Package 'rvest'

February 20, 2015

Version 0.2.0

Title Easily Harvest (Scrape) Web Pages

Description Wrappers around the XML and httr packages to make it easy to download, then manipulate, both html and xml.

Depends R (>= 3.0.1)

Imports httr (>= 0.5), XML, selectr, magrittr

Suggests testthat, knitr, png, stringi (>= 0.3.1)

License GPL-3

LazyData true

VignetteBuilder knitr

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

Maintainer Hadley Wickham <h.wickham@gmail.com>

NeedsCompilation no

Repository CRAN

Date/Publication 2015-01-01 19:49:23

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`,

stringi

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

Examples

```
# This page claims to be in iso-8859-1:
url <- 'http://www.elections.ca/content.aspx?section=res&dir=cir/list&document=index&lang=e#list'
elections <- html(url)
x <- elections %>% html_nodes("table") %>% .[[2]] %>% html_table() %>% .$TO
# But something looks wrong:
x

# It's acutally UTF-8!
guess_encoding(x)

# We can repair this vector:
repair_encoding(x)

# But it's better to start from scratch with correctly encoded file
elections <- html(url, encoding = "UTF-8")
```

```
elections %>% html_nodes("table") %>% .[[2]] %>% html_table() %>% .$TO
```

`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` *Parse an HTML page.*

Description

Parse an HTML page.

Usage

```
html(x, ..., encoding = NULL)
```

Arguments

x	A url, a local path, a string containing html, or a response from an httr request.
...	If x is a URL, additional arguments are passed on to `GET()`.
encoding	Specify encoding of document. See `iconvlist()` for complete list. If you have problems determining the correct encoding, try `stri_enc_detect`

Examples

```
# From a url:
google <- html("http://google.com")
google %>% xml_structure()
google %>% html_nodes("p")

# From a string: (minimal html 5 document)
# http://www.brucelawson.co.uk/2010/a-minimal-html5-document/
minimal <- html("<!doctype html>
  <meta charset=utf-8>
 <title>blah</title>
 <p>I'm the content")
minimal

# From an httr request
google2 <- html(httr::GET("http://google.com"))
```

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

box_office <- 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 have'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 complete document (XMLInternalDocument), a list of tags (XMLNodeSet) or a single tag (XMLInternalElementNode).
`css,xpath`	Nodes to select. Supply one of `css` or `xpath` depending on whether you want to use a css or xpath 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 ------------------------------------------------
ateam <- html("http://www.boxofficemojo.com/movies/?id=ateam.htm")
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")

# When applied to a list of nodes, html_nodes() collapses output
# html_node() selects a single element from each
td <- ateam %>% html_nodes("center") %>% html_nodes("td")
td %>% html_nodes("font")
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 noot
# 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 cookies(), headers(), and 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://had.co.nz")
s %>% jump_to("thesis") %>% jump_to("/") %>% session_history()
s %>% jump_to("thesis") %>% 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

```
tdist <- html("http://en.wikipedia.org/wiki/Student%27s_t-distribution")
tdist %>%
  html_node("table.infobox") %>%
  html_table(header = FALSE)

births <- html("http://www.ssa.gov/oact/babynames/numberUSbirths.html")
html_table(html_nodes(births, "table")[[2]])

# If the table is badly formed, and has different number of rows in
# each column use fill = TRUE. Here's it's due to incorrect colspan
# specification.
skiing <- html("http://data.fis-ski.com/dynamic/results.html?sector=CC&raceid=22395")
skiing %>%
  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, ...)

html_tag(x)

html_children(x)

html_attrs(x)

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

Arguments

`x`	Either a complete document (XMLInternalDocument), a list of tags (XMLNodeSet) or a single tag (XMLInternalElementNode).
`...`	Other arguments passed onto `xmlValue()`. The most useful argument is `trim = TRUE` which will remove leading and trailing whitespace.
`name`	Name of attribute to extract.
`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 <- html("http://www.imdb.com/title/tt1490017/")
cast <- html_nodes(movie, "#titleCast span.itemprop")
html_text(cast)
html_tag(cast)
html_attrs(cast)
html_attr(cast, "class")
html_attr(cast, "itemprop")

basic <- html("<p class='a'><b>Bold text</b></p>")
p <- html_node(basic, "p")
p
# Can subset with numbers to extract children
p[[1]]
# Use html_attr to get attributes
html_attr(p, "class")
```

---

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

Examples

```
s <- html_session("http://had.co.nz")
s %>% jump_to("thesis/")
s %>% follow_link("vita")
s %>% follow_link(3)
s %>% follow_link("vita")
```

`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

`scrape` *scrape.*

Description

scrape.

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(html("https://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 GET() or POST()

Value

If successful, the parsed html response. Throws an error if http request fails. To access other elements of response, construct it yourself using the elements returned by submit_request.

Examples

```
test <- google_form("1M9B8DsYNFyDjpwSK6ur_bZf8Rv_04ma3rmaaBiveoUI")
f0 <- html_form(test)[[1]]
f1 <- set_values(f0, entry.564397473 = "abc")
```

xml *Work with xml.*

Description

All methods work the same as their HTML equivalents. Currently xml parses XML files as HTML because I can't find another way to ignore namespaces.

Usage

```
xml(x, ..., encoding = NULL)

xml_tag(x)

xml_attr(x, name, default = NA_character_)

xml_attrs(x)

xml_node(x, css, xpath)

xml_nodes(x, css, xpath)

xml_text(x, ...)

xml_children(x)
```

Arguments

x	A url, a local path, a string containing html, or a response from an httr request.
...	If x is a URL, additional arguments are passed on to GET().
encoding	Specify encoding of document. See iconvlist() for complete list. If you have problems determining the correct encoding, try stri_enc_detect
name	Name of attribute to extract.
default	A string used as a default value when the attribute does not exist in every node.
css	Nodes to select. Supply one of css or xpath depending on whether you want to use a css or xpath selector.
xpath	Nodes to select. Supply one of css or xpath depending on whether you want to use a css or xpath selector.

Examples

```
search <- xml("http://stackoverflow.com/feeds")

entries <- search %>% xml_nodes("entry")
entries[[1]] %>% xml_structure()

entries %>% xml_node("author name") %>% xml_text()
entries %>% lapply(. %>% xml_nodes("category") %>% xml_attr("term"))
```

---

xml_structure *Show the structure of an html/xml document.*

Description

Show the structure of an html/xml document without displaying any of the values. This is useful if you want to get a high level view of the way a document is organised.

Usage

```
xml_structure(x, indent = 0)
```

Arguments

x	HTML/XML document (or part there of)
indent	Number of spaces to ident

Index