Package 'DT'

June 9, 2015

Type Package

Title A Wrapper of the JavaScript Library 'DataTables'

Version 0.1

Date 2015-06-08

Maintainer Yihui Xie <xie@yihui.name>

Description Data objects in R can be rendered as HTML tables using the JavaScript library 'DataTables' (typically via R Markdown or Shiny). The 'DataTables' library has been included in this R package. The package name 'DT' is an abbreviation of 'DataTables'.

URL http://rstudio.github.io/DT

BugReports https://github.com/rstudio/DT/issues

License GPL-3 | file LICENSE

Imports htmltools, htmlwidgets (>= 0.4), magrittr

Suggests jsonlite (>= 0.9.16), knitr (>= 1.8), shiny (>= 0.12.0)

VignetteBuilder knitr

NeedsCompilation no

Author Yihui Xie [aut, cre],
Joe Cheng [ctb],
jQuery contributors [ctb, cph] (jQuery in htmlwidgets/lib),
SpryMedia Limited [ctb, cph] (DataTables in htmlwidgets/lib),
Brian Reavis [ctb, cph] (selectize.js in htmlwidgets/lib),
Leon Gersen [ctb, cph] (noUiSlider in htmlwidgets/lib),
Bartek Szopka [ctb, cph] (jquery.highlight.js in htmlwidgets/lib),
RStudio Inc [cph]

Repository CRAN

Date/Publication 2015-06-09 07:47:04

R topics documented:

`copySWF` *Copy the Flash SWF file from the TableTools extension*

Description

This is a convenience function to copy the SWF file since the TableTools extension depends on it.

Usage

```
copySWF(dest = ".", pdf = FALSE)
```

Arguments

`dest`	the destination directory
`pdf`	`TRUE` if you want to save the table as PDF ('`copy_csv_xls_pdf.swf`' will be copied); `FALSE` otherwise (use '`copy_csv_xls.swf`')

Value

A character string of the path of the SWF file, which may be used as the `sSwfPath` option for TableTools.

References

http://datatables.net/extensions/tabletools

datatable *Create an HTML table widget using the DataTables library*

Description

This function creates an HTML widget to display rectangular data (a matrix or data frame) using the JavaScript library DataTables.

Usage

```
datatable(data, options = list(), class = "display", callback = JS("return table;"),
  rownames, colnames, container, caption = NULL, filter = c("none", "bottom", "top"),
  escape = TRUE, style = "default", selection = c("multiple", "single", "none"),
   extensions = list())
```

Arguments

data	a data object (either a matrix or a data frame)
options	a list of initialization options (see http://datatables.net/reference/option/); the character options wrapped in JS() will be treated as literal JavaScript code instead of normal character strings; you can also set options globally via options(DT.options = list(. and global options will be merged into this options argument if set
class	the CSS class(es) of the table; see http://datatables.net/manual/styling/classes
callback	the body of a JavaScript callback function with the argument table to be applied to the DataTables instance (i.e. table)
rownames	TRUE (show row names) or FALSE (hide row names) or a character vector of row names; by default, the row names are displayed in the first column of the table if exist (not NULL)
colnames	if missing, the column names of the data; otherwise it can be an unnamed character vector of names you want to show in the table header instead of the default data column names; alternatively, you can provide a *named* numeric or character vector of the form 'newName1' = i1,'newName2' = i2 or c('newName1' = 'oldName1', 'newName2' = 'oldName2', ...), where newName is the new name you want to show in the table, and i or oldName is the index of the current column name
container	a sketch of the HTML table to be filled with data cells; by default, it is generated from htmltools::tags$table() with a table header consisting of the column names of the data
caption	the table caption; a character vector or a tag object generated from htmltools::tags$caption()
filter	whether/where to use column filters; none: no filters; bottom/top: put column filters at the bottom/top of the table; range sliders are used to filter numeric/date/time columns, select lists are used for factor columns, and text input boxes are used for character columns; if you want more control over the styles of

	filters, you can provide a list to this argument of the form `list(position = 'top', clear = TRUE, pla = FALSE)`, where `clear` indicates whether you want the clear buttons in the input boxes, and `plain` means if you want to use Bootstrap form styles or plain text input styles for the text input boxes
`escape`	whether to escape HTML entities in the table: `TRUE` means to escape the whole table, and `FALSE` means not to escape it; alternatively, you can specify numeric column indices or column names to indicate which columns to escape, e.g. `1:5` (the first 5 columns), `c(1, 3, 4)`, or `c(-1, -3)` (all columns except the first and third), or `c('Species', 'Sepal.Length')`
`style`	the style name (http://datatables.net/manual/styling/); currently only `'default'` and `'bootstrap'` are supported
`selection`	the row selection mode (single or multiple selection or disable selection) when a table widget is rendered in a Shiny app
`extensions`	a character vector of the names of the DataTables extensions (http://datatables.net/extensions/index), or a named list of initialization options for the extensions (the names of the list are the names of extensions)

Note

You are recommended to escape the table content for security reasons (e.g. XSS attacks) when using this function in Shiny or any other dynamic web applications.

References

See http://rstudio.github.io/DT for the full documentation.

Examples

```
library(DT)

# see the package vignette for examples and the link to website
vignette('DT', package = 'DT')

# some boring edge cases for testing purposes
m = matrix(nrow = 0, ncol = 5, dimnames = list(NULL, letters[1:5]))
datatable(m)  # zero rows
datatable(as.data.frame(m))

m = matrix(1, dimnames = list(NULL, 'a'))
datatable(m)  # one row and one column
datatable(as.data.frame(m))

m = data.frame(a = 1, b = 2, c = 3)
datatable(m)
datatable(as.matrix(m))

# dates
datatable(data.frame(
  date = seq(as.Date("2015-01-01"), by = "day", length.out = 5), x = 1:5
))
```

```
datatable(data.frame(x = Sys.Date()))
datatable(data.frame(x = Sys.time()))
```

`dataTableAjax` *Register a data object in a shiny session for DataTables*

Description

This function stores a data object in a shiny session and returns a URL that returns JSON data based on DataTables Ajax requests. The URL can be used as the `url` option inside the `ajax` option of the table. It is basically an implementation of server-side processing of DataTables in R. Filtering, sorting, and pagination are processed through R instead of JavaScript (client-side processing).

Usage

```
dataTableAjax(session, data, rownames, filter = dataTablesFilter)
```

Arguments

`session`	the `session` object in the shiny server function (`function(input, output, session)`)
`data`	a data object (will be coerced to a data frame internally)
`rownames`	see `datatable()`; it must be consistent with what you use in `datatable()`, e.g. if the widget is generated by `datatable(rownames = FALSE)`, you must also use `dataTableAjax(rownames = FALSE)` here
`filter`	(for expert use only) a function with two arguments `data` and `params` (Ajax parameters, a list of the form `list(search = list(value = 'FOO', regex = 'false'), length = 10,` that return the filtered table result according to the DataTables Ajax request

Details

Normally you should not need to call this function directly. It is called internally when a table widget is rendered in a Shiny app to configure the table option `ajax` automatically. If you are familiar with **DataTables**' server-side processing, and want to use a custom filter function, you may call this function to get an Ajax URL.

Value

A character string (an Ajax URL that can be queried by DataTables).

References

http://rstudio.github.io/DT/server.html

Examples

```
DTApp = function(data, ..., options = list()) {
  library(shiny)
  library(DT)
  shinyApp(
    ui = fluidPage(
      title = 'Server-side processing of DataTables',
      fluidRow(
        DT::dataTableOutput('tbl')
      )
    ),
    server = function(input, output, session) {
      options$serverSide = TRUE
      options$ajax = list(url = dataTableAjax(session, data))
      # create a widget using an Ajax URL created above
      widget = datatable(data, ..., options = options)
      output$tbl = DT::renderDataTable(widget)
    }
  )
}

if (interactive()) DTApp(iris)
if (interactive()) DTApp(iris, filter = 'top')
```

`dataTableOutput` *Helper functions for using DT in Shiny*

Description

These two functions are like most `fooOutput()` and `renderFoo()` functions in the **shiny** package. The former is used to create a container for table, and the latter is used in the server logic to render the table.

Usage

```
dataTableOutput(outputId, width = "100%", height = "auto")

renderDataTable(expr, server = TRUE, env = parent.frame(), quoted = FALSE, ...)
```

Arguments

`outputId`	output variable to read the table from
`width`	the width of the table container
`height`	the height of the table container
`expr`	an expression to create a table widget (normally via `datatable()`), or a data object to be passed to `datatable()` to create a table widget

server	whether to use server-side processing. If TRUE, then the data is kept on the server and the browser requests a page at a time; if FALSE, then the entire data frame is sent to the browser at once. Highly recommended for medium to large data frames, which can cause browsers to slow down or crash.
env	The environment in which to evaluate expr.
quoted	Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable.
...	ignored when expr returns a table widget, and passed as additional arguments to datatable() when expr returns a data object

References

http://rstudio.github.io/DT/shiny.html

Examples

```
if (interactive()) {
  library(shiny)
  shinyApp(
    ui = fluidPage(fluidRow(column(12, DT::dataTableOutput('tbl')))),
    server = function(input, output) {
      output$tbl = DT::renderDataTable(
        iris, options = list(lengthChange = FALSE)
      )
    }
  )
}
```

DT-imports *Objects imported from other packages*

Description

These objects are imported from other packages. Follow the links to their documentation.

htmlwidgets JS, saveWidget

magrittr %>%

`formatCurrency` *Format table columns*

Description

Format numeric columns in a table as currency (`formatCurrency()`) or percentages (`formatPercentage()`), or round numbers to a specified number of decimal places (`formatRound()`). The function `formatStyle()` applies CSS styles to table cells by column.

Usage

```
formatCurrency(table, columns, currency = "$", interval = 3, mark = ",")

formatPercentage(table, columns, digits = 0)

formatRound(table, columns, digits = 2)

formatDate(table, columns, method = "toDateString")

formatStyle(table, columns, fontWeight = NULL, color = NULL, backgroundColor = NULL,
    background = NULL, ...)
```

Arguments

`table`	a table object created from `datatable()`
`columns`	the indices of the columns to be formatted (can be character, numeric, logical, or a formula of the form `~ V1 + V2`, which is equivalent to `c('V1', 'V2')`)
`currency`	the currency symbol
`interval`	put a marker after how many digits of the numbers
`mark`	the marker after every `interval` decimals in the numbers
`digits`	the number of decimal places to round to
`method`	the method(s) to convert a date to string in JavaScript; see `DT:::DateMethods` for a list of possible methods, and `http://mzl.la/1xGe99W` for a full reference
`fontWeight`	the font weight, e.g. `'bold'` and `'normal'`
`color`	the font color, e.g. `'red'` and `'#ee00aa'`
`backgroundColor`	the background color of table cells
`background`	the background of table cells
`...`	other CSS properties, e.g. `'border'`, `'font-size'`, `'text-align'`, and so on; if you want to condition CSS styles on the cell values, you may use the helper functions such as `styleInterval()`; note the actual CSS property names are dash-separated, but you can use camelCase names in this function (otherwise you will have to use backticks to quote the names, e.g. `` `font-size` = '12px' ``), and this function will automatically convert camelCase names to dash-separated names (e.g. `'fontWeight'` will be converted to `'font-weight'` internally)

References

See http://rstudio.github.io/DT/functions.html for detailed documentation and examples.

Examples

```
library(DT)
m = cbind(matrix(rnorm(120, 1e5, 1e6), 40), runif(40), rnorm(40, 100))
colnames(m) = head(LETTERS, ncol(m))
m

# format the columns A and C as currency, and D as percentages
datatable(m) %>% formatCurrency(c('A', 'C')) %>% formatPercentage('D', 2)

# the first two columns are Euro currency, and round column E to 3 decimal places
datatable(m) %>% formatCurrency(1:2, '\U20AC') %>% formatRound('E', 3)

# apply CSS styles to columns
datatable(iris) %>%
  formatStyle('Sepal.Length', fontWeight = styleInterval(5, c('bold', 'weight'))) %>%
  formatStyle('Sepal.Width',
    color = styleInterval(3.4, c('red', 'white')),
    backgroundColor = styleInterval(3.4, c('yellow', 'gray'))
  )
```

---

styleInterval *Conditional CSS styles*

Description

A few helper functions for the formatStyle() function to calculate CSS styles for table cells based on the cell values. Under the hood, they just generate JavaScript and CSS code from the values specified in R.

Usage

```
styleInterval(cuts, values)

styleEqual(levels, values)

styleColorBar(data, color)
```

Arguments

cuts	a vector of cut points (sorted increasingly)
values	a vector of CSS values
levels	a character vector of data values to be mapped (one-to-one) to CSS values
data	the numeric vector to be represented as color bars (in fact, only its range, i.e. min and max, is needed here)
color	the color of the bars

Details

The function `styleInterval()` maps intervals to CSS values. Its argument `values` must be of length `n + 1` where `n = length(cuts)`. The right-closed interval '`(cuts[i - 1], cuts[i]]`' is mapped to '`values[i]`' for '`i = 2, 3, ..., n`'; '`values[1]`' is for the interval '`(-Inf, cuts[1]]`', and '`values[n + 1]`' is for '`(cuts[n], +Inf)`'. You can think of the order of `cuts` and `values` using this diagram: '`-Inf -> values[1] -> cuts[1] -> values[2] -> cuts[2] -> ... -> values[n] -> cuts[n] -> v +Inf`'.

The function `styleEqual()` maps data values to CSS values in the one-to-one manner, i.e. `values[i]` is used when the table cell value is `levels[i]`.

The function `styleColorBar()` can be used to draw background color bars behind table cells in a column, and the width of bars is proportional to the column values.

---

tableHeader *Generate a table header or footer from column names*

Description

Convenience functions to generate a table header ('`<thead></thead>`') or footer ('`<tfoot></tfoot>`') given the column names. They are basically wrappers of `htmltools::tags$th` applied to the column names.

Usage

```
tableHeader(names, escape = TRUE)

tableFooter(names, escape = TRUE)
```

Arguments

`names`	a character vector of the column names of the table (if it is an object with column names, its column names will be used instead)
`escape`	whether to escape the names (see `datatable`)

Value

A tag object generated by `htmltools::tags`.

Examples

```
library(DT)
tableHeader(iris)  # or equivalently,
tableHeader(colnames(iris))
tableFooter(iris)  # footer

library(htmltools)
tags$table(tableHeader(iris), tableFooter(iris))
```

Index