HTTP Requests for Users & Package Developers

Scott Chamberlain (@sckottie)
3 packages:
crul, webmockr, vcr

rOpenSci has a lot of pkgs that do http requests giving rise to the tools presented here
crul - a new http client

ropensci/crul
crul - features

- asynchronous requests
- pagination
- supports mocking and caching
- writing to disk + streaming
- request + response hooks
- does not have: OAuth
crul - lots of example usage

Reverse imports: bold, bbranching, ccafs, codemetar, erminer, discgolf, duckduckr, elastic, fulltext, geojsonlint, getlandsat, handlr, jaod, microdemic, nasapower, natsery, nsapi, openadds, originr, pangaeae, pleiades, postlightmercury, rbhl, rbison, rbraries, rcitoid, rcoreoa, rcrossref, rdatacite, rdpla, rdryad, rerddap, rgbif, rif, ritis, rjsonapi, rmoaa, ropenaq, rorcid, rphylogic, rplos, redlist, rsnps, rsunlight, rtimes, rvertnet, searoundus, sofa, solrium, spocc, taxize, tradestatistics, traits, vcr, webmockr, wikitaxa, worrms, zbank

Reverse suggests: fauxpas, finch
```r
con <- crul::HttpClient$new(url = "https://httpbin.org")
con$get(path = "get")

<crul response>
  url: https://httpbin.org/get
  request_headers:
    User-Agent: libcurl/7.54.0 r-curl/3.3 crul/0.7.4
    Accept-Encoding: gzip, deflate
    Accept: application/json, text/xml, application/xml, */*
  response_headers:
    status: HTTP/1.1 200 OK
    access-control-allow-credentials: true
    access-control-allow-origin: *
    content-encoding: gzip
    content-type: application/json
    date: Wed, 12 Jun 2019 23:21:09 GMT
    referrer-policy: no-referrer-when-downgrade
    server: nginx
    x-content-type-options: nosniff
    x-frame-options: DENY
    x-xss-protection: 1; mode=block
    content-length: 218
    connection: keep-alive
    status: 200
```

Returns an R6 object
crul demo

Index to results and methods with "$"

res$request
res$content
res$times
res$modified
res$response_headers_all
res$response_headers
res$request_headers
res$status_code
res$handle
res$opts
res$url
res$method
res$clone()
res$raise_for_status()
res$status_http()
res$success()
res$parse()
res$initialize()
res$print()
crul asynchronous

Same http options for every URL

```r
cc <- Async$new(
  urls = c(  
    'https://httpbin.org/get',
    'https://httpbin.org/get?a=5',
    'https://httpbin.org/get?foo=bar'
  )
)
res <- cc$get()
vapply(res, function(z) z$parse("UTF-8"), "")
#>   [1] "{\n  "args": {}, 
  "headers": {\n    "Accept": "application/json"
  }\n}  
#>   [2] "{\n  "args": {\n    "a": "5"
  }, 
  "headers": {\n    "Accept": "application/json"
  }\n}  
#>   [3] "{\n  "args": {\n    "foo": "bar"
  }, 
  "headers": {\n    "Accept": "application/json"
  }\n}
```

Async varied: custom http options for every request

```r
req1 <- HttpRequest$new("https://httpbin.org/get", headers = list(a="b"))$get()
req2 <- HttpRequest$new("https://httpbin.org/post")$post()
out <- AsyncVaried$new(req1, req2)
out$parse()
#>   [1] "{\n  "args": {}, 
  "headers": {\n    "Accept": "application/json"
  }\n}  
#>   [2] "{\n  "args": {}, 
  "data": "", 
  "files": {}, 
  "form": {\n    "a": "b"
  }\n}
```
Only supports pagination done via query parameters
Link headers and cursors to come
**crul request/response hooks**

- request hook: run *before* the request occurs
- response hook: run *once* the request is done

**request and response hooks example**

```r
fun_req <- function(request) {
  cat(paste0("Requesting: ", request$url$url, " at ", as.character(Sys.time())), sep = "\n")
}
fun_res <- function(response) {
  cat(paste0("status_code: ", response$status_code), sep = "\n")
}
x <- HttpClient$new(url = "https://httpbin.org", hooks = list(request = fun_req, response = fun_res))
invisible(x$get('get'))
#> Requesting: https://httpbin.org/get at 2019-07-06 02:10:38
#> status_code: 200
```
Mocking/caching

webmockr & vcr: forked from another language (Ruby)

we can take advantage of all they've learned

& both general purpose work with current and future http pkgs
Other langs

keep an eye out for other languages

what good ideas can we adopt in R land
webmockr - mock http requests

arose: because needed to make vcr

ropensci/webmockr
webmockr - what does it do?

set what you want to match against & what to return

make a request

if it matches you get what you set to return

if it doesn't match: error
webmockr - huh?

webmockr hooks into curl, hijacking the normal request.

constructing a response that matches a real response.

based on what you told webmockr to respond with.

& vcr builds on webmockr ...
library(crul)
library(webmockr)

stub_request("get", "https://httpbin.org/get") %>%
  wi_th(query = list(hello = "world")) %>% to_return(status = 418)

#> <webmockr stub>
#>   method: get
#>   uri: https://httpbin.org/get
#>   with:
#>     query: hello=world
#>     body:
#>     request_headers:
#>     to_return:
#>     status: 418
#>     body:
#>     response_headers:
#>     should_timeout: FALSE
#>     should_raise: FALSE

HttpClient$new()$get(path = 'get', query = list(hello = "world"))

#> <crul response>
#>   url: https://httpbin.org/get?hello=world
#>   request_headers:
#>     User-Agent: libcurl/7.54.0 r-curl/3.3 crul/0.7.0.9310
#>     Accept-Encoding: gzip, deflate
#>     Accept: application/json, text/xml, application/xml, */*
#>   response_headers:
#>   params:
#>   hello: world
#>   status: 418
library(httr)
GET("https://httpbin.org/get")
#> Error: Real HTTP connections are disabled.
#> Unregistered request:
#>   GET https://httpbin.org/get with headers
#>     {Accept: application/json, text/xml, application/xml, */*}
#>
#> You can stub this request with the following snippet:
#>
#>    stub_request('get', uri = 'https://httpbin.org/get') %>%
#>      with(
#>        headers = list(
#>          'Accept' = 'application/json, text/xml, application/xml, */*',
#>        )
#>      )
usage in the wild

```r
upload_file_job_json <- jsonlite::read_json("upload-file-job-2.json")
mockery::stub(upload_forecast, 'httr::upload_file', NULL)
stub_request('post', uri='http://example.com/api/model/1/forecasts/') %>%
to_return(
  body=upload_file_job_json,
  status=200,
  headers=list('Content-Type'='application/json; charset=utf-8')
)
```

src: [https://github.com/reichlab/zoltr](https://github.com/reichlab/zoltr)

```r
test_that('create_database works with mock', {
  stub_request("post", "https://api.treasuredata.com/v3/database/create/test") %>
to_return(body = "{}", status = 200)
  expect_true(create_database(conn, "test"))
})
```

src: [https://github.com/cran/RTD](https://github.com/cran/RTD)

Note - mocking requests with curl/httr inside of other fxns
expect failures?!

Expectation to timeout

```r
library(cruel)
library(webmockr)
cru::mock()

stub_request("get", "https://httpbin.org/get") %>% to_timeout()
x <- HttpClient$new(url = "https://httpbin.org")

x$get('get')
#> Error: Request Timeout (HTTP 408).
#> - The client did not produce a request within the time that the server
#>   was prepared to wait. The client MAY repeat the request without
#>   modifications at any later time.
```

Expectation to raise exception

```r
library(fauxpas)
stub_request("get", "https://httpbin.org/get") %>%
to_raise(fauxpas::HTTPBadGateway)

HttpClient$new(url = "https://httpbin.org")$get("get")
#> Error: Bad Gateway (HTTP 502).
#> - The server, while acting as a gateway or proxy, received an invalid
#>   response from the upstream server it accessed in attempting to
#>   fulfill the request.
```
vcr - record and replay HTTP requests/responses

arose: observing other language communities & need to improve testing in many API clients

ropensci/vcr
vcr - hardest software project I've worked on
vcr - hardest software project I've worked on

Ruby

def has_interaction_matching?(request)
  !!matching_interaction_index_for(request) ||
  !!matching_used_interaction_for(request) ||
  @parent_list.has_interaction_matching?(request)
end

R

has_interaction_matching = function(request) {
  private$matching_interaction_bool(request) ||
  private$matching_used_interaction_for(request) ||
  self$parent_list$has_interaction_matching()
}
vcr - no monkey patching in R!

Allowed in Ruby, but not in R

in R we can do

```r
assignInNamespace("some_object", value = function(e) e, ns = "some_other_pkg")
```

But not allowed on CRAN
vcr - how does it work?
vcr - how does it work?

I thought vcr worked by listening for requests in R

realized it most definitely did not

it modifies an HTTP request & looks for a match

so had to make webmockr first
vcr - what does it do?

HTTP requests in a test suite as usual

w/o making real HTTP requests

so you test your package

not the remote service

(p.s. great for rate-limited services)
what is a cassette?

http_interactions:
- request:
  method: get
  uri: http://www.marinespecies.org/rest/AphiaExternalIDByAphiaID/1080?type=tsn
  body:
    encoding: ''
    string: ''
  headers:
    User-Agent: libcurl/7.54.0 r-curl/3.3 crul/0.8.0
    Accept-Encoding: gzip, deflate
    Accept: application/json, text/xml, application/xml, */*
response:
status:
  status_code: '200'
  message: OK
  explanation: Request fulfilled, document follows
headers:
  status: HTTP/1.1 200 OK
  date: Fri, 28 Jun 2019 16:55:51 GMT
  server: Apache/2.4.25 (Win32) PHP/5.6.29
  x-powered-by: PHP/5.6.29
  access-control-allow-origin: '*'
  access-control-allow-headers: X-Requested-With, Content-Type, Accept, Origin, Authorization
  access-control-allow-methods: GET, POST, OPTIONS
  content-length: '9'
  content-type: application/json
body:
  encoding: UTF-8
  string: '[[85257]]'
recorded_at: 2019-06-28 16:55:51 GMT
vcr - a brief example

```r
library(vcr)
library(crul)

cli <- crul::HttpClient$new(url = "https://api.crossref.org")
use_cassette(name = "helloworld", {
  res1 <- cli$get("works", query = list(rows = 3))
})

use_cassette(name = "helloworld", {
  res2 <- cli$get("works", query = list(rows = 3))
})

identical(res1$parse(), res2$parse())
#> [1] TRUE
```
speeds up your tests

w/o vcr

```r
Rscript -e 'devtools::test()
Testing worrms

<table>
<thead>
<tr>
<th>OK</th>
<th>F</th>
<th>W</th>
<th>S</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td>wm_children [10.0 s]</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>wm_classification [1.4 s]</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

--- Results
Duration: 141.0 s
```

w/ vcr

```r
Rscript -e 'devtools::test()
Testing worrms

<table>
<thead>
<tr>
<th>OK</th>
<th>F</th>
<th>W</th>
<th>S</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td>wm_children [3.8 s]</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>wm_classification [0.5 s]</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

--- Results
Duration: 35.6 s
```
vcr - in the works

- JSON cassettes
- testthat reporter for cassette usage
- dates 🕒MONTH📅
- data security 💠, always more to do
- responses written to disk
- many more
further reading

HTTP Testing Book: bit.ly/http-testing
crul/webmockr/vcr in detail w/ caveats/edge cases/etc.
slides: scotttalks.info/user-http
Made w/: reveal.js v3.7.0, FontAwesome v5.7.2