Package 'V8'

February 15, 2024

Type Package

Title Embedded JavaScript and WebAssembly Engine for R

Version 4.4.2

Description An R interface to V8 <https://v8.dev>: Google's open source JavaScript and WebAssembly engine. This package can be compiled either with V8 version 6 and up or NodeJS when built as a shared library.

License MIT + file LICENSE

URL https://jeroen.r-universe.dev/V8

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

SystemRequirements V8 engine version 6+ is needed for ES6 and WASM support. On Linux you can build against libv8-dev (Debian) or v8-devel (Fedora). We also provide static libv8 binaries for most platforms, see the README for details.

NeedsCompilation yes

VignetteBuilder knitr

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

LinkingTo Rcpp

Suggests testthat, knitr, rmarkdown

RoxygenNote 7.2.3

Language en-US

Encoding UTF-8

Biarch true

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

Maintainer Jeroen Ooms <jeroen@berkeley.edu>

Repository CRAN

Date/Publication 2024-02-15 15:00:02 UTC

R topics documented:

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

global	character vector indicating name(s) of the global environment. Use NULL for no name.
console	expose console API (console.log, console.warn, console.error).
typed_arrays	(deprecated) enable typed arrays in legacy libv8. Deprecated because typed arrays are natively supported in recent versions of libv8.

Details

A V8 context cannot be saved or duplicated, but creating a new context 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 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)").

V8 Context Methods

```
## ctx <- v8()
<V8 engine 11.7.439.14>
 $assign(name, value, auto_unbox = TRUE, ...)
 $call(fun, ..., auto_unbox = TRUE, await = FALSE)
 $console()
 $eval(src, serialize = FALSE, await = FALSE)
 $get(name, ..., await = FALSE)
 $reset()
 $source(file)
 $validate(src)
```

The ct$eval method evaluates a string of JavaScript code in the same way as eval in JavaScript. By default eval() returns a string with console output; but when the serialize parameter is set to TRUE it serializes the JavaScript return object to a JSON string or a raw buffer.

The ctget, ctassign and ct$call functions automatically convert arguments and return value between R and JavaScript (using JSON). To pass literal JavaScript arguments that should not be converted to JSON, wrap them in JS(), see examples.

If a call to ct$eval(),ct$get(), or ct$call() returns a JavaScript promise, you can set await = TRUE to wait for the promise to be resolved. It will then return the result of the promise, or an error in case the promise is rejected.

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.

In an interactive R session you can use `ct$console()` to switch to an interactive JavaScript console. Here you can use `console.log` to print objects, and there is some support for JS tab-completion. This is mostly for testing and debugging, it may not work perfectly in every IDE or R-frontend.

Data Interchange

JSON is used for data interchange between R and JavaScript. Therefore you can (and should) only exchange data types that have a sensible JSON representation. One exception is raw vectors which are converted to/from Uint8Array buffers, see below. All other arguments and objects are automatically converted according to the mapping described in Ooms (2014), and implemented by the jsonlite package in `fromJSON()` and `toJSON()`.

As for version 3.0 of this R package, Raw vectors are converted to `Uint8Array` typed arrays, and vice versa. This makes it possible to efficiently copy large chunks binary data between R and JavaScript, which is useful for running wasm or emscripten.

Note about Linux and Legacy V8 engines

This R package can be compiled against modern (V8 version 6+) libv8 API, or the legacy libv8 API (V8 version 3.15 and below). You can check `V8::engine_info()` to see the version that is running. The legacy version does not support modern JS (ES6) or WASM, but it is still the default on older versions of Ubuntu and CentOS. The latest versions of all major Linux distributions now provide a modern version of V8. For Ubuntu 16.04 and 18.04 we provide backports of libv8 (via libnode-dev), see the readme for details.

References

A Mapping Between JSON Data and R Objects (Ooms, 2014): https://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")
outlist <- ctx$get("mydata", simplifyVector = FALSE)
outlist[1]

# 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$source("https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.13.6/underscore-min.js")
ctx$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$source("http://coffeescript.org/v1/browser-compiler/coffee-script.js")
jscode <- ct2$call("CoffeeScript.compile", "square = (x) -> x * x", list(bare = TRUE))
ct2$eval(jscode)
ct2$call("square", 9)

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

## End(Not run)
```

`wasm` *Experimental WebAssembly*

Description

Experimental wrapper to load a WebAssembly program. Returns a list of exported functions. This will probably be moved into it's own package once WebAssembly matures.

Usage

```
wasm(data)
```

```
wasm_features()
```

Arguments

data	either raw vector or file path with the binary wasm program

Details

The `wasm_features()` function uses the wasm-feature-detect JavaScript library to test which WASM capabilities are supported in the current version of libv8.

Examples

```
# Load example wasm program
instance <- wasm(system.file('wasm/add.wasm', package = 'V8'))
instance$exports$add(12, 30)
wasm_features()
```

Index