Package ‘httpcache’

October 13, 2022

Type Package

Title Query Cache for HTTP Clients

Description In order to improve performance for HTTP API clients, 'httpcache' provides simple tools for caching and invalidating cache. It includes the HTTP verb functions GET, PUT, PATCH, POST, and DELETE, which are drop-in replacements for those in the 'httr' package. These functions are cache-aware and provide default settings for cache invalidation suitable for RESTful APIs; the package also enables custom cache-management strategies. Finally, 'httpcache' includes a basic logging framework to facilitate the measurement of HTTP request time and cache performance.

Version 1.2.0

URL https://enpiar.com/r/httpcache/,
https://github.com/nealrichardson/httpcache/

BugReports https://github.com/nealrichardson/httpcache/issues

License MIT + file LICENSE

Depends R (>= 3.0.0)

Imports digest, httr (>= 1.0.0), utils

Suggests httptest (>= 3.0.0), knitr, rmarkdown, spelling

RoxygenNote 7.1.1

VignetteBuilder knitr

Language en-US

Encoding UTF-8

NeedsCompilation no

Author Neal Richardson [aut, cre]

Maintainer Neal Richardson <neal.p.richardson@gmail.com>

Repository CRAN

Date/Publication 2021-01-10 23:10:02 UTC
**R topics documented:**

- buildCacheKey .......................... 2
- cache-api ................................ 3
- cache-management ....................... 3
- cached-http-verbs ....................... 4
- cachedPOST ................................ 5
- cacheLogSummary ......................... 5
- dropCache ................................ 6
- halt ....................................... 6
- loadLogfile ................................ 7
- logMessage ................................ 7
- requestLogSummary ....................... 8
- saveCache ................................ 8
- startLog .................................. 9
- uncached .................................. 9

**Index** .................................... 11

---

**buildCacheKey**  
*Construct a unique cache key for a request*

**Description**

This function encapsulates the logic of making a cache key, allowing other code or libraries to access the HTTP cache programmatically.

**Usage**

```r
buildCacheKey(url, query = NULL, body = NULL, extras = c())
```

**Arguments**

- `url` character request URL
- `query` Optional query parameters for the request
- `body` Optional request body
- `extras` character Optional additional annotations to include in the cache key.

**Value**

Character value, starting with `url` and including hashed query and body values if provided, to be used as the cache key for this request.
HTTP Cache API

Description

These functions provide access to what’s stored in the cache.

Usage

hitCache(key)
getCache(key)
setCache(key, value)

Arguments

key character, typically a URL or similar
value For setCache, an R object to set in the cache for key.

Value

hitCache returns logical whether key exists in the cache. getCache returns the value stored in the cache, or NULL if there is nothing cached. setCache is called for its side effects.

Manage the HTTP cache

Description

These functions turn the cache on and off and clear the contents of the query cache.

Usage

cacheOn()
cacheOff()
clearCache()

Value

Nothing. Functions are run for their side effects.
**cached-http-verbs**

**Cache-aware versions of httr verbs**

**Description**

These functions set, read from, and bust the HTTP query cache. They wrap the similarly named functions in the httr package and can be used as drop-in replacements for them.

**Usage**

```
GET(url, ...)

PUT(url, ..., drop = dropCache(url))

POST(url, ..., drop = dropOnly(url))

PATCH(url, ..., drop = dropCache(url))

DELETE(url, ..., drop = dropCache(url))
```

**Arguments**

- `url` character URL of the request
- `...` additional arguments passed to the httr functions
- `drop` For PUT, PATCH, POST, and DELETE, code to be executed after the request. This is intended to be for supplying cache-invalidation logic. By default, POST drops cache only for the specified `url` (i.e. `dropOnly()`), while the other verbs drop cache for the request URL and for any URLs nested below it (i.e. `dropCache()`).

**Details**

GET checks the cache before making an HTTP request, and if there is a cache miss, it sets the response from the request into the cache for future requests. The other verbs, assuming a more or less RESTful API, would be assumed to modify server state, and thus they should trigger cache invalidation. They have default cache-invalidation strategies, but you can override them as desired.

**Value**

The corresponding httr response object, potentially read from cache

**See Also**

`dropCache()` `cachedPOST()`
cachedPOST

Cache the response of a POST

Description
Some APIs have resources where a POST is used to send a command that returns content and doesn’t modify state. In this case, it’s more like a GET. This may occur where one might normally GET but the request URI would be too long for the server to accept. cachedPOST thus behaves more like GET, checking for a cached response before performing the request and setting cache if the request is successful. It does no cache dropping, unlike POST().

Usage
cachedPOST(url, ...)

Arguments
url character URL of the request
... additional arguments passed to the httr functions

Value
The corresponding httr response object, potentially read from cache

cacheLogSummary

Summarize cache performance from a log

Description
Summarize cache performance from a log

Usage
cacheLogSummary(logdf)

Arguments
logdf A logging data.frame, as loaded by loadLogfile().

Value
A list containing counts of cache hit/set/drop events, plus a cache hit rate.
dropCache

Invalidate cache

Description

These functions let you control cache invalidation. dropOnly invalidates cache only for the specified URL. dropPattern uses regular expression matching to invalidate cache. dropCache is a convenience wrapper around dropPattern that invalidates cache for any resources that start with the given URL.

Usage

dropCache(x)
dropOnly(x)
dropPattern(x)

Arguments

x character URL or regular expression

Value

Nothing. Functions are run for their side effects.

halt

Stop, log, and no call

Description

Wrapper around base::stop() that logs the error message and then stops with call.=FALSE by default.

Usage

halt(..., call. = FALSE)

Arguments

... arguments passed to stop
call. logical: print the call? Default is FALSE, unlike stop

Value

**loadLogfile**

*Read in a httpcache log file*

**Description**

Read in a httpcache log file

**Usage**

```
loadLogfile(filename, scope = c("CACHE", "HTTP"))
```

**Arguments**

- `filename` character name of the log file, passed to `utils::read.delim()`
- `scope` character optional means of selecting only certain log messages. By default, only "CACHE" and "HTTP" log messages are kept. Other logged messages, such as "ERROR" messages from `halt()`, will be dropped from the resulting data.frame.

**Value**

A data.frame of log results.

---

**logMessage**

*Log a message*

**Description**

Log a message

**Usage**

```
logMessage(...)```

**Arguments**

- `...` Strings to pass to `base::cat()`

**Value**

Nothing
requestLogSummary

Summarize HTTP requests from a log

Description
Summarize HTTP requests from a log

Usage
requestLogSummary(logdf)

Arguments
logdf A logging data.frame, as loaded by loadLogfile().

Value
A list containing counts of HTTP requests by verb, as well as summaries of time spent waiting on HTTP requests.

saveCache

Save and load cache state

Description
Warm your query cache from a previous session by saving out the cache and loading it back in.

Usage
saveCache(file)
loadCache(file)

Arguments
file character file path to write the cache data to, in .rds format

Value
Nothing; called for side effects.
**startLog**

Enable logging

**Description**

Enable logging

**Usage**

startLog(filename = "", append = FALSE)

**Arguments**

- filename
  - character: a filename/path where the log can be written out. If "" , messages will print to stdout (the screen). See base::cat().
- append
  - logical: if the file already exists, append to it? Default is FALSE, and if not in append mode, if the filename exists, it will be deleted.

**Value**

Nothing.

**uncached**

Context manager to temporarily turn cache off if it is on

**Description**

If you don’t want to store the response of a GET request in the cache, wrap it in uncached(). It will neither read from nor write to cache.

**Usage**

uncached(...)

**Arguments**

... Things to evaluate with caching off

**Details**

uncached will not invalidate cache records, if present. It only ignores them.

**Value**

Whatever ... returns.
Examples

uncached(GET("http://httpbin.org/get"))
Index

base::cat(), 7, 9
base::stop(), 6
buildCacheKey, 2

setCache (cache-api), 3
startLog, 9

uncached, 9
utils::read.delim(), 7

cache-api, 3
cache-management, 3
cached-http-verbs, 4
cachedPOST, 5
cachedPOST(), 4
cacheLogSummary, 5
cacheOff (cache-management), 3
cacheOn (cache-management), 3
clearCache (cache-management), 3

DELET E (cached-http-verbs), 4
dropCache, 6
dropCache(), 4
dropOnly (dropCache), 6
dropOnly(), 4
dropPattern (dropCache), 6

GET (cached-http-verbs), 4
getchache (cache-api), 3

halt, 6
halt(), 7
hitCache (cache-api), 3

loadCache (saveCache), 8
loadLogfile, 7
loadLogfile(), 5, 8
logMessage, 7

PATCH (cached-http-verbs), 4
POST (cached-http-verbs), 4
POST(), 5
PUT (cached-http-verbs), 4

requestLogSummary, 8

saveCache, 8