

Package ‘V8’

April 13, 2019

Type Package

Title Embedded JavaScript Engine for R

Version 2.2

Description An R interface to Google's open source JavaScript engine. This package can now be compiled either with V8 version 6 or 7 (LTS) from nodejs or with the legacy 3.14/3.15 branch of V8.

License MIT + file LICENSE

URL <https://github.com/jeroen/v8> (devel) <https://v8.dev> (upstream)

BugReports <https://github.com/jeroen/v8/issues>

SystemRequirements V8 version 6 or 7 is recommended, but 3.14 (legacy) is still supported as well. On Debian / Ubuntu you need either libv8-dev or libnode-dev, on Fedora use v8-devel.

NeedsCompilation yes

VignetteBuilder knitr

Imports Rcpp (>= 0.12.12), jsonlite (>= 1.0), curl (>= 1.0), utils

LinkingTo Rcpp

Suggests testthat, knitr, rmarkdown

RoxygenNote 6.1.1

Language en-US

Author Jeroen Ooms [aut, cre] (<<https://orcid.org/0000-0002-4035-0289>>)

Maintainer Jeroen Ooms <jeroen@berkeley.edu>

Repository CRAN

Date/Publication 2019-04-13 07:22:41 UTC

R topics documented:

JS	2
V8	2
Index	6

 JS

Mark character strings as literal JavaScript code

Description

This function JS() marks character vectors with a special class, so that it will be treated as literal JavaScript code. It was copied from the htmlwidgets package, and does exactly the same thing.

Usage

```
JS(...)
```

Arguments

... character vectors as the JavaScript source code (all arguments will be pasted into one character string)

Author(s)

Yihui Xie

Examples

```
ct <- v8()
ct$eval("1+1")
ct$eval(JS("1+1"))
ct$assign("test", JS("2+3"))
ct$get("test")
```

 V8

Run JavaScript in a V8 context

Description

The v8 function (formerly called new_context) creates a new V8 *context*. A context provides an execution environment that allows separate, unrelated, JavaScript code to run in a single instance of V8, like a tab in a browser.

Usage

```
v8(global = "global", console = TRUE, typed_arrays = TRUE)
```

```
engine_info()
```

Arguments

<code>global</code>	character vector indicating name(s) of the global environment. Use NULL for no name.
<code>console</code>	expose console API (<code>console.log</code> , <code>console.warn</code> , <code>console.error</code>).
<code>typed_arrays</code>	used to enable support for typed arrays in legacy libv8. This is enabled by default in recent versions of libv8.

Details

V8 contexts cannot be serialized but creating a new contexts and sourcing code is very cheap. You can run as many parallel v8 contexts as you want. R packages that use V8 can use a separate V8 context for each object or function call.

The `ct$eval` method evaluates a string of raw code in the same way as `eval` would do in JavaScript. It returns a string with console output. The `ct$get`, `ct$assign` and `ct$call` functions on the other hand automatically convert arguments and return value from/to JSON, unless an argument has been wrapped in `JS()`, see examples. The `ct$validate` function is used to test if a piece of code is valid JavaScript syntax within the context, and always returns TRUE or FALSE.

JSON is used for all data interchange between R and JavaScript. Therefore you can (and should) only exchange data types that have a sensible JSON representation. All arguments and objects are automatically converted according to the mapping described in [Ooms \(2014\)](#), and implemented by the `jsonlite` package in `fromJSON` and `toJSON`.

The name of the global object (i.e. `global` in node and `window` in browsers) can be set with the `global` argument. A context always have a global scope, even when no name is set. When a context is initiated with `global = NULL`, the global environment can be reached by evaluating this in the global scope, for example: `ct$eval("Object.keys(this)")`.

Methods

<code>console()</code>	starts an interactive console
<code>eval(src)</code>	evaluates a string with JavaScript source code
<code>validate(src)</code>	test if a string of JavaScript code is syntactically valid
<code>source(file)</code>	evaluates a file with JavaScript code
<code>get(name, ...)</code>	convert a JavaScript to R via JSON. Optional arguments (...) are passed to <code>fromJSON</code> to set JSON coercion options.
<code>assign(name, value)</code>	copy an R object to JavaScript via JSON
<code>call(fun, ...)</code>	call a JavaScript function with arguments ... Arguments which are not wrapped in <code>JS()</code> automatically get converted to JSON
<code>reset()</code>	resets the context (removes all objects)

References

A Mapping Between JSON Data and R Objects (Ooms, 2014): <http://arxiv.org/abs/1403.2805>

Examples

```

# Create a new context
ctx <- v8();

# Evaluate some code
ctx$eval("var foo = 123")
ctx$eval("var bar = 456")
ctx$eval("foo+bar")

# Functions and closures
ctx$eval("JSON.stringify({x:Math.random()})")
ctx$eval("(function(x){return x+1;})(123)")

# Objects (via JSON only)
ctx$assign("mydata", mtcars)
ctx$get("mydata")
ctx$get("mydata", simplifyVector = FALSE)

# Assign JavaScript
ctx$assign("foo", JS("function(x){return x*x}"))
ctx$assign("bar", JS("foo(9)"))
ctx$get("bar")

# Validate script without evaluating
ctx$validate("function foo(x){2*x}") #TRUE
ctx$validate("foo = function(x){2*x}") #TRUE
ctx$validate("function(x){2*x}") #FALSE

# Use a JavaScript library
ctx$source(system.file("js/underscore.js", package="V8"))
ctx$call("_.filter", mtcars, JS("function(x){return x.mpg < 15}"))

# Example from underscore manual
ctx$eval("_.templateSettings = {interpolate: /\{\{\{(.+)\}\}\}/g}")
ctx$eval("var template = _.template('Hello {{ name }}!')")
ctx$call("template", list(name = "Mustache"))

# Call anonymous function
ctx$call("function(x, y){return x * y}", 123, 3)

## Not run: CoffeeScript
ct2 <- v8()
ct2$source("http://coffeescript.org/v1/browser-compiler/coffee-script.js")
jrcode <- ct2$call("CoffeeScript.compile", "square = (x) -> x * x", list(bare = TRUE))
ct2$eval(jrcode)
ct2$call("square", 9)

# Interactive console
ct3 <- v8()
ct3$console()
//this is JavaScript
var test = [1,2,3]

```

```
JSON.stringify(test)
exit
## End(Not run)
```

Index

`engine_info` (V8), [2](#)

`fromJSON`, [3](#)

JS, [2](#)

`new_context` (V8), [2](#)

`toJSON`, [3](#)

V8, [2](#)

v8, [2](#)

v8 (V8), [2](#)