Package 'XRJulia'

March 18, 2018

Type Package

Title Structured Interface to Julia

Version 0.7.7

Date 2018-03-02

Author John M. Chambers

Maintainer John Chambers <jmc@r-project.org>

Description A Julia interface structured according to the general
form described in package 'XR' and in the book ``Extending R''.

License GPL (>= 2)

Imports methods, XR

NeedsCompilation no

RoxygenNote 6.0.1

Repository CRAN

Date/Publication 2018-03-18 22:50:56 UTC

R topics documented:

XRjulia-package 2
asServerObjectMethods 2
findJulia 3
from_Julia-class 4
functions 4
juliaClassDef 6
JuliaFunction-class 7
JuliaInterface-class 8
JuliaObject-class 9
noScalar 9
proxyJuliaObjects 9
RJulia 10
setJuliaClass 10

Index **11**

XRjulia-package *Structured Interface to Julia*

Description

A Julia interface structured according to the general form described in package XR and the book (in progress) "Extending R"

Author(s)

John Chambers

Maintainer: John Chambers <jmc@stat.stanford.edu>

References

Chambers, John M. *Extending R* Chapman & Hall/CRC 2016.

asServerObjectMethods *Julia methods for asServerObject()*

Description

The default method for JuliaObject is modelled on the overall default method in XR.

For arrays, the method uses the reshape() function in Julia to create a suitable multi-way array.

For "plain" lists, the method produces the Julia expression for a list or a dictionary; with other attributes, uses the .RClass form.

Usage

```
## S4 method for signature 'ANY,JuliaObject'
asServerObject(object, prototype)

## S4 method for signature 'array,JuliaObject'
asServerObject(object, prototype)

## S4 method for signature 'list,JuliaObject'
asServerObject(object, prototype)
```

Arguments

object	The R object.
prototype	The proxy for a prototype of the Julia object, supplied by the evaluator.

`findJulia` *Find a Julia Executable*

Description

This function looks for an executable Julia application in the local operating system. The location can be prespecified by setting environment variable `JULIA_BIN`; otherwise, the function looks in various conventional locations and if that doesn't work, runs a shell command to look for `julia`.

Usage

```
findJulia(test = FALSE)
```

Arguments

`test`	Should the function test for the existence of the application. Default `FALSE`. Calling with `TRUE` is useful to bullet-proof examples or tests for the absence of Julia. If the test search succeeds, the location is saved in environment variable `JULIA_BIN`.

Value

The location as a character string, unless `test` is `TRUE`, in which case success or failure is returned, and the location found (or the empty string) is saved as the environment variable. Note that in this case, `FALSE` is returned if the Julia package `JSON` has not been added.

If `test` is `FALSE`, failure to find a Julia in the current system is an error.

On Mac OS X

Installing Julia in the usual way does not put the command line version in a standard location, but instead in a folder under `/Applications`. Assuming one wants to have Julia available from the command line, creating a symbolic link to it in `/usr/local/bin` is a standard approach. If the current version of Julia is `0.6`:

```
sudo ln -s /Applications/Julia-0.6.app/Contents/Resources/julia/bin/julia /usr/local/bin/julia
```

If for some reason you did not want this to be available, set the shell variable `JULIA_BIN` to the first file in the command, the one in `/Applications`.

from_Julia-class *Class for General Julia Composite Type Objects*

Description

The Julia side of the interface will return a general object from a composite type as an R object of class "from_Julia. its Julia fields (converted to R objects) can be accessed by the $ operator.

Slots

`serverClass` the Julia type.

`module` the Julia module, or ""

`fields` the converted versioin of the Julia fields; these are accessed by the $ operator.

functions *Function Versions of Methods for Julia Interface evaluators.*

Description

Function Versions of Methods for Julia Interface evaluators.

Usage

```
juliaSource(..., evaluator = RJulia())

juliaAddToPath(directory = "julia",
  package = utils::packageName(topenv(parent.frame())), pos = NA,
  evaluator = RJulia(.makeNew = FALSE), where = topenv(parent.frame()))

juliaUsing(module, evaluator)

juliaImport(..., evaluator)

juliaSend(object, evaluator = XR::getInterface(.JuliaInterfaceClass))

juliaGet(object, evaluator = XR::getInterface(.JuliaInterfaceClass))

juliaPrint(object, ..., evaluator = XRJulia::RJulia())

juliaEval(expr, ..., evaluator = XR::getInterface(.JuliaInterfaceClass))

juliaCommand(expr, ..., evaluator = XR::getInterface(.JuliaInterfaceClass))

juliaCall(expr, ..., evaluator = XR::getInterface(.JuliaInterfaceClass))
```

```
juliaSerialize(object, file, append = FALSE,
  evaluator = XR::getInterface(.JuliaInterfaceClass))

juliaUnserialize(file, all = FALSE,
  evaluator = XR::getInterface(.JuliaInterfaceClass))

juliaName(object)

juliaImport(..., evaluator)
```

Arguments

`...`	arguments to the corresponding method for an evaluator object.
`evaluator`	The evaluator object to use. By default, and usually, the current evaluator is used, and one is started if none has been. But see the note under `juliaImport` for the load actions created in special cases.
`directory`	the directory to add, defaults to "julia"
`package, pos`	arguments to the method, usually omitted.
`where`	for the load action, omitted if called from a package source file. Otherwise, must be the environment in which a load action can take place.
`module`	String identifying a Julia module.
`object`	A proxy in R for a Julia object.
`expr`	A string that should be legal when parsed and evaluated in Julia.
`file, append, all`	Arguments to the evalutor's serialize and unserialize methods. See the reference, Chapter 10.

Functions

- `juliaSource`: evaluate the file of Julia source.
- `juliaAddToPath`: adds the directory specified to the search path for Julia modules. If called from the source directory of a package during installation, sets up a load action for that package. If you want to add the path to all evaluators in *this* session, call the function before creating an evaluator. Otherwise, the action applies only to the specified evaluator or, by default, to the current evaluator.
- `juliaUsing`: the "using" form of Julia imports: the module is imported with all exports exposed.
- `juliaImport`: adds the module information specified to the modules imported for future Julia evaluator objects.

  Add the module to the table of imports for Julia evaluators, and import it to the current evaluator if there is one. If called from the source directory of a package during installation, both `juliaImport` and `juliaAddToPath()` set up a load action for that package. The functional versions, not the methods themselves, should be called from package source files to ensure that the load actions are created. Note that calling either function before any evaluator has been generated will install that call as a setup action for all XRJulia evaluators.

- `juliaSend`: sends the `object` to Julia, converting it via methods for `asServerObject` and returns a proxy for the converted object.
- `juliaGet`: converts the proxy object that is its argument to an R object.
- `juliaPrint`: Print an object in Julia. Either one object or several arguments as would be given to the Eval() method.
- `juliaEval`: evaluates the `expr` string subsituting the arguments. See the corresponding evaluator method for details.
- `juliaCommand`: evaluates the `expr` string subsituting the arguments; used for a command that is not an expression.
- `juliaCall`: call the function in Julia, with arguments given; expr is the string name of the function
- `juliaSerialize`: serialize the `object` in Julia
- `juliaUnserialize`: unserialize the file in Julia
- `juliaName`: return the name by which this proxy object was assigned in Julia
- `juliaImport`: Import a Julia module or add a directory to the Julia Search Path If called from the source directory of a package during installation, both `juliaImport` and `juliaAddToPath()` also set up a load action for that package. The functional versions, not the methods themselves, should be called from package source files to ensure that the load actions are created.

`juliaClassDef` *Information about a Julia Class*

Description

The Julia class definition information is computed, and converted to R.

Usage

```
juliaClassDef(Class, module = "", ..., .ev = RJulia())
```

Arguments

`Class, module`	Strings identifying the Julia composite type and optionally, the module containing it.
`..., .ev`	Don't supply these, `.ev` defaults to the current Julia interface evaluator.

Value

the Julia definition of the specified class, optionally from the module.

`JuliaFunction-class` *Proxy Objects in R for Julia Functions*

Description

A class and generator function for proxies in R for Julia functions.

Usage

```
JuliaFunction(...)

## S4 method for signature 'JuliaFunction'
initialize(name, module = "", evaluator =
RJulia(, ...))
```

Arguments

`name, module`	The name and module of the Julia function.
`evaluator`	The evaluator object to use. By default, and usually, the current evaluator is used, and one is started if none has been.
`...`	For `RJulia`, the arguments as interpreted by the initialize method, so typically `name` and optionally `module`. Remaining arguments are passed along to the next method.

Details

An object from this class is an R function that is a proxy for a function in Julia. Calls to the R function evaluate a call to the Julia function. The arguments in the call are converted to equivalent Julia objects; these typically include proxy objects for results previously computed through the XRJulia interface.

Slots

`name` the name of the server language function

`module` the name of the module, if that needs to be imported

`evaluatorClass` the class for the evaluator, by default and usually, `JuliaInterface`

Examples

```
if(findJulia(test = TRUE)) {
  set.seed(228)
  x <- matrix(rnorm(1000),20,5)
  xm <- juliaSend(x)
  svdJ <- JuliaFunction("svdfact")
  sxm <- svdJ(xm)
  sxm
}
```

JuliaInterface-class *An Interface to Julia*

Description

The JuliaInterface class provides an evaluator for computations in Julia, following the structure in the XR package. Proxy functions and classes allow use of the interface with no explicit reference to the evaluator. The function `RJulia()` returns an evaluator object.

Fields

`port,host` The parameters for communicating with the Julia evaluator.

`julia_bin` The command for starting a Julia process.

`connection` The connection object through which commands are sent to Julia

Methods

`Import(module, ...)` Import the module. The "Interface" method assumes a command "import" in the server language and does not handle any extra arguments (e.g., for importing specific members).

`initialize(...)` initializes the evaluator in a language-independent sense.

`ProxyClassName(serverClass)` If there is a proxy class defined corresponding to this serverClass, return the name of that class (typically pasted with the server langauge, separated by underscore). If no such class is defined, return NA.

`ServerClassDef(Class, module, ...)` Individual interface packages will define this to return a named list or other object such that value$fields and value$methods are the server fields and methods, character vectors of names or named objects whose elements give further information. This default version returns NULL, indicating that no metadata is available.

`ServerEval(expr, key, get)` Must be defined by the server language interface: evaluates ‘expr‘(a text string). If ‘key‘ is an empty string, ‘expr‘ is treated as a directive, with no defined value. Otherwise, ‘key‘ is a non-empty string, and the server object should be assigned with this name. The value returned is the R result, which may be an AssignedProxy() object. If ‘get‘ is TRUE or the value judged simple enough, it will be converted to an ordinary R object instead.

`ServerRemove(key)` Should be defined by the server language interface: The reference previously created for ‘key‘ should be removed. What happens has no effect on the client side; the intent is to potentially recover memory.

`ServerTask(task, expr, key = "", get = NA)` Call the task operation in the Julia code for the interface; the arguments must be the simple strings or logical value expected.

`Source(filename)` Parse and evaluate the contents of the file. This method is likely to be overriden for particular langauges with a directive to include the contents of the file. The ‘XR‘ version reads the file and processes the entire contents as a single string, newlines inserted between lines of the file.

`Using(...)` The Julia "using" form of importing. Arguments are module names. All the exported members of these modules will then be available, without prefix.

JuliaObject-class *Proxy Objects in R for Julia Objects*

Description

This is a class for all proxy objects from a Julia class with an R proxy class definition. Objects will normally be from a subclass of this class, for the specific Julia class.

Details

Proxy objects returned from the Julia interface will be promoted to objects from a specific R proxy class for their Julia class, if such a class has been defined.

noScalar *Send a Non-scalar Version of an Object*

Description

Ensures that an object is interpreted as a vector (array) when sent to the server language. The default strategy is to send length-1 vectors as scalars.

Usage

```
noScalar(object)
```

Arguments

object	A vector object. Calling with a non-vector is an error.

Value

the object, but with the S4 bit turned on. Relies on the convention that XR interfaces leave S4 objects as vectors, not scalars, even when they are of length 1

References

Chambers, John M. (2016) *Extending R*, Chapman & Hall/CRC. (Chapter 12, discussing this package, is included in the package: ../doc/Chapter_XR.pdf.)

proxyJuliaObjects *Proxy Objects in R for Julia Objects*

Description

Proxy Objects in R for Julia Objects

RJulia *An Evaluator for the Julia Interface.*

Description

Returns an evaluator for the Julia interface. Starts one on the first call, or if arguments are provided; providing argument `.makeNew = TRUE` will force a new evaluator. Otherwise, the current evaluator is returned.

Usage

```
RJulia(...)
```

Arguments

`...`	Arguments passed to `getInterface()` but none usually required. See `JuliaInterface` for details of the evaluator.

setJuliaClass *Define a Proxy Julia Class (Composite Type)*

Description

Given the name and optionally the module for a Julia composite type, defines an R proxy class with the same fields as the Julia type. By default, uses metadata from Julia to find the fields. If the call supplies the desired field names explicitly, metadata is not used.

Usage

```
setJuliaClass(juliaType, module = "", fields = character(),
  where = topenv(parent.frame()), proxyObjectClass = "JuliaObject", ...)
```

Arguments

`juliaType, module`	Strings identifying the composite type and optionally the module containing it. In normal use, metadata from Julia is used to find the definition of the type.
`fields, where, proxyObjectClass, ...`	Overriding arguments that should not be used by direct calls from package source code.

Index

∗Topic **package**
    XRjulia-package, 2

asServerObject, *6*
asServerObject,ANY,JuliaObject-method
        *(*asServerObjectMethods*)*, 2
asServerObject,array,JuliaObject-method
        *(*asServerObjectMethods*)*, 2
asServerObject,list,JuliaObject-method
        *(*asServerObjectMethods*)*, 2
asServerObjectMethods, 2

findJulia, 3
from_Julia-class, 4
functions, 4

getInterface, *10*

initialize,JuliaFunction-method
        *(*JuliaFunction-class*)*, 7

juliaAddToPath *(*functions*)*, 4
juliaCall *(*functions*)*, 4
juliaClassDef, 6
juliaCommand *(*functions*)*, 4
juliaEval *(*functions*)*, 4
JuliaFunction *(*JuliaFunction-class*)*, 7
JuliaFunction-class, 7
juliaGet *(*functions*)*, 4
juliaImport *(*functions*)*, 4
JuliaInterface, *7*, *10*
JuliaInterface *(*JuliaInterface-class*)*, 8
JuliaInterface-class, 8
juliaName *(*functions*)*, 4
JuliaObject *(*JuliaObject-class*)*, 9
JuliaObject-class, 9
juliaPrint *(*functions*)*, 4
juliaSend *(*functions*)*, 4
juliaSerialize *(*functions*)*, 4
juliaSource *(*functions*)*, 4
juliaUnserialize *(*functions*)*, 4
juliaUsing *(*functions*)*, 4

noScalar, 9

proxyJuliaObjects, 9

RJulia, 10

setJuliaClass, 10

XRjulia *(*XRjulia-package*)*, 2
XRjulia-package, 2