Package ‘rvest’

July 25, 2020

Title Easily Harvest (Scrape) Web Pages

Version 0.3.6

Description Wrappers around the 'xml2' and 'httr' packages to make it easy to download, then manipulate, HTML and XML.

License GPL-3

URL `http://rvest.tidyverse.org/`, `https://github.com/tidyverse/rvest`

BugReports `https://github.com/tidyverse/rvest/issues`

Depends R (>= 3.2), xml2

Imports httr (>= 0.5), magrittr, selectr

Suggests covr, knitr, png, rmarkdown, spelling, stringi (>= 0.3.1), testthat

VignetteBuilder knitr

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.1.1

NeedsCompilation no

Author Hadley Wickham [aut, cre],
RStudio [cph]

Maintainer Hadley Wickham `<hadley@rstudio.com>`

Repository CRAN

Date/Publication 2020-07-25 21:50:02 UTC

R topics documented:

encoding *Guess and repair faulty character encoding.*

Description

These functions help you respond to web pages that declare incorrect encodings. You can use `guess_encoding` to figure out what the real encoding is (and then supply that to the `encoding` argument of html), or use `repair_encoding` to fix character vectors after the fact.

Usage

```
guess_encoding(x)

repair_encoding(x, from = NULL)
```

Arguments

`x`	A character vector.
`from`	The encoding that the string is actually in. If `NULL`, `guess_encoding` will be used.

stringi

These function are wrappers around tools from the fantastic stringi package, so you'll need to make sure to have that installed.

Examples

```
# A file with bad encoding included in the package
path <- system.file("html-ex", "bad-encoding.html", package = "rvest")
x <- read_html(path)
x %>% html_nodes("p") %>% html_text()

guess_encoding(x)
# Two valid encodings, only one of which is correct
read_html(path, encoding = "ISO-8859-1") %>% html_nodes("p") %>% html_text()
read_html(path, encoding = "ISO-8859-2") %>% html_nodes("p") %>% html_text()
```

`google_form` *Make link to google form given id*

Description

Make link to google form given id

Usage

```
google_form(x)
```

Arguments

`x`	Unique identifier for form

Examples

```
google_form("1M9B8DsYNFyDjpwSK6ur_bZf8Rv_04ma3rmaaBiveoUI")
```

`html_form` *Parse forms in a page.*

Description

Parse forms in a page.

Usage

```
html_form(x)
```

Arguments

`x`	A node, node set or document.

See Also

HTML 4.01 form specification: http://www.w3.org/TR/html401/interact/forms.html

Examples

```

html_form(read_html("https://hadley.wufoo.com/forms/libraryrequire-quiz/"))
html_form(read_html("https://hadley.wufoo.com/forms/r-journal-submission/"))

box_office <- read_html("http://www.boxofficemojo.com/movies/?id=ateam.htm")
box_office %>% html_node("form") %>% html_form()
```

html_nodes *Select nodes from an HTML document*

Description

More easily extract pieces out of HTML documents using XPath and CSS selectors. CSS selectors are particularly useful in conjunction with http://selectorgadget.com/: it makes it easy to find exactly which selector you should be using. If you haven't used CSS selectors before, work your way through the fun tutorial at http://flukeout.github.io/

Usage

```
html_nodes(x, css, xpath)

html_node(x, css, xpath)
```

Arguments

x	Either a document, a node set or a single node.
css, xpath	Nodes to select. Supply one of css or xpath depending on whether you want to use a CSS or XPath 1.0 selector.

html_node vs html_nodes

html_node is like [[it always extracts exactly one element. When given a list of nodes, html_node will always return a list of the same length, the length of html_nodes might be longer or shorter.

CSS selector support

CSS selectors are translated to XPath selectors by the **selectr** package, which is a port of the python **cssselect** library, https://pythonhosted.org/cssselect/.

It implements the majority of CSS3 selectors, as described in http://www.w3.org/TR/2011/REC-css3-selectors-20110929/. The exceptions are listed below:

- Pseudo selectors that require interactivity are ignored: :hover, :active, :focus, :target, :visited
- The following pseudo classes don't work with the wild card element, *: *:first-of-type, *:last-of-type, *:nth-of-type, *:nth-last-of-type, *:only-of-type
- It supports :contains(text)
- You can use !=, [foo!=bar] is the same as :not([foo=bar])
- :not() accepts a sequence of simple selectors, not just single simple selector.

Examples

```
# CSS selectors ----------------------------------------------
url <- paste0(
  "https://web.archive.org/web/20190202054736/",
  "https://www.boxofficemojo.com/movies/?id=ateam.htm"
)
ateam <- read_html(url)
html_nodes(ateam, "center")
html_nodes(ateam, "center font")
html_nodes(ateam, "center font b")

# But html_node is best used in conjunction with %>% from magrittr
# You can chain subsetting:
ateam %>% html_nodes("center") %>% html_nodes("td")
ateam %>% html_nodes("center") %>% html_nodes("font")

td <- ateam %>% html_nodes("center") %>% html_nodes("td")
td
# When applied to a list of nodes, html_nodes() returns all nodes,
# collapsing results into a new nodelist.
td %>% html_nodes("font")
# html_node() returns the first matching node. If there are no matching
# nodes, it returns a "missing" node
if (utils::packageVersion("xml2") > "0.1.2") {
  td %>% html_node("font")
}

# To pick out an element at specified position, use magrittr::extract2
# which is an alias for [[
library(magrittr)
ateam %>% html_nodes("table") %>% extract2(1) %>% html_nodes("img")
ateam %>% html_nodes("table") %>% `[[`(1) %>% html_nodes("img")

# Find all images contained in the first two tables
ateam %>% html_nodes("table") %>% `[`(1:2) %>% html_nodes("img")
ateam %>% html_nodes("table") %>% extract(1:2) %>% html_nodes("img")

# XPath selectors ---------------------------------------------
# chaining with XPath is a little trickier - you may need to vary
# the prefix you're using - // always selects from the root node
# regardless of where you currently are in the doc
ateam %>%
  html_nodes(xpath = "//center//font//b") %>%
  html_nodes(xpath = "//b")
```

html_session *Simulate a session in an html browser.*

Description

Simulate a session in an html browser.

Usage

```
html_session(url, ...)

is.session(x)
```

Arguments

url	Location to start session
...	Any additional httr config to use throughout session.
x	An object to test to see if it's a session.

Methods

A session object responds to a combination of httr and html methods: use `httr::cookies()`, `httr::headers()`, and `httr::status_code()` to access properties of the request; and `html_nodes()` to access the html.

Examples

```
# http://stackoverflow.com/questions/15853204

s <- html_session("http://hadley.nz")
s %>% jump_to("hadley-wickham.jpg") %>% jump_to("/") %>% session_history()
s %>% jump_to("hadley-wickham.jpg") %>% back() %>% session_history()

s %>% follow_link(css = "p a")
```

html_table *Parse an html table into a data frame.*

Description

Parse an html table into a data frame.

Usage

```
html_table(x, header = NA, trim = TRUE, fill = FALSE, dec = ".")
```

Arguments

x	A node, node set or document.
header	Use first row as header? If NA, will use first row if it consists of <th> tags.
trim	Remove leading and trailing whitespace within each cell?
fill	If TRUE, automatically fill rows with fewer than the maximum number of columns with NAs.
dec	The character used as decimal mark.

Assumptions

`html_table` currently makes a few assumptions:

- No cells span multiple rows
- Headers are in the first row

Examples

```
sample1 <- minimal_html("<table>
  <tr><th>Col A</th><th>Col B</th></tr>
  <tr><td>1</td><td>x</td></tr>
  <tr><td>4</td><td>y</td></tr>
  <tr><td>10</td><td>z</td></tr>
</table>")
sample1 %>%
  html_node("table") %>%
  html_table()

# Values in merged cells will be duplicated
sample2 <- minimal_html("<table>
  <tr><th>A</th><th>B</th><th>C</th></tr>
  <tr><td>1</td><td>2</td><td>3</td></tr>
  <tr><td colspan='2'>4</td><td>5</td></tr>
  <tr><td>6</td><td colspan='2'>7</td></tr>
</table>")

sample2 %>%
  html_node("table") %>%
  html_table()

# If the table is badly formed, and has different number of columns
# in each row, use `fill = TRUE` to fill in the missing values
sample3 <- minimal_html("<table>
  <tr><th>A</th><th>B</th><th>C</th></tr>
  <tr><td colspan='2'>1</td><td>2</td></tr>
  <tr><td colspan='2'>3</td></tr>
  <tr><td>4</td></tr>
</table>")

sample3 %>%
  html_node("table") %>%
  html_table(fill = TRUE)
```

`html_text` *Extract attributes, text and tag name from html.*

Description

Extract attributes, text and tag name from html.

Usage

```
html_text(x, trim = FALSE)

html_name(x)

html_children(x)

html_attrs(x)

html_attr(x, name, default = NA_character_)
```

Arguments

`x` A document, node, or node set.

`trim` If `TRUE` will trim leading and trailing spaces.

`name` Name of attribute to retrieve.

`default` A string used as a default value when the attribute does not exist in every node.

Value

`html_attr`, `html_tag` and `html_text`, a character vector; `html_attrs`, a list.

Examples

```
movie <- read_html("https://en.wikipedia.org/wiki/The_Lego_Movie")
cast <- html_nodes(movie, "tr:nth-child(8) .plainlist a")
html_text(cast)
html_name(cast)
html_attrs(cast)
html_attr(cast, "href")
```

jump_to *Navigate to a new url.*

Description

`jump_to()` takes a url (either relative or absolute); `follow_link` takes an expression that refers to a link (an <a> tag) on the current page.

Usage

```
jump_to(x, url, ...)

follow_link(x, i, css, xpath, ...)
```

Arguments

x A session.

url A URL, either relative or absolute, to navigate to.

... Any additional httr configs to apply to this request.

i You can select with:

an integer selects the ith link

a string first link containing that text (case sensitive)

css Nodes to select. Supply one of `css` or `xpath` depending on whether you want to use a CSS or XPath 1.0 selector.

xpath Nodes to select. Supply one of `css` or `xpath` depending on whether you want to use a CSS or XPath 1.0 selector.

Examples

```
s <- html_session("http://hadley.nz")
s <- s %>% follow_link("github")
s <- s %>% back()
s %>% follow_link("readr")
```

pluck *Extract elements of a list by position.*

Description

Extract elements of a list by position.

Usage

```
pluck(x, i, type)
```

Arguments

x A list

i A string or integer.

type Type of output, if known

session_history *History navigation tools*

Description

History navigation tools

Usage

```
session_history(x)

back(x)
```

Arguments

x	A session.

set_values *Set values in a form.*

Description

Set values in a form.

Usage

```
set_values(form, ...)
```

Arguments

form	Form to modify
...	Name-value pairs giving fields to modify

Value

An updated form object

Examples

```
search <- html_form(read_html("http://www.google.com"))[[1]]
set_values(search, q = "My little pony")
set_values(search, hl = "fr")
## Not run: set_values(search, btnI = "blah")
```

`submit_form` *Submit a form back to the server.*

Description

Submit a form back to the server.

Usage

```
submit_form(session, form, submit = NULL, ...)
```

Arguments

`session`	Session to submit form to.
`form`	Form to submit
`submit`	Name of submit button to use. If not supplied, defaults to first submission button on the form (with a message).
`...`	Additional arguments passed on to `httr::GET()` or `httr::POST()`

Value

If successful, the parsed html response. Throws an error if http request fails.

Index