Package 'epivizrServer'

April 15, 2020

Type Package

Title WebSocket server infrastructure for epivizr apps and packages

Version 1.14.0

URL https://epiviz.github.io

BugReports https://github.com/epiviz/epivizrServer

Description This package provides objects to manage WebSocket connections to epiviz apps. Other epivizr package use this infrastructure.

biocViews Infrastructure, Visualization

VignetteBuilder knitr

Depends R (>= 3.2.3), methods

Imports httpuv (>= 1.3.0), R6 (>= 2.0.0), rjson, mime (>= 0.2)

Suggests testthat, knitr, rmarkdown, BiocStyle

License MIT + file LICENSE

LazyData true

Collate 'IndexedArray-class.R' 'Queue-class.R' 'utils.R' 'zzz.R' 'middleware-plus-supporting.R' 'dummyTestPage.R' 'EpivizServer-class.R' 'createServer.R'

RoxygenNote 5.0.1

NeedsCompilation no

Author Hector Corrada Bravo [aut, cre]

Maintainer Hector Corrada Bravo <hcorrada@gmail.com>

git_url https://git.bioconductor.org/packages/epivizrServer

git_branch RELEASE_3_10

git_last_commit 60da989

git_last_commit_date 2019-10-29

Date/Publication 2020-04-14

R topics documented:

`createServer` *Create a new EpivizServer object*

Description

Create a new EpivizServer object

Usage

```
createServer(port = 7123L, static_site_path = "", try_ports = FALSE,
  daemonized = NULL, verbose = FALSE, non_interactive = FALSE)
```

Arguments

`port`	(int) port to which server will listen to.
`static_site_path`	(character) path to serve static html files.
`try_ports`	(logical) try various ports until an open port is found.
`daemonized`	(logical) run in background using httpuv's daemonized libuv server.
`verbose`	(logical) print verbose output.
`non_interactive`	(logical) run in non-interactive mode. For development purposes only.

Value

an `EpivizServer` object

See Also

`EpivizServer` for the class of objects returned

Examples

```
server <- createServer(port=7123,
                       verbose=TRUE
                       )
```

EpivizServer-class *Class providing WebSocket connection server*

Description

Class providing WebSocket connection server

Details

The most important aspect of the API of this server are methods `register_action` and `send_request`. These are used to interact with the epiviz JS app through the provided websocket connection. `register_action(action,callback)` registers a callback function to be executed upon request from the epiviz JS app. When the server receives a JSON message through the websocket, it checks for an `action` field in the received request message, and then evaluates the expression `callback(message_data)` where `message_data` is obtained from the `data` field in the received message. A response will be sent to the epiviz app with field `data` populated with the result of the callback. If an error occurs during evaluation of the callback function, the response will be sent with field `success` set to `false`.

To send requests to the JS app, method `send_request(request_data,callback)` should be used. This is sends a request to the JS app with the `data` field populated with argument `request_data`. Once a response is received (with field `success` equal to `true`) the expression `callback(response_data)` is evaluated where `response_data` is obtained from the `data` field in the received response message.

Value

RC object with methods for communication with epiviz JS app

Methods

`has_action(action)` Check if a callback is registered for given action<character>, <logical>. (See Details)

`has_request_waiting()` Check if there is a sent request waiting for a response from JS app, <logical>

`is_closed()` Check if server is closed, <logical>

`is_daemonized()` Check if server is running in background, <logical>

`is_interactive()` Check if server is running in interactive mode, <logical>

`is_socket_connected()` Check if there is an open websocket connection to JS app, <logical>

`register_action(action, callback)` Register a callback<function> to evaluate when epiviz JS sends a request for given action<character>. (See Details)

`run_server(...)` Run server in blocking mode

`send_request(request_data, callback)` Send request to epiviz JS app with given request_data<list>, and evaluate callback<function> when response arrives. (See Details)

`service()` Listen to requests from server. Only has effect when non-daemonized

`start_server()` Start the underlying httpuv server, daemonized if applicable

`stop_server()` Stop the underlying httpuv server

`stop_service()` Stop listenning to requests from server. Only has effect when non-daemonized.

`unregister_action(action)` Unregister a callback function for given action<character> (if registered). (See Details)

`wait_to_clear_requests(timeout = 3L)` Wait for `timeout` seconds to clear all pending requests.

Examples

```
server <- createServer()
server$register_action("getData", function(request_data) {
  list(x=1,y=3)
})

server$start_server()

server$send_request(list(x=2,y=5), function(response_data) {
  cat(response_data$x)
})

server$stop_server()
```

IndexedArray-class *Class providing an indexed array (hashtable)*

Description

Class providing an indexed array (hashtable)

Methods

`append(item)` Append item to tail of array, returns id of item <int>

`empty()` Remove all items from array

`get(id)` Get item with given id<int>, returns <ANY>, returns NULL if no item with given id

`length()` Return number of items on array <int>

---

json_parser *JSON parser used by this package*

Description

Currently this just renames `fromJSON` in the `rjson` package.

Usage

```
json_parser(json_str, file, method = "C", unexpected.escape = "error",
  simplify = TRUE)
```

Arguments

`json_str`	json string to parse
`file`	file to read json_Str from
`method`	method used to parse json
`unexpected.escape`	handling escape characters, one of error, skip, keep
`simplify`	if TRUE, convert json-encoded lists to vectors

Value

a JSON object

See Also

`fromJSON`

`json_writer` *JSON writer used by this package*

Description

Currently this just renames `toJSON` in the `rjson` package.

Usage

```
json_writer(x, indent = 0, method = "C")
```

Arguments

`x`	object to write to json
`indent`	integer specifying how much indentation to use when formatting the JSON object; if 0, no pretty-formatting is used
`method`	method used to write json

Value

a string with JSON encoding of object

See Also

`toJSON`

`Queue-class` *Class providing a queue data structure*

Description

Class providing a queue data structure

Methods

`empty()` Remove all items from queue

`has_more()` Return TRUE if there are more items in queue <logical>

`length()` Return the number of items in queue <int>

`pop()` Pop next item from queue (returns NULL if queue is empty)

`push(item)` Push <item> onto queue

Index