
 yy$do(res)

## End(Not run)

 ## with verbose message
 library(crul)
 res <- HttpClient$new("https://httpbin.org/status/401")$get()
 yy <- HTTPUnauthorized$new()
 zz <- HTTPUnauthorized$new(
   message_template = "HTTP({{status}}): {{reason}}\n  {{message}}"
 )
 yy$message_template; zz$message_template
 ## Not run:
 yy$do(res)
 zz$do(res)
 yy$do_verbose(res)
 zz$do_verbose(res)
```

```
## End(Not run)

 yy <- Error$new(
   message_template = "HTTP({{status}}): {{reason}}\n  {{message}}"
 )
 yy$message_template
 ## Not run: yy$do(res)

 # muffle responses
 (x <- Error$new(muffle = TRUE))
 res <- crul::HttpClient$new("https://httpbin.org/status/226")$get()
 z <- x$do(res)
 z
}
```

`Error-Classes` *Individual error classes*

Description

These error classes are for each HTTP error, and inherit from the `Error` class in this package.

Details

In addition to what's available in `Error`, these classes have a single variable `mssg` that is the very verbose complete message describing the HTTP condition in detail. You can include that message in your condition by using `do_verbose` (see below)

Methods

In addition to the methods documented in `Error`, these methods also have:

- `do_verbose(response, template)`
  Execute condition, whether it be message, warning, or error.
  - response: is any response from **crul**, **curl**, or **httr** Execute condition, whether it be message, warning, error, or your own custom function. This method uses `message_template_verbose`, and uses it's default value.
  - template: a template to use for the verbose message, see `Error` for details

See Also

`Error`, `http`

Examples

```
if (requireNamespace("crul")) {

 library("crul")
 res <- HttpClient$new("https://httpbin.org/status/414")$get()
```

```
 x <- HTTPRequestURITooLong$new()
 x
 ## Not run:
 x$do(res)
 x$do_verbose(res)

## End(Not run)

 # behavior
 x <- HTTPRequestURITooLong$new(behavior = "warning")
 ## Not run:
 x$do(res)
 x$do_verbose(res)

## End(Not run)

 x <- HTTPRequestURITooLong$new(behavior = "message")
 ## Not run:
 x$do(res)
 x$do_verbose(res)

## End(Not run)

 # with message template
 (x <- HTTPRequestURITooLong$new(
   message_template = "{{reason}} ..... {{status}}",
   message_template_verbose = "{{reason}} .>.>.>.>.>.> {{status}}\n {{message}}"
 ))
 ## Not run:
 x$do(res)
 x$do_verbose(res)

## End(Not run)
}
```

`find_error_class` *Find error classes*

Description

Find error classes

Usage

```
find_error_class(status_code)
```

Arguments

`status_code`	(numeric,integer) A status code

Value

an object of class `R6ClassGenerator`. call `$new()` to initialize a new instance

See Also

`Error`, `Error-Classes`

Examples

```
find_error_class(414)
find_error_class(418)
find_error_class(505)

# initialize the class
find_error_class(418)$new()

# not found
## Not run: find_error_class(999)
```

`http100` *higher level error wrappers*

Description

higher level error wrappers

Usage

```
http100(response, behavior = "auto", message_template, muffle = FALSE)

http101(response, behavior = "auto", message_template, muffle = FALSE)

http102(response, behavior = "auto", message_template, muffle = FALSE)

http200(response, behavior = "auto", message_template, muffle = FALSE)

http201(response, behavior = "auto", message_template, muffle = FALSE)

http202(response, behavior = "auto", message_template, muffle = FALSE)

http203(response, behavior = "auto", message_template, muffle = FALSE)

http204(response, behavior = "auto", message_template, muffle = FALSE)

http205(response, behavior = "auto", message_template, muffle = FALSE)

http206(response, behavior = "auto", message_template, muffle = FALSE)
```

```
http207(response, behavior = "auto", message_template, muffle = FALSE)

http208(response, behavior = "auto", message_template, muffle = FALSE)

http226(response, behavior = "auto", message_template, muffle = FALSE)

http300(response, behavior = "auto", message_template, muffle = FALSE)

http301(response, behavior = "auto", message_template, muffle = FALSE)

http302(response, behavior = "auto", message_template, muffle = FALSE)

http303(response, behavior = "auto", message_template, muffle = FALSE)

http304(response, behavior = "auto", message_template, muffle = FALSE)

http305(response, behavior = "auto", message_template, muffle = FALSE)

http306(response, behavior = "auto", message_template, muffle = FALSE)

http307(response, behavior = "auto", message_template, muffle = FALSE)

http308(response, behavior = "auto", message_template, muffle = FALSE)

http400(response, behavior = "auto", message_template, muffle = FALSE)

http401(response, behavior = "auto", message_template, muffle = FALSE)

http402(response, behavior = "auto", message_template, muffle = FALSE)

http403(response, behavior = "auto", message_template, muffle = FALSE)

http404(response, behavior = "auto", message_template, muffle = FALSE)

http405(response, behavior = "auto", message_template, muffle = FALSE)

http406(response, behavior = "auto", message_template, muffle = FALSE)

http407(response, behavior = "auto", message_template, muffle = FALSE)

http408(response, behavior = "auto", message_template, muffle = FALSE)

http409(response, behavior = "auto", message_template, muffle = FALSE)

http410(response, behavior = "auto", message_template, muffle = FALSE)

http411(response, behavior = "auto", message_template, muffle = FALSE)
```

```
http412(response, behavior = "auto", message_template, muffle = FALSE)

http413(response, behavior = "auto", message_template, muffle = FALSE)

http414(response, behavior = "auto", message_template, muffle = FALSE)

http415(response, behavior = "auto", message_template, muffle = FALSE)

http416(response, behavior = "auto", message_template, muffle = FALSE)

http417(response, behavior = "auto", message_template, muffle = FALSE)

http418(response, behavior = "auto", message_template, muffle = FALSE)

http419(response, behavior = "auto", message_template, muffle = FALSE)

http420(response, behavior = "auto", message_template, muffle = FALSE)

http422(response, behavior = "auto", message_template, muffle = FALSE)

http423(response, behavior = "auto", message_template, muffle = FALSE)

http424(response, behavior = "auto", message_template, muffle = FALSE)

http425(response, behavior = "auto", message_template, muffle = FALSE)

http426(response, behavior = "auto", message_template, muffle = FALSE)

http428(response, behavior = "auto", message_template, muffle = FALSE)

http429(response, behavior = "auto", message_template, muffle = FALSE)

http431(response, behavior = "auto", message_template, muffle = FALSE)

http440(response, behavior = "auto", message_template, muffle = FALSE)

http444(response, behavior = "auto", message_template, muffle = FALSE)

http449(response, behavior = "auto", message_template, muffle = FALSE)

http450(response, behavior = "auto", message_template, muffle = FALSE)

http451(response, behavior = "auto", message_template, muffle = FALSE)

http494(response, behavior = "auto", message_template, muffle = FALSE)

http495(response, behavior = "auto", message_template, muffle = FALSE)
```

```
http496(response, behavior = "auto", message_template, muffle = FALSE)

http497(response, behavior = "auto", message_template, muffle = FALSE)

http498(response, behavior = "auto", message_template, muffle = FALSE)

http499(response, behavior = "auto", message_template, muffle = FALSE)

http500(response, behavior = "auto", message_template, muffle = FALSE)

http501(response, behavior = "auto", message_template, muffle = FALSE)

http502(response, behavior = "auto", message_template, muffle = FALSE)

http503(response, behavior = "auto", message_template, muffle = FALSE)

http504(response, behavior = "auto", message_template, muffle = FALSE)

http505(response, behavior = "auto", message_template, muffle = FALSE)

http506(response, behavior = "auto", message_template, muffle = FALSE)

http507(response, behavior = "auto", message_template, muffle = FALSE)

http508(response, behavior = "auto", message_template, muffle = FALSE)

http509(response, behavior = "auto", message_template, muffle = FALSE)

http510(response, behavior = "auto", message_template, muffle = FALSE)

http511(response, behavior = "auto", message_template, muffle = FALSE)

http598(response, behavior = "auto", message_template, muffle = FALSE)

http599(response, behavior = "auto", message_template, muffle = FALSE)

http(response, behavior = "auto", message_template, muffle = FALSE)
```

Arguments

response	The result of a call via **crul**, **curl**, or **httr**
behavior	Behavior of the error. default: auto. See Details
message_template	A message template. optional. use whisker templating. names to use are: reason and status. use in template like {{reason}} and {{status}}. Note that {{message}} that is used in message_template_verbose will be ignored here.
muffle	(logical) whether to not respond when status codes in 1xx-3xx series. Default: FALSE

behavior parameter options

- stop - use `stop`
- warning - use `warning`
- message - use `message`
- auto - toggle between `stop` and `message` depending on the HTTP status code series. Defaults will be:
  - 1xx: `message`
  - 2xx: `message`
  - 3xx: `message`
  - 4xx: `stop`
  - 5xx: `stop`

Of course, you can always override the defaults.

using package curl

curl reponses are simple lists, so we have little to go on to make sure it's a response from the **curl** package. We check for list names internally but of course you could pass in a list with the right named elements, while the values are complete nonsense, in which case we'll probably fail badly. There's not much we can do.

Note

These `http*` methods only use `$do` and not `$do_verbose`.

See Also

`Error`, `Error-Classes`

Examples

```
if (requireNamespace("crul")) {
 library("crul")
 res <- HttpClient$new("https://httpbin.org/status/418")$get()
 ## Not run: http(res)
 http(res, behavior = "warning")
 http(res, behavior = "message")

 res <- HttpClient$new("https://httpbin.org/status/414")$get()
 ## Not run: http414(res)
 http(res, behavior = "warning")
 http(res, behavior = "message")

 res <- HttpClient$new("https://httpbin.org/asdfafadsf")$get()
 ## Not run: http404(res)
 http(res, behavior = "warning")
 http(res, behavior = "message")
}
```

```
if (requireNamespace("curl")) {
 library("curl")
 h <- curl::new_handle()
 curl::handle_setopt(h)
 res <- curl::curl_fetch_memory("https://httpbin.org/status/418", h)
 ## Not run: http(res)
 http(res, behavior = "warning")
 http(res, behavior = "message")
}

if (requireNamespace("httr")) {
 library("httr")
 res <- GET("https://httpbin.org/status/418")
 ## Not run: http(res)
 http(res, behavior = "warning")
 http(res, behavior = "message")
}

# muffle responses
if (requireNamespace("crul")) {
 library("crul")
 res201 <- HttpClient$new("https://httpbin.org")$get("status/201")
 res404 <- HttpClient$new("https://httpbin.org")$get("status/404")
 # status codes < 300 CAN be muffled - i.e., return the http response object
 http(res201, muffle = TRUE)
 # status codes > 300 CAN NOT be muffled - i.e., return the http response object
 ## Not run:
 http(res404, muffle = TRUE)

## End(Not run)
}
```

Index