Package 'shinyjqui'

October 14, 2022

Type Package

Title 'jQuery UI' Interactions and Effects for Shiny

Version 0.4.1

Maintainer Yang Tang <tang_yang@outlook.com>

Description An extension to shiny that brings interactions and animation effects from 'jQuery UI' library.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 3.2.0)

Imports shiny (>= 1.5.0), htmltools, htmlwidgets, jsonlite, rlang

Suggests ggplot2, highcharter, knitr, markdown, rmarkdown, plotly

URL https://github.com/yang-tang/shinyjqui, https://yang-tang.github.io/shinyjqui/

BugReports https://github.com/yang-tang/shinyjqui/issues

RoxygenNote 7.1.2

VignetteBuilder knitr

NeedsCompilation no

Author Yang Tang [aut, cre]

Repository CRAN

Date/Publication 2022-02-03 07:00:02 UTC

R topics documented:

`Animation_effects` *Animation effects.*

Description

Allow element(s) to show animation effects.

- `jqui_effect()`: Apply an animation effect to matched element(s).
- `jqui_hide()`: Hide the matched element(s) with animation effect.
- `jqui_show()`: Display the matched element(s) with animation effect.
- `jqui_toggle()`: Display or hide the matched element(s) with animation effect.

Usage

```
jqui_effect(ui, effect, options = NULL, duration = 400, complete = NULL)

jqui_show(ui, effect, options = NULL, duration = 400, complete = NULL)

jqui_hide(ui, effect, options = NULL, duration = 400, complete = NULL)

jqui_toggle(ui, effect, options = NULL, duration = 400, complete = NULL)
```

Arguments

`ui`	The target ui element(s) to be manipulated. Can be • A string of jQuery_selector • A JS() wrapped javascript expression that returns a jQuery object.
`effect`	A string indicating which animation effect to use for the transition.
`options`	A list of effect-specific properties and easing.
`duration`	A string or number determining how long the animation will run.
`complete`	A function to call once the animation is complete, called once per matched element.

Details

These functions are R wrappers of `effect()`, `hide()`, `show()` and `toggle()` from jQuery UI library. They should be used in `server` of a shiny document.

Examples

```
## Not run:
  # in shiny ui create a plot
  plotOutput('foo')

  # in shiny server apply a 'bounce' effect to the plot
  jqui_effect('#foo', 'bounce')

  # in shiny server hide the plot with a 'fold' effect
  jqui_hide('#foo', 'fold')

  # in shiny server show the plot with a 'blind' effect
  jqui_show('#foo', 'blind')

## End(Not run)
```

`Class_effects` *Class effects.*

Description

Manipulate specified class(es) to matched elements while animating all style changes.

- `jqui_add_class()`: Add class(es).
- `jqui_remove_class()`: Remove class(es).
- `jqui_switch_class()`: Switch class(es).

Usage

```
jqui_add_class(
  ui,
  className,
  duration = 400,
  easing = "swing",
  complete = NULL
)

jqui_remove_class(
  ui,
  className,
  duration = 400,
  easing = "swing",
```

```
  complete = NULL
)

jqui_switch_class(
  ui,
  removeClassName,
  addClassName,
  duration = 400,
  easing = "swing",
  complete = NULL
)
```

Arguments

ui	The target ui element(s) to be manipulated. Can be • A string of jQuery_selector • A JS() wrapped javascript expression that returns a jQuery object.
className	One or more class names (space separated) to be added to or removed from the class attribute of each matched element.
duration	A string or number determining how long the animation will run.
easing	A string indicating which easing function to use for the transition.
complete	A js function to call once the animation is complete, called once per matched element.
removeClassName	One or more class names (space separated) to be removed from the class attribute of each matched element.
addClassName	One or more class names (space separated) to be added to the class attribute of each matched element.

Details

These functions are the R wrappers of addClass(), removeClass() and switchClass() from jQuery UI library. They should be used in `server` of a shiny app.

Examples

```
## Not run:
  # in shiny ui create a span
  tags$span(id = 'foo', 'class animation demo')

  # in shiny server add class 'lead' to the span
  jqui_add_class('#foo', className = 'lead')

## End(Not run)
```

draggableModalDialog *Create a draggable modal dialog UI*

Description

This creates the UI for a modal dialog similar to shiny::modalDialog except its content is draggable.

Usage

```
draggableModalDialog(
  ...,
  title = NULL,
  footer = shiny::modalButton("Dismiss"),
  size = c("m", "s", "l"),
  easyClose = FALSE,
  fade = TRUE
)
```

Arguments

`...`	UI elements for the body of the modal dialog box.
`title`	An optional title for the dialog.
`footer`	UI for footer. Use `NULL` for no footer.
`size`	One of `"s"` for small, `"m"` (the default) for medium, or `"l"` for large.
`easyClose`	If `TRUE`, the modal dialog can be dismissed by clicking outside the dialog box, or be pressing the Escape key. If `FALSE` (the default), the modal dialog can't be dismissed in those ways; instead it must be dismissed by clicking on a `modalButton()`, or from a call to `removeModal()` on the server.
`fade`	If `FALSE`, the modal dialog will have no fade-in animation (it will simply appear rather than fade in to view).

Value

A modified shiny modal dialog UI with its content draggable.

get_jqui_effects *Get available animation effects.*

Description

Use this function to get all animation effects in jQuery UI.

Usage

```
get_jqui_effects()
```

Value

A character vector of effect names

`Interactions` *Mouse interactions*

Description

Attach mouse-based interactions to shiny html tags, shiny input/output widgets or static htmlwidgets and provide ways to manipulate them. The interactions include:

- draggable: Allow elements to be moved using the mouse.
- droppable: Create targets for draggable elements.
- resizable: Change the size of an element using the mouse.
- selectable: Use the mouse to select elements, individually or in a group.
- sortable: Reorder elements in a list or grid using the mouse.

Usage

```
jqui_draggable(
  ui,
  operation = c("enable", "disable", "destroy", "save", "load"),
  options = NULL
)

jqui_droppable(
  ui,
  operation = c("enable", "disable", "destroy", "save", "load"),
  options = NULL
)

jqui_resizable(
  ui,
  operation = c("enable", "disable", "destroy", "save", "load"),
  options = NULL
)

jqui_selectable(
  ui,
  operation = c("enable", "disable", "destroy", "save", "load"),
  options = NULL
)

jqui_sortable(
  ui,
  operation = c("enable", "disable", "destroy", "save", "load"),
  options = NULL
)
```

Arguments

`ui`	The target ui element(s) to be manipulated. Can be • A `shiny.tag` or `shiny.tag.list` object • A static `htmlwidget` object • A string of jQuery_selector • A JS() wrapped javascript expression that returns a jQuery object.
`operation`	A string to determine how to manipulate the mouse interaction. Can be one of `enable`, `disable`, `destroy`, `save` and `load`. Ignored when `ui` is a `shiny.tag` or `shiny.tag.list` object. See Details.
`options`	A list of interaction_specific_options. Ignored when `operation` is set as `destroy`. This parameter also accept a `shiny` option that controls the shiny input value returned from the element. See Details.

Details

The first parameter `ui` determines the target ui and working mode. If the target ui is a `shiny.tag` (e.g., shiny inputs/outputs or ui created by tags) or a `shiny.tag.list` (by tagList()) object or a static `htmlwidget`, the functions return the a modified ui object with interaction effects attached. When a jQuery_selector or a javascript expression is provided as the `ui` parameter, the functions first use it to locate the target ui element(s) in the shiny app, and then attach or manipulate the interactions. Therefore, you can use the first way in the `ui` of a shiny app to create elements with interaction effects (the ui mode), or use the second way in the `server` to manipulate the interactions (the server mode).

The `operation` parameter is valid only in server mode. It determines how to manipulate the interaction, which includes:

- `enable`: Attach the corresponding mouse interaction to the target(s).
- `disable`: Attach the interaction if not and disable it at once (only set the options).
- `destroy`: Destroy the interaction.
- `save`: Attach the interaction if not and save the current interaction state.
- `load`: Attach the interaction if not and restore the target(s) to the last saved interaction state.

With mouse interactions attached, the corresponding interaction states, e.g. `position` of draggable, `size` of resizable, `selected` of selectable and `order` of sortable, will be sent to server side in the form of `input$<id>_<state>`. The default values can be overridden by setting the `shiny` option in the `options` parameter. Please see the vignette `Introduction to shinyjqui` for more details.

Value

The same object passed in the `ui` parameter

Examples

```
library(shiny)
library(highcharter)

## used in ui
```

```
jqui_resizable(actionButton('btn', 'Button'))
jqui_draggable(plotOutput('plot', width = '400px', height = '400px'),
               options = list(axis = 'x'))
jqui_selectable(
  div(
    id = 'sel_plots',
    highchartOutput('highchart', width = '300px'),
    plotOutput('ggplot', width = '300px')
  ),
  options = list(
    classes = list(`ui-selected` = 'ui-state-highlight')
  )
)
jqui_sortable(tags$ul(
  id = 'lst',
  tags$li('A'),
  tags$li('B'),
  tags$li('C')
))

## used in server
## Not run:
  jqui_draggable('#foo', options = list(grid = c(80, 80)))
  jqui_droppable('.foo', operation = "enable")

## End(Not run)

## use shiny input
if (interactive()) {
  shinyApp(
    server = function(input, output) {
      output$foo <- renderHighchart({
        hchart(mtcars, "scatter", hcaes(x = cyl, y = mpg))
      })
      output$position <- renderPrint({
        print(input$foo_position)
      })
    },
    ui = fluidPage(
      verbatimTextOutput('position'),
      jqui_draggable(highchartOutput('foo', width = '200px', height = '200px'))
    )
  )
}

## custom shiny input
func <- JS('function(event, ui){return $(event.target).offset();}')
options <-  list(
  shiny = list(
    abs_position = list(
      dragcreate = func, # send returned value back to shiny when interaction is created.
      drag = func # send returned value to shiny when dragging.
    )
```

```
  )
)
jqui_draggable(highchartOutput('foo', width = '200px', height = '200px'),
               options = options)
```

`jqui_bookmarking` *Enable bookmarking state of mouse interactions*

Description

Enable shiny bookmarking_state of mouse interactions. By calling this function in `server`, the elements' `position`, `size`, `selection state` and `sorting state` changed by mouse operations can be saved and restored through an URL.

Usage

```
jqui_bookmarking()
```

`jqui_icon` *Create a jQuery UI icon*

Description

Create an jQuery UI pre-defined icon. For lists of available icons, see https://api.jqueryui.com/theming/icons/.

Usage

```
jqui_icon(name)
```

Arguments

`name`	Class name of icon. The "ui-icon-" prefix can be omitted (i.e. use "ui-icon-flag" or "flag" to display a flag icon)

Value

An icon element

Examples

```
jqui_icon('caret-1-n')

library(shiny)

# add an icon to an actionButton
actionButton('button', 'Button', icon = jqui_icon('refresh'))

# add an icon to a tabPanel
tabPanel('Help', icon = jqui_icon('help'))
```

`jqui_position` *Position an element relative to another*

Description

Wrapper of the jQuery UI .position() method, allows you to position an element relative to the window, document, another element, or the cursor/mouse, without worrying about offset parents.

Usage

```
jqui_position(
  ui,
  my = "center",
  at = "center",
  of,
  collision = "flip",
  within = JS("$(window)")
)
```

Arguments

`ui`	Which element to be positioned. Can be a string of jQuery_selector or a JS() wrapped javascript expression that returns a jQuery object. Only the first matching element will be used.
`my`	String. Defines which position **on the element being positioned** to align with the target element: "horizontal vertical" alignment. A single value such as "right" will be normalized to "right center", "top" will be normalized to "center top" (following CSS convention). Acceptable horizontal values: "left", "center", "right". Acceptable vertical values: "top", "center", "bottom". Example: "left top" or "center center". Each dimension can also contain offsets, in pixels or percent, e.g., "right+10 top-25%". Percentage offsets are relative to the element being positioned.
`at`	String. Defines which position **on the target element** to align the positioned element against: "horizontal vertical" alignment. See the `my` option for full details on possible values. Percentage offsets are relative to the target element.

of	Which element to position against. Can be a string of jQuery_selector or a JS() wrapped javascript expression that returns a jQuery object. Only the first matching element will be used.
collision	String. When the positioned element overflows the window in some direction, move it to an alternative position. Similar to my and at, this accepts a single value or a pair for horizontal/vertical, e.g., "flip", "fit", "fit flip", "fit none". • "flip": Flips the element to the opposite side of the target and the collision detection is run again to see if it will fit. Whichever side allows more of the element to be visible will be used. • "fit": Shift the element away from the edge of the window. • "flipfit": First applies the flip logic, placing the element on whichever side allows more of the element to be visible. Then the fit logic is applied to ensure as much of the element is visible as possible. • "none": Does not apply any collision detection.
within	Element to position within, affecting collision detection. Can be a string of jQuery_selector or a JS() wrapped javascript expression that returns a jQuery object. Only the first matching element will be used.

---

orderInput *Create a shiny input control to show the order of a set of items*

Description

Display a set of items whose order can be changed by drag and drop inside or between orderInput(s). The item order is send back to server in the from of input$inputId.

Usage

```
orderInput(
  inputId,
  label,
  items,
  as_source = FALSE,
  connect = NULL,
  item_class = c("default", "primary", "success", "info", "warning", "danger"),
  placeholder = NULL,
  width = "500px",
  legacy = FALSE,
  ...
)
```

Arguments

inputId	The input slot that will be used to access the current order of items.
label	Display label for the control, or NULL for no label.

items	Items to display, can be a list, an atomic vector or a factor. For list or atomic vector, if named, the names are displayed and the order is given in values. For factor, values are displayed and the order is given in levels
as_source	A boolean value to determine whether the orderInput is set as source mode. Only works if the connect argument was set.
connect	Optional. Allow items to be dragged between orderInputs. Should be a vector of inputId(s) of other orderInput(s) that the items from this orderInput should be connected to.
item_class	One of the Bootstrap color utility classes to apply to each item.
placeholder	A character string to show when there is no item left in the orderInput.
width	The width of the input, e.g. '400px', or '100\ shiny::validateCssUnit.
legacy	A boolean value. Whether to use the old version of the orderInput function.
...	Arguments passed to shiny::tags$div which is used to build the container of the orderInput.

Details

orderInputs can work in either connected mode or stand-alone mode. In stand-alone mode, items can only be drag and drop inside the input control. In connected mode, items to be dragged between orderInputs, which is controlled by the connect parameter. This is a one-way relationship. To connect items in both directions, the connect parameter must be set in both orderInputs.

When in connected mode, orderInput can be set as source-only through the as_source parameter. The items in a "source" orderInput can only be copied, instead of moved, to other connected non-source orderInput(s). From shinyjqui v0.4.0, A "source" orderInput will become a "recycle bin" for items from other orderInputs as well. This means, if you want to delete an item, you can drag and drop it into a "source" orderInput. This feature can be disabled by setting the options of non-source orderInput(s) as list(helper = "clone").

From shinyjqui v0.4.0 and above, the orderInput function was implemented in the similar way as other classical shiny inputs, which brought two changes:

1. The input value was changed from input$inputId_order to input$inputId;
2. The new version supports updateOrderInput function which works in the same way as other shiny input updater functions. To keep the backward compatibility, a legacy argument was provided if user wanted to use the old version.

Value

An orderInput control that can be added to a UI definition.

Examples

```
orderInput('items1', 'Items1', items = month.abb, item_class = 'info')

## build connections between orderInputs
orderInput('items2', 'Items2 (can be moved to Items1 and Items4)', items = month.abb,
           connect = c('items1', 'items4'), item_class = 'primary')
```

```
## build connections in source mode
orderInput('items3', 'Items3 (can be copied to Items2 and Items4)', items = month.abb,
           as_source = TRUE, connect = c('items2', 'items4'), item_class = 'success')

## show placeholder
orderInput('items4', 'Items4 (can be moved to Items2)', items = NULL, connect = 'items2',
           placeholder = 'Drag items here...')
```

`selectableTableOutput` *Create a table output element with selectable rows or cells*

Description

Render a standard HTML table with its rows or cells selectable. The server will receive the index of selected rows or cells stored in `input$<outputId>_selected`.

Usage

```
selectableTableOutput(outputId, selection_mode = c("row", "cell"))
```

Arguments

`outputId`	output variable to read the table from
`selection_mode`	one of `"row"` or `"cell"` to define either entire row or individual cell can be selected.

Details

Use mouse click to select single target, lasso (mouse dragging) to select multiple targets, and Ctrl + click to add or remove selection. In `row` selection mode, `input$<outputId>_selected` will receive the selected row index in the form of numeric vector. In `cell` selection mode, `input$<outputId>_selected` will receive a dataframe with `rows` and `columns` index of each selected cells.

Value

A table output element that can be included in a panel

See Also

shiny::tableOutput, sortableTableOutput

Examples

```
## Only run this example in interactive R sessions
if (interactive()) {
  shinyApp(
    ui = fluidPage(
      verbatimTextOutput("selected"),
      selectableTableOutput("tbl")
```

```
    ),
    server = function(input, output) {
      output$selected <- renderPrint({input$tbl_selected})
      output$tbl <- renderTable(mtcars, rownames = TRUE)
    }
  )
}
```

`sortableCheckboxGroupInput` *Create a Checkbox Group Input Control with Sortable Choices*

Description

Render a group of checkboxes with multiple choices toggleable. The choices are also sortable by drag and drop. In addition to the selected values stored in `input$<inputId>`, the server will also receive the order of choices in `input$<inputId>_order`.

Usage

```
sortableCheckboxGroupInput(
  inputId,
  label,
  choices = NULL,
  selected = NULL,
  inline = FALSE,
  width = NULL,
  choiceNames = NULL,
  choiceValues = NULL
)
```

Arguments

`inputId`	The `input` slot that will be used to access the value.
`label`	Display label for the control, or `NULL` for no label.
`choices`	List of values to show checkboxes for. If elements of the list are named then that name rather than the value is displayed to the user. If this argument is provided, then `choiceNames` and `choiceValues` must not be provided, and vice-versa. The values should be strings; other types (such as logicals and numbers) will be coerced to strings.
`selected`	The values that should be initially selected, if any.
`inline`	If `TRUE`, render the choices inline (i.e. horizontally)
`width`	The width of the input, e.g. `'400px'`, or `'100%'`; see `validateCssUnit()`.

`choiceNames` List of names and values, respectively, that are displayed to the user in the app and correspond to the each choice (for this reason, `choiceNames` and `choiceValues` must have the same length). If either of these arguments is provided, then the other *must* be provided and `choices` *must not* be provided. The advantage of using both of these over a named list for `choices` is that `choiceNames` allows any type of UI object to be passed through (tag objects, icons, HTML code, ...), instead of just simple text. See Examples.

`choiceValues` List of names and values, respectively, that are displayed to the user in the app and correspond to the each choice (for this reason, `choiceNames` and `choiceValues` must have the same length). If either of these arguments is provided, then the other *must* be provided and `choices` *must not* be provided. The advantage of using both of these over a named list for `choices` is that `choiceNames` allows any type of UI object to be passed through (tag objects, icons, HTML code, ...), instead of just simple text. See Examples.

Value

A list of HTML elements that can be added to a UI definition

See Also

shiny::checkboxGroupInput, `sortableRadioButtons()`, `sortableTableOutput()`, `sortableTabsetPanel()`

Examples

```
## Only run this example in interactive R sessions
if (interactive()) {
  shinyApp(
    ui = fluidPage(
      sortableCheckboxGroupInput("foo", "SortableCheckboxGroupInput",
                                 choices = month.abb),
      verbatimTextOutput("order")
    ),
    server = function(input, output) {
      output$order <- renderPrint({input$foo_order})
    }
  )
}
```

`sortableRadioButtons` *Create radio buttons with sortable choices*

Description

Create a set of radio buttons used to select an item from a list. The choices are sortable by drag and drop. In addition to the selected values stored in `input$<inputId>`, the server will also receive the order of choices in `input$<inputId>_order`.

Usage

```
sortableRadioButtons(
  inputId,
  label,
  choices = NULL,
  selected = NULL,
  inline = FALSE,
  width = NULL,
  choiceNames = NULL,
  choiceValues = NULL
)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
choices	List of values to select from (if elements of the list are named then that name rather than the value is displayed to the user). If this argument is provided, then choiceNames and choiceValues must not be provided, and vice-versa. The values should be strings; other types (such as logicals and numbers) will be coerced to strings.
selected	The initially selected value. If not specified, then it defaults to the first item in choices. To start with no items selected, use character(0).
inline	If TRUE, render the choices inline (i.e. horizontally)
width	The width of the input, e.g. '400px', or '100%'; see validateCssUnit().
choiceNames	List of names and values, respectively, that are displayed to the user in the app and correspond to the each choice (for this reason, choiceNames and choiceValues must have the same length). If either of these arguments is provided, then the other *must* be provided and choices *must not* be provided. The advantage of using both of these over a named list for choices is that choiceNames allows any type of UI object to be passed through (tag objects, icons, HTML code, ...), instead of just simple text. See Examples.
choiceValues	List of names and values, respectively, that are displayed to the user in the app and correspond to the each choice (for this reason, choiceNames and choiceValues must have the same length). If either of these arguments is provided, then the other *must* be provided and choices *must not* be provided. The advantage of using both of these over a named list for choices is that choiceNames allows any type of UI object to be passed through (tag objects, icons, HTML code, ...), instead of just simple text. See Examples.

Value

A set of radio buttons that can be added to a UI definition.

See Also

shiny::radioButtons, sortableCheckboxGroupInput, sortableTableOutput, sortableTabsetPanel

Examples

```
## Only run this example in interactive R sessions
if (interactive()) {
  shinyApp(
    ui = fluidPage(
      sortableRadioButtons("foo", "SortableRadioButtons",
                           choices = month.abb),
      verbatimTextOutput("order")
    ),
    server = function(input, output) {
      output$order <- renderPrint({input$foo_order})
    }
  )
}
```

`sortableTableOutput` *Create a table output element with sortable rows*

Description

Render a standard HTML table with table rows sortable by drag and drop. The order of table rows is recorded in `input$<outputId>_order`.

Usage

```
sortableTableOutput(outputId)
```

Arguments

`outputId`	output variable to read the table from

Value

A table output element that can be included in a panel

See Also

shiny::tableOutput, sortableRadioButtons, sortableCheckboxGroupInput, sortableTabsetPanel, selectableTableOutput

Examples

```
## Only run this example in interactive R sessions
if (interactive()) {
  shinyApp(
    ui = fluidPage(
      verbatimTextOutput("rows"),
      sortableTableOutput("tbl")
```

```
    ),
    server = function(input, output) {
      output$rows <- renderPrint({input$tbl_row_index})
      output$tbl <- renderTable(mtcars, rownames = TRUE)
    }
  )
}
```

`sortableTabsetPanel` *Create a tabset panel with sortable tabs*

Description

Create a tabset that contains shiny::tabPanel elements. The tabs are sortable by drag and drop. In addition to the activated tab title stored in `input$<id>`, the server will also receive the order of tabs in `input$<id>_order`.

Usage

```
sortableTabsetPanel(
  ...,
  id = NULL,
  selected = NULL,
  type = c("tabs", "pills", "hidden"),
  header = NULL,
  footer = NULL
)
```

Arguments

`...`	`tabPanel()` elements to include in the tabset
`id`	If provided, you can use `input$id` in your server logic to determine which of the current tabs is active. The value will correspond to the `value` argument that is passed to `tabPanel()`.
`selected`	The `value` (or, if none was supplied, the `title`) of the tab that should be selected by default. If `NULL`, the first tab will be selected.
`type`	`"tabs"` Standard tab look `"pills"` Selected tabs use the background fill color `"hidden"` Hides the selectable tabs. Use `type = "hidden"` in conjunction with `tabPanelBody()` and `updateTabsetPanel()` to control the active tab via other input controls. (See example below)
`header`	Tag or list of tags to display as a common header above all tabPanels.
`footer`	Tag or list of tags to display as a common footer below all tabPanels

Value

A tabset that can be passed to shiny::mainPanel

See Also

shiny::tabsetPanel, sortableRadioButtons, sortableCheckboxGroupInput, sortableTableOutput

Examples

```
## Only run this example in interactive R sessions
if (interactive()) {
  shinyApp(
    ui = fluidPage(
      sortableTabsetPanel(
        id = "tabs",
        tabPanel(title = "A", "AAA"),
        tabPanel(title = "B", "BBB"),
        tabPanel(title = "C", "CCC")
      ),
      verbatimTextOutput("order")
    ),
    server = function(input, output) {
      output$order <- renderPrint({input$tabs_order})
    }
  )
}
```

---

updateOrderInput *Change the value of an orderInput on the client*

Description

Similar to the input updater functions of shiny package, this function send a message to the client, telling it to change the settings of an orderInput object. Any arguments with NULL values will be ignored; they will not result in any changes to the input object on the client. The function can't update the "source" `orderInputs`.

Usage

```
updateOrderInput(
  session,
  inputId,
  label = NULL,
  items = NULL,
  connect = NULL,
  item_class = NULL
)
```

Arguments

session	The session object passed to function given to shinyServer.
inputId	The input slot that will be used to access the current order of items.
label	Display label for the control, or NULL for no label.
items	Items to display, can be a list, an atomic vector or a factor. For list or atomic vector, if named, the names are displayed and the order is given in values. For factor, values are displayed and the order is given in levels
connect	Optional. Allow items to be dragged between orderInputs. Should be a vector of inputId(s) of other orderInput(s) that the items from this orderInput should be connected to.
item_class	One of the Bootstrap color utility classes to apply to each item.

Examples

```
library(shiny)

if (interactive()) {

  ui <- fluidPage(
    orderInput("foo", "foo",
               items = month.abb[1:3],
               item_class = 'info'),
    verbatimTextOutput("order"),
    actionButton("update", "update")
  )

  server <- function(input, output, session) {
    output$order <- renderPrint({input$foo})
    observeEvent(input$update, {
      updateOrderInput(session, "foo",
                       items = month.abb[1:6],
                       item_class = "success")
    })
  }

  shinyApp(ui, server)

}
```

Index