Package ‘JuliaCall’

November 27, 2019

<table>
<thead>
<tr>
<th>Type</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Seamless Integration Between R and 'Julia'</td>
</tr>
<tr>
<td>Version</td>
<td>0.17.1</td>
</tr>
<tr>
<td>Date</td>
<td>2019-11-26</td>
</tr>
<tr>
<td>Description</td>
<td>Provides an R interface to 'Julia', which is a high-level, high-performance dynamic programming language for numerical computing, see <a href="https://julialang.org/">https://julialang.org/</a> for more information. It provides a high-level interface as well as a low-level interface. Using the high level interface, you could call any 'Julia' function just like any R function with automatic type conversion. Using the low level interface, you could deal with C-level SEXP directly while enjoying the convenience of using a high-level programming language like 'Julia'.</td>
</tr>
<tr>
<td>Depends</td>
<td>R (&gt;= 3.4.0)</td>
</tr>
<tr>
<td>License</td>
<td>MIT + file LICENSE</td>
</tr>
<tr>
<td>URL</td>
<td><a href="https://github.com/Non-Contradiction/JuliaCall">https://github.com/Non-Contradiction/JuliaCall</a></td>
</tr>
<tr>
<td>BugReports</td>
<td><a href="https://github.com/Non-Contradiction/JuliaCall/issues">https://github.com/Non-Contradiction/JuliaCall/issues</a></td>
</tr>
<tr>
<td>Encoding</td>
<td>UTF-8</td>
</tr>
<tr>
<td>LazyData</td>
<td>true</td>
</tr>
<tr>
<td>Imports</td>
<td>utils, Rcpp (&gt;= 0.12.7), knitr (&gt;= 1.18)</td>
</tr>
<tr>
<td>RoxygenNote</td>
<td>7.0.0</td>
</tr>
<tr>
<td>LinkingTo</td>
<td>Rcpp</td>
</tr>
<tr>
<td>NeedsCompilation</td>
<td>yes</td>
</tr>
<tr>
<td>ByteCompile</td>
<td>yes</td>
</tr>
<tr>
<td>SystemRequirements</td>
<td>Julia &gt;= 0.6.0, RCall.jl</td>
</tr>
<tr>
<td>Suggests</td>
<td>testthat, rmarkdown</td>
</tr>
<tr>
<td>VignetteBuilder</td>
<td>knitr</td>
</tr>
<tr>
<td>Author</td>
<td>Changcheng Li [aut, cre], Randy Lai [ctb], Dmitri Grominski [ctb], Nagi Teramo [ctb]</td>
</tr>
</tbody>
</table>
Description

autowrap tells ‘JuliaCall’ to use automatic wrapper for julia type.

Usage

autowrap(type, fields = NULL, methods = c())

Arguments

type the julia type to wrap.
fields names of fields to be included in the wrapper. If the value is NULL, then every julia fields will be included in the wrapper.
methods names of methods to be overloaded for the wrapper.
Description

julia_do.call is the do.call for julia. And julia_call calls julia functions. For usage of these functions, see documentation of arguments and examples.

Usage

julia_do.call(
    func_name,
    arg_list,
    need_return = c("R", "Julia", "None"),
    show_value = FALSE
)

julia_call(
    func_name,
    ...,
    need_return = c("R", "Julia", "None"),
    show_value = FALSE
)

Arguments

func_name the name of julia function you want to call. If you add "." after ‘func_name’, the julia function call will be broadcasted.
arg_list the list of the arguments you want to pass to the julia function.
need_return whether you want julia to return value as an R object, a wrapper for julia object or no return. The value of need_return could be TRUE (equal to option "R") or FALSE (equal to option "None"), or one of the options "R", "Julia" and "None".
show_value whether to invoke the julia display system or not.
... the arguments you want to pass to the julia function.

Details

Note that named arguments will be discarded if the call uses dot notation, for example, "sqrt.".

Examples

```r
## julia_setup is quite time consuming
julia_do.call("sqrt", list(2))
julia_call("sqrt", 2)
julia_call("sqrt.", 1:10)
```
Description

Julia language engine in R Markdown

Usage

eng_juliacall(options)

Arguments

options a list of chunk options

Examples

knitr::knit_engines$set(julia = JuliaCall::eng_juliacall)

JuliaCall

JuliaCall: Seamless Integration Between R and Julia.

Description

JuliaCall provides you with functions to call Julia functions and to use Julia packages as easy as possible.

Examples

## The examples are quite time consuming
## Do initiation for JuliaCall
julia <- julia_setup()

## Different ways for calculating \texttt{sqrt(2)}
# julia$command("a = sqrt(2)"PILE); julia$eval("a")
julia_command("a = sqrt(2)"PILE); julia_eval("a")

# julia$eval("sqrt(2)"PILE)
julia_eval("sqrt(2)"PILE)

# julia$call("sqrt", 2)
julia_call("sqrt", 2)

# julia$eval("sqrt")(2)
julia_eval("sqrt")(2)

## You can use `julia_exists` as `exists` in R to test whether a function or name exists in Julia or not

# julia$exists("sqrt")
julia_exists("sqrt")

## You can use `julia$help` to get help for Julia functions

# julia$help("sqrt")
julia_help("sqrt")

## You can install and use Julia packages through JuliaCall

# julia$install_package("Optim")
julia_install_package("Optim")

# julia$install_package_if_needed("Optim")
julia_install_package_if_needed("Optim")

# julia$installed_package("Optim")
julia_installed_package("Optim")

# julia$library("Optim")
julia_library("Optim")

---

**JuliaObject**

Convert an R Object to Julia Object.

**Description**

JuliaObject converts an R object to julia object and returns a reference of the corresponding julia object.

**Usage**

JuliaObject(x)

**Arguments**

x the R object you want to convert to julia object.

**Value**

an environment of class JuliaObject, which contains an id corresponding to the actual julia object.
## JuliaObjectFields

### Examples

```r
## julia_setup is quite time consuming
a <- JuliaObject(1)
```

### Description

Get the field names, get or set certain fields of an JuliaObject.

### Usage

```r
fields(object)
```

```r
## S3 method for class 'JuliaObject'
fields(object)
```

```r
field(object, name)
```

```r
## S3 method for class 'JuliaObject'
field(object, name)
```

```r
field(object, name) <- value
```

```r
## S3 replacement method for class 'JuliaObject'
field(object, name) <- value
```

### Arguments

- **object**: the JuliaObject.
- **name**: a character string specifying the fields to be accessed or set.
- **value**: the new value of the field of the JuliaObject.
julia_assign

Assign a value to a name in julia.

Description

julia_assign assigns a value to a name in julia with automatic type conversion.

Usage

julia_assign(x, value)

Arguments

x
a variable name, given as a character string.

value
a value to be assigned to x, note that R value will be converted to corresponding julia value automatically.

Examples

## julia_setup is quite time consuming
julia_assign("x", 2)
julia_assign("rsqrt", sqrt)

julia_command

Evaluate string commands in julia and (may) invoke the julia display system.

Description

julia_command evaluates string commands in julia without returning the result back to R. However, it may evoke julia display system, see the documentation of the argument ‘show_value‘ for more details. If you need to get the evaluation result in R, you can use julia_eval.

Usage

julia_command(cmd, show_value = !endsWith(trimws(cmd, "right"), ";"))

Arguments

cmd
the command string you want to evaluate in julia.

show_value
whether to display julia returning value or not, the default value is ‘FALSE‘ if the ‘cmd‘ ends with semicolon and ‘TRUE‘ otherwise.
Examples

```r
## julia_setup is quite time consuming
julia_command("a = sqrt(2);")
```

### julia_console

Open julia console.

#### Description

Open julia console.

#### Usage

```r
julia_console()
```

#### Examples

```r
## Not run: ## julia_setup is quite time consuming
julia_console()
```

```r
## End(Not run)
```

### julia_eval

Evaluate string commands in julia and get the result back in R.

#### Description

`julia_eval` evaluates string commands in julia and returns the result to R. The returning julia object will be automatically converted to an R object or a JuliaObject wrapper, see the documentation of the argument `need_return` for more details. `julia_eval` will not invoke julia display system. If you don’t need the returning result in R or you want to invoke the julia display system, you can use `julia_command`.

#### Usage

```r
julia_eval(cmd, need_return = c("R", "Julia"))
```

#### Arguments

- `cmd`: the command string you want to evaluate in julia.
- `need_return`: whether you want julia to return value as an R object or a wrapper for julia object.
**julia_exists**

**Value**

the R object automatically converted from julia object.

**Examples**

```r
## julia_setup is quite time consuming
julia_eval("sqrt(2")
```

---

**julia_exists**  
Check whether a julia object with the given name exists or not.

**Description**

julia_exists returns whether a julia object with the given name exists or not.

**Usage**

```r
julia_exists(name)
```

**Arguments**

name  
the name of julia object you want to check.

**Examples**

```r
## julia_setup is quite time consuming
julia_exists("sqrt")
```

---

**julia_help**  
Get help for a julia function.

**Description**

julia_help outputs the documentation of a julia function.

**Usage**

```r
julia_help(fname)
```
### julia_setup

**Arguments**

- **fname**: the name of julia function you want to get help with.

**Examples**

```r
## julia_setup is quite time consuming
julia_help("sqrt")
```

---

### julia_markdown_setup

**Description**

`julia_markdown_setup` does the initial setup for JuliaCall in RMarkdown document and RStudio notebooks. The function should be invoked automatically most of the case. It can also be called explicitly in RMarkdown documents or notebooks.

**Usage**

```r
julia_markdown_setup(..., notebook = FALSE)
```

**Arguments**

- **...**: The same arguments accepted by `julia_setup`.
- **notebook**: whether it is in RStudio notebook environment or not.

---

### julia_notebook_setup

**Description**

`julia_notebook_setup` is deprecated, use `julia_markdown_setup(notebook=TRUE)` instead.

**Usage**

```r
julia_notebook_setup(...)
```

**Arguments**

- **...**: The same arguments accepted by `julia_setup`. 

---
Using Julia packages.

Description

Using Julia packages.

Usage

julia_install_package(pkg_name_or_url)

julia_installed_package(pkg_name)

julia_install_package_if_needed(pkg_name)

julia_update_package(...)  

julia_library(pkg_name)

Arguments

pkg_name_or_url
   the Julia package name or url.

pkg_name
   the Julia package name.

...  
   you can provide none or one or multiple Julia package names here.

Value

julia_installed_package will return the version number of the Julia package, "nothing" if the package is not installed.

Examples

## Julia_setup is quite time consuming  
julia_install_package("DataFrames")  
julia_install_package("https://github.com/JuliaData/DataFrames.jl")  
julia_installed_package("DataFrames")  
julia_install_package_if_needed("DataFrames")  
julia_update_package("DataFrame")  
julia_library("DataFrames")
### julia_setup

Do initial setup for the JuliaCall package.

#### Description

julia_setup does the initial setup for the JuliaCall package. It setups automatic type conversion, Julia display systems, etc, and is necessary for every new R session to use the package. If not carried out manually, it will be invoked automatically before other julia_xxx functions.

---

#### julia_pkg_wrap

Wrap julia functions and packages the easy way.

**Description**

Wrap julia functions and packages the easy way.

**Usage**

```r
julia_function(func_name, pkg_name = "Main", env = new.env(emptyenv()))
```

```r
julia_pkg_import(pkg_name, func_list, env = new.env(parent = emptyenv()))
```

```r
julia_pkg_hook(env, hook)
```

**Arguments**

- **func_name**: the julia function name to be wrapped.
- **pkg_name**: the julia package name to be wrapped.
- **env**: the environment where the functions and packages are wrapped.
- **func_list**: the list of julia function names to be wrapped.
- **hook**: the function to be executed before the execution of wrapped functions.

**Examples**

```r
## julia_setup is quite time consuming
julia_install_package_if_needed("Optim")
opt <- julia_pkg_import("Optim",
    func_list = c("optimize", "BFGS"))
rosenbrock <- function(x) (1.0 - x[1])^2 + 100.0 * (x[2] - x[1]^2)^2
result <- opt$optimize(rosenbrock, rep(0,2), opt$BFGS())
result
```

---

#### julia_setup

Do initial setup for the JuliaCall package.
Usage

julia_setup(
    JULIA_HOME = NULL,
    verbose = TRUE,
    install = TRUE,
    force = FALSE,
    useRCall = TRUE,
    rebuild = FALSE
)

Arguments

JULIA_HOME          the file folder which contains julia binary, if not set, JuliaCall will look at the
global option JULIA_HOME, if the global option is not set, JuliaCall will then
look at the environmental variable JULIA_HOME, if still not found, JuliaCall
will try to use the julia in path.
verbose             whether to print out detailed information about julia_setup.
install             whether to execute installation script for dependent julia packages, whose de-
default value is TRUE; but can be set to FALSE to save startup time when no
installation of dependent julia packages is needed.
force               whether to force julia_setup to execute again.
useRCall            whether or not you want to use RCall.jl in julia, which is an amazing package to
access R in julia.
rebuild             whether to rebuild RCall.jl, whose default value is FALSE to save startup time.
                      If a new version of R is used, then this parameter needs to be set to TRUE.

Value

The julia interface, which is an environment with the necessary methods like command, source and
things like that to communicate with julia.

Examples

## julia_setup is quite time consuming
julia <- julia_setup()

julia_source          Source a julia source file.

Description

julia_source sources a julia source file.
Usage

julia_source(file_name)

Arguments

file_name the name of julia source file.

plotsViewer Julia plots viewer in R.

Description

plotsViewer lets you view julia plots in R.

Usage

plotsViewer()

%>J% Language piper for julia language.

Description

The experimental language piper for julia language.

Usage

obj %>J% func_call

Arguments

obj the object to pass to the piper.

func_call the impartial julia function call.

Examples

## julia_setup is quite time consuming
2
3
Index

%>!, 14

autowrap, 2

call, 3

eng_juliacall, 4

field(JuliaObjectFields), 6
field<-(JuliaObjectFields), 6
fields(JuliaObjectFields), 6

julia_assign, 7
julia_call(call), 3
julia_command, 7
julia_console, 8
julia_do.call(call), 3
julia_eval, 8
julia_exists, 9
julia_function(julia_pkg_wrap), 12
julia_help, 9
julia_install_package(julia_package), 11
julia_install_package_if_needed
   (julia_package), 11
julia_installed_package
   (julia_package), 11
julia_library(julia_package), 11
julia_markdown_setup, 10
julia_notebook_setup, 10
julia_package, 11
julia_pkg_hook(julia_pkg_wrap), 12
julia_pkg_import(julia_pkg_wrap), 12
julia_pkg_wrap, 12
julia_setup, 12
julia_source, 13
julia_update_package(julia_package), 11
JuliaCall, 4
JuliaObject, 5
JuliaObjectFields, 6

plotsViewer, 14