Package: tryr (via r-universe)

August 28, 2024

Type Package

Title Client/Server Error Handling for HTTP API Frameworks

Version 0.1.2

Date 2024-05-29

Description Differentiate client errors (4xx) from server errors (5xx) for the 'plumber' and 'RestRserve' HTTP API frameworks. The package also includes a built-in logging mechanism to standard output (STDOUT) or standard error (STDERR) depending on the log level.

License MIT + file LICENSE

LazyLoad yes

RoxygenNote 7.3.1

Encoding UTF-8

Roxygen list(markdown = TRUE)

BugReports https://github.com/analythium/tryr/issues

URL https://github.com/analythium/tryr

Language en-US

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

RemoteUrl https://github.com/analythium/tryr

RemoteRef HEAD

RemoteSha 773b2ca0f0cc9fb8028cce2951c006a5b772fc40

Contents

`http-messages` *Generic HTTP Response Messages*

Description

These functions provide generic HTTP response messages based on the HTTP response status codes.

Usage

```
http_error(status = 500L, message = NULL)

http_success(status = 200L, message = NULL, body = NULL)

http_response(status = 200L, message = NULL, body = NULL)

http_handler(req, res, status = 200L, message = NULL, body = NULL)
```

Arguments

`status`	HTTP status code.
`message`	An HTTP response message or `NULL`. A generic response message is provided when it is `NULL` based on `http_status_codes`.
`body`	A list, additional values to be returned.
`req`	The request object.
`res`	The response object.

Value

`http_error` returns an error with a custom condition attribute after checking if the status code is at least 400.

`http_success` returns a list but checks that the status code is <400.

`http_response` returns a list checking only that the status code is valid.

`http_handler` behaves like `http_response` but it also sets the status code and the body of the response object.

See Also

`http_status_codes`

Examples

```
try(http_error())
try(http_error(400))
try(http_error(400, "Sorry"))

str(http_success())
str(http_success(201))
str(http_success(201, "Awesome"))

str(http_response(201, "Awesome", list(name = "Jane", count = 6)))

req <- new.env()
res <- new.env()
str(http_handler(req, res, 201, "Awesome", list(name = "Jane", count = 6)))
res$status
str(res$body)
```

`http-try` *Client/Server Error Handling*

Description

Differentiate between client (4xx) and server (5xx) errors. Provides a mechanism to return custom status codes in combination with `http_error()` and `http_success()`.

Usage

```
http_try(req, res, expr, silent = TRUE, ...)

http_try_handler(req, res, x)
```

Arguments

`req`	The request object.
`res`	The response object.
`expr`	An R expression to try.
`silent`	Logical, should the report of error messages be suppressed by `try()`?
`...`	Arguments passed to `try()`
`x`	The return value from `try(expr)`.

Details

If we catch an error:

- the error is a clear server error coming from `stop()`
  - log it as an ERROR + print the error message to STDERR
  - return a generic status 500 message
  - set the status code of the response object to 500
- the error is a structured HTTP error coming from `http_error()`
  - log it as an ERROR with the message from the condition attribute
  - return the specific HTTP error code with the structured output
  - set the status code of the response object

If we don't catch an error:

- the object is of class `http_success()` (this comes in handy for async jobs and redirects)
  - log it as a SUCCESS with the message element
  - return the specific HTTP status code with the structured output
  - set the status code of the response object
- the object is NOT of class `http_success()`
  - log it as a SUCCESS with a generic 200 message
  - return the object as is (default status code 200 assumed)

Value

A list or the results from `expr`. A side effect is setting of the response status code on the response object and a log message to STDOUT or STDERR.

See Also

`try()`, `msg()`

Examples

```
req <- new.env()
res <- new.env()
http_try(req, res)
res$status

req <- new.env()
res <- new.env()
http_try(req, res, { 2 + 2 })
res$status

req <- new.env()
res <- new.env()
http_try(req, res, http_error(401))
res$status
```

```
req <- new.env()
res <- new.env()
http_try(req, res, http_success(201))
res$status

req <- new.env()
res <- new.env()
http_try(req, res, { lm(NULL) })
res$status

req <- new.env()
res <- new.env()
http_try(req, res, { stop("Stop!!!") })
res$status

req <- new.env()
res <- new.env()
f <- function() stop("Stop!!!")
http_try(req, res, { f() })
res$status

req <- new.env()
res <- new.env()
http_try_handler(req, res, { try(f()) })
res$status
```

`http_status_codes` *HTTP Response Status Codes*

Description

Data frame with possible status codes and default messages based on RFC 9110. See also https://developer.mozilla.org/en-US/docs/Web/HTTP/Status.

Usage

```
http_status_codes
```

Format

An object of class `data.frame` with 62 rows and 3 columns.

Examples

```
str(http_status_codes)

http_status_codes[http_status_codes$category == "Successful",]

http_status_codes["500",]
```

msg *Write Logging Information to STDOUT or STDERR*

Description

A simple logging utility that writes the log message to STDOUT or STDERR in plain text, JSON, or comma separated format.

Usage

```
msg(title = "", message = "", level = "INFO", format = NULL, digits = NULL)

x %+% y
```

Arguments

title	Character, the title of the logging message.
message	Character, a more detailed logging message.
level	Log level; one of "ALL", "TRACE", "DEBUG", "INFO", "SUCCESS", "WARN", "ERROR", "FATAL", "OFF" (case insensitive).
format	Log format: "PLAIN" (default), "JSON", "CSV" (case insensitive).
digits	Integer of length 1, digits for seconds (default 3L meaning milliseconds).
x, y	Strings to combine.

Details

The TRYR_ERR_LEVEL environment variable determines where the log message is written. If the log level is at least the one or above the TRYR_ERR_LEVEL value (or "WARN" when it is unset or null) the message is written to STDERR, otherwise it is written to STDOUT.

The log message is only written when the log level is at least the one or higher than specified by the TRYR_LOG_LEVEL environment variables; or "INFO" when the variable is unset or null.

The log format can be plain text, JSON, or comma separated text. When the log format is NULL the TRYR_LOG_FORMAT environment variables is checked. If it is unset or null, the format is considered plain text.

The logging message will be formed by combining the title and the message parts. The log info also contains the process ID, a timestamp (using Sys.time()), and the log level. The timestamp prints out fractional seconds according to digits. When digits is NULL it checks the TRYR_LOG_DIGITS environment variables and uses that value. The default is 3 when TRYR_LOG_DIGITS unset or null.

Besides the usual log levels, there is an extra one "SUCCESS" that is used to signal successful HTTP response codes (2xx).

Value

msg invisibly returns logical indicating if the log message was written (TRUE) or not (FALSE). A side effect is a log message to STDOUT or STDERR.

The %+% special pastes the right and left hand side together into s single string.

See Also

paste0(), sprintf()

Examples

```
n <- 5
"Sample " %+% "size " %+% "n = " %+% n %+% "."

msg("Success", "We did it!")
msg("Success", "We did it!", "SUCCESS")
msg("Error", "Oh no! n cannot be " %+% n, "ERROR")

msg("Success", "We did it!", "SUCCESS", format = "JSON")
msg("Success", "We did it!", format = "JSON")
msg("Error", "Oh no ...", "ERROR", format = "JSON")

msg("Success", "We did it!", digits = 0)
msg("Success", "We did it!", digits = 6)
```

Index