Package ‘J4R’

July 23, 2020

Type Package

Title Create 'Java' Objects and Execute 'Java' Methods

Version 1.0.8

Author Mathieu Fortin, Canadian Wood Fibre Centre, Canadian Forest Service

Maintainer Mathieu Fortin <mathieu.fortin.re@gmail.com>

Copyright Her Majesty the Queen in right of Canada

Description
Makes it possible to create 'Java' objects and to execute 'Java' methods from the 'R' environment. The 'Java' Virtual Machine is handled by a gateway server. Commands are sent to the server through a socket connection from the 'R' environment. Calls to 'Java' methods allow for vectors so that a particular method is iteratively run on each element of the vector. A score algorithm also makes the calls to 'Java' methods less restrictive. The gateway server relies on the runnable 'Java' library 'j4r.jar'. This library is licensed under the LGPL-3. Its sources are included in this package.

URL https://sourceforge.net/p/repiceasource/wiki/J4R/

Imports utils (>= 3.4), methods (>= 3.4)

License GPL-3

BugReports https://sourceforge.net/p/repiceasource/tickets/

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

SystemRequirements Java 8 or later

Suggests testthat

NeedsCompilation no

Repository CRAN

Date/Publication 2020-07-23 08:02:10 UTC

R topics documented:

`addToClassPath` *Dynamically adds a path or a jar file to the classpath.*

Description

This function makes it possible to add a directory or a JAR file to the class path. If the packageName parameter is null then the urlString parameter must be the complete path to the directory. Otherwise, it can be the name of the JAR file and the function will find the path through the package name. A non null packageName parameter is typically used in packages that rely on J4R.

Usage

```
addToClassPath(path, packageName = NULL)
```

Arguments

`path`	a character representing the path to the directory or the JAR file if the package-Name parameter is null. Otherwise, it can just be the name of the JAR file. This path is normalized so that expressions like myJar.jar or ./extensions/myJar.jar will be processed.
`packageName`	a character representing the package.

`addUrlToClassPath` *Dynamically adds an url to the classpath.*

Description

This function makes it possible to add a directory or a JAR file to the class path. If the packageName parameter is null then the urlString parameter must be the complete path to the directory. Otherwise, it can be the name of the JAR file and the function will find the path through the package name. A non null packageName parameter is typically used in packages that rely on J4R.

Usage

```
addUrlToClassPath(urlString, packageName = NULL)
```

Arguments

`urlString`	a character representing the complete path to the directory or the JAR file if the packageName parameter is null. Otherwise, it can just be the name of the JAR file.
`packageName`	a character representing the package.

Details

This function is deprecated. Use the addToClassPath function instead.

`as.float` *Cast the object into a Java float type*

Description

Cast the object into a Java float type

Usage

```
as.float(obj)
```

Arguments

`obj`	a numeric or a vector of numerics

`as.JavaArray` *Create a Java array from an R array*

Description

Converts an R array into a Java array.

Usage

```
as.JavaArray(values, affinity. = 1)
```

Arguments

`values`	a vector or a matrix
`affinity.`	an optional parameter for multithreading (see the mclapply.j4r function)

Value

a java.object reference that points a Java array

`as.long` *Cast the object into a Java long type*

Description

Cast the object into a Java long type

Usage

```
as.long(obj)
```

Arguments

`obj`	a numeric or a vector of numerics

`bufferLength` *Length of the buffer when reading from the socket connection.*

Description

The buffer has a length of 16Kb by default.

Usage

```
bufferLength
```

Format

An object of class `numeric` of length 1.

`cacheEnv` *The cache environment of this package*

Description

This environment contains the objects that enable the connection to the gateway server.

Usage

```
cacheEnv
```

Format

An object of class `environment` of length 0.

`callJavaGC` *Synchronize the Java environment with the R environment*

Description

This function call the garbage collector in R and sends the list of Java references that have been collected to the Java server. These references are then removed from the internal map.

Usage

```
callJavaGC()
```

See Also

J4R webpage

`callJavaMethod` *Call a Java method*

Description

This function calls a public method in a particular class of object. If the javaObject parameters or the additional parameters (...) include vectors, the method is called several times and a vector of primitive or a list of java instances can be returned.

Usage

```
callJavaMethod(source, methodName, ..., affinity = 1)
```

Arguments

`source`	this should be either a java.list instance or a single java.object instance for non-static methods or a string representing the Java class name in case of static method
`methodName`	the name of the method
`...`	the parameters of the method
`affinity`	a parameter used by the mclapply.j4r function in case of multithreading.

Details

There is no need to cast a particular parameter to a super class. Actually, the Java server tries to find the method that best matches the types of the parameters. Primitive type are converted on the fly, numeric to double, integer to int, logical to boolean and character to String. Factors are also converted to String.

When the source is a java.object instance, this function can be substituted for the $ operator.

Value

It depends on the method. It can return a primitive type (or a vector of primitive), a Java instance (or a list of Java instances) or nothing at all.

See Also

J4R webpage

Examples

```
### starting Java
connectToJava(memorySize = 200)

### creating an empty ArrayList object
myList <- createJavaObject("java.util.ArrayList")

### adding 3 to the list
callJavaMethod(myList, "add", 3)

### adding 5 to the list
myList$add(3)

### shutting down Java
shutdownJava()
```

`checkIfClasspathContains`
Check if a Library has been loaded

Description

It checks if a particular library is part of the classpath.

Usage

```
checkIfClasspathContains(myJavaLibrary)
```

Arguments

`myJavaLibrary`	a character string that stands for the java library (e.g. repicea.jar)

connectToJava — *Connect to Java environment*

Description

This function connects the R environment to a gateway server that runs in Java.

Usage

```
connectToJava(
  port = c(0, 0),
  extensionPath = NULL,
  memorySize = NULL,
  debug = FALSE
)
```

Arguments

`port`	a vector of the listening ports for the Java server
`extensionPath`	the path to jar files that can be loaded by the system classloader
`memorySize`	the memory size of the Java Virtual Machine in Mb (if not specified, the JVM runs with the default memory size)
`debug`	for debugging only (should be left as is)

Details

The first argument of the function provides the listening port for the Java server. A maximum of four ports is allowed. When set to 0, these ports are randomly selected. By default, the server listens to two random ports.

The extensionPath can either be set in this function or dynamically changed (see the addToClassPath function).

Value

a logical TRUE if the function managed to get connected to the server or if it was already connected or FALSE if the connection has failed

See Also

`addToClassPath`

`createJavaObject` *Create Java objects*

Description

This function creates one or many object of a particular class. If the parameters contain vectors, then a series of instances of this class can be created. Primitive type are converted on the fly, numeric to double, integer to int, logical to boolean and character to String. Factors are also converted to String.

Usage

```
createJavaObject(
  class,
  ...,
  isNullObject = FALSE,
  isArray = FALSE,
  affinity = 1
)
```

Arguments

`class`	the Java class of the object (e.g. java.util.ArrayList)
`...`	the parameters to be passed to the constructor of the object
`isNullObject`	a logical that indicates whether the instance should be null (by default it is set to FALSE)
`isArray`	a logical that indicates whether the instance is an array. By default, it is set to FALSE. When creating an array, the parameters must be integers that define the dimensions of the array
`affinity`	a parameter used by the mclapply.j4r function in case of multithreading.

Value

a java.object or java.list instance in the R environment

See Also

J4R webpage

Examples

```
### starting Java
connectToJava(memorySize = 200)

### creating an empty ArrayList object
createJavaObject("java.util.ArrayList")
```

```
### creating an ArrayList instance with initial capacity of 3
createJavaObject("java.util.ArrayList", as.integer(3))

### creating two ArrayList with different capacities
createJavaObject("java.util.ArrayList", c(as.integer(3), as.integer(4)))

### creating a 3x3 array of integers
myArray <- createJavaObject("int", 3, 3, isArray = TRUE)

### creating two arrays of integers with length 3
myArrays <- createJavaObject("int", c(3,3), isArray = TRUE)

### shutting down Java
shutdownJava()
```

`getAllValuesFromArray` *Returns all the elements of a Java array*

Description

All the elements of an array are returned. If these elements are Java instances, then the function value is a java.list of java.object references. Otherwise, the value is either a vector or a matrix

Usage

```
getAllValuesFromArray(object, affinity. = 1)
```

Arguments

`object`	a java.object reference pointing to a Java array
`affinity.`	an optional parameter for multithreading (see the mclapply.j4r function)

Value

either a java.list object, a vector or a matrix

`getAllValuesFromListObject` *Returns all the elements of a Java instance of List*

Description

All the elements of a Java List instance are returned.

Usage

```
getAllValuesFromListObject(object, affinity. = 1)
```

Arguments

`object`	a java.object that represents a List instance in Java
`affinity.`	an optional parameter for multithreading (see the mclapply.j4r function)

Value

either a java.list object or an R vector

`getArrayLength` *Return the length of an Array instance*

Description

This method returns an integer that is the length of the Array.

Usage

```
getArrayLength(object, affinity. = 1)
```

Arguments

`object`	a java.object instance that represents an array
`affinity.`	an optional parameter for multithreading (see the mclapply.j4r function)

Value

an integer that is the length of the array

`getClassLoaderPaths` *Retrieve the paths of the current classloader*

Description

This functions returns the paths that are currently included in the System classloader.

Usage

```
getClassLoaderPaths()
```

`getClassLoaderURLs` *Retrieve the URLs of the current classloader*

Description

This function returns the URLs that are currently included in the System classloader.

Usage

```
getClassLoaderURLs()
```

Details

This function is deprecated. Please use the getClassLoaderPaths instead.

`getJavaArchitecture` *Get Java architecture*

Description

Return the architecture of the Java installation, i.e. either 32-Bit or 64-Bit. It actually returns the second slot of the list produced by the getJavaVersion function.

Usage

```
getJavaArchitecture()
```

Value

the architecture, i.e. 32-Bit or 64-Bit

See Also

getJavaVersion

`getJavaField` *Get the value of a public field*

Description

This function gets the value of a particular field, which can be either static or not. If the field is static, the source should be a valid class name.

Usage

```
getJavaField(source, fieldName, affinity = 1)
```

Arguments

`source`	this should be either a java.list instance or a single java.object instance for non-static methods or a string representing the Java class name in case of static method
`fieldName`	the name of the field to be set
`affinity`	a parameter used by the mclapply.j4r function in case of multithreading.

Details

When the source is a java.object instance, this function can be substituted for the $ operator.

`getJavaVersion` *Get the current Java version*

Description

Returns the current Java version either through the command line if not connected to the Java server or through the Java server if connected.

Usage

```
getJavaVersion()
```

Value

a list with the first slot (version) being the version and the second slot (architecture) referring to the 32-Bit or 64-Bit architecture

See Also

getJavaArchitecture

getListOfJavaReferences

Provide a list of the Java references

Description

The function provides the list of the Java references in an environment.

Usage

```
getListOfJavaReferences(envir = .GlobalEnv)
```

Arguments

`envir`	the environment to be scanned for java.object and java.list instances. By default, it is the global environment

Details

By default this function provides the Java reference in the current environment. If there is no Java references then the value of the function is an empty list. If just.names is set to true, the value is a vector with the names of the instances. If false, then the function returns a list with the instances.

Value

a vector with the names of the instances

getMemorySettings

Returns the maximum, total and free memory in Mb

Description

This function calls the Runtime static methods maxMemory(), totalMemory() and freeMemory(). The results are divided by 1024 in order to report the memory sizes in Mb.

Usage

```
getMemorySettings()
```

Value

a data.frame object with the maximum, total and free memory in Mb.

getNbConnections *The number of connections to the server*

Description

The number of connections to the server

Usage

```
getNbConnections()
```

Value

the number of sockets connected to the server

getNbInstancesInInternalMap *Return the number of instances stored in the internal map of the Java server*

Description

Return the number of instances stored in the internal map of the Java server

Usage

```
getNbInstancesInInternalMap()
```

Value

an integer

`getValueFromArray` *Get a value from an array*

Description

This function returns the value at location given by the index parameter.

Usage

```
getValueFromArray(object, ..., affinity. = 1)
```

Arguments

`object`	a java.object that represents an array
`...`	a series of integers that correspond to the index of the value. Note that in Java the first index is 0
`affinity.`	an optional parameter for multithreading (see the mclapply.j4r function)

Value

the value at the location

`interruptJava` *Interrupt the current task on the Java server*

Description

Interrupt the current task on the Java server

Usage

```
interruptJava()
```

`is.JavaArray` *Check if the java.object instance represents an Array*

Description

This function returns true if the Java instance represented by this java.object is an Array.

Usage

```
is.JavaArray(object)
```

Arguments

`object`	a java.object instance

Value

a logical

`isConnectedToJava` *Checks if the Java server is running*

Description

This is done by checking f the socket connection to the JVM exists.

Usage

```
isConnectedToJava()
```

Value

a logical

`isJavaArray` *Check if the java.object instance represents an Array*

Description

This function returns true if the Java instance represented by this java.object is an Array.

Usage

```
isJavaArray(object)
```

Arguments

`object`	a java.object instance

Details

This function is deprecated. Please use the is.JavaArray instead.

`j4r.config.setDefaultJVMMemorySize` *Set a default memory size for the Java Virtual Machine*

Description

Allows to specify a default JVM size in Mb so that the option memorySize in hte connectToJava function does not need to be used.

Usage

```
j4r.config.setDefaultJVMMemorySize(defaultJVMMemory)
```

Arguments

`defaultJVMMemory`	the number of Mb for the JVM (must be equal to or greater than 50). If set to NULL, this option has no effect.

`j4r.config.setVerbose` *Enabling/disabling Verbose*

Description

It enables or diasble the verbose in hte J4R package. By default, the verbose is disabled.

Usage

```
j4r.config.setVerbose(verbose)
```

Arguments

`verbose`	a logical

`killJava` *Force the JVM to shut down*

Description

This is the not so gentle way to exit the JVM.

Usage

```
killJava()
```

Details

In case the JVM is stuck and does not respond to interrupt. It is possible to force the shutdown through this function.

`length.java.list` *Override the default length function*

Description

A java.list class is an environment containing an inner list. The length of this inner list is returned by this function.

Usage

```
## S3 method for class 'java.list'
length(x)
```

Arguments

`x`	a java.list instance

Value

the length of the inner list

`length.java.object` *Override the default length function*

Description

A java.object class is a list by definition. However, its length is 1.

Usage

```
## S3 method for class 'java.object'
length(x)
```

Arguments

`x`	a java.object instance

Value

1

`maxVectorLength` *Maximum length of the vector in the parameters.*

Description

A maximum length of the vector is set in order to avoid buffer size issues when reading

Usage

```
maxVectorLength
```

Format

An object of class `numeric` of length 1.

`mclapply.j4r` *Using multithreading with J4R*

Description

Applies the mclapply function in the context of the J4R package.

Usage

```
mclapply.j4r(X, FUN, ..., nbCores = getNbConnections())
```

Arguments

`X`	a vector of numerics
`FUN`	a two-argument function. The first argument is called by the mclapply function and the second argument defines the affinity and MUST be used in all the calls to the createJavaObject, callJavaMethod, getJavaField and setJavaField functions.
`...`	optional arguments to FUN (see mclapply)
`nbCores`	the number of threads to be used. By default, this argument is set to the number of available connections.

Details

Multithreading a function requires that the Java code is thread safe. The server must listen to at least two ports. Otherwise, this function will reduce to a single thread. Each port is given an affinity to an R thread.

The multithreading is not available on Windows. In such a case, the function will proceed in a single thread. The $ operator should not be used to substitute the getJavaField and setJavaField functions because it does not allow for the specification of the affinity. Use the original getJavaField and setJavaField functions. The $ operator can be used to call functions though as in the example below.

See Also

mclapply in the parallel package

`getNbConnections`

Examples

```
## Not run:
f <- function(i, aff) {
   myArrayList <- createJavaObject("java.util.ArrayList", affinity = aff)
   myArrayList$add(5, affinity = aff)
}

result <- mclapply.j4r(1:1000, f)
```

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

`print.java.list` *Print a java.list object*

Description

The java.object instances that are included in this list are displayed up to a maximum number.

Usage

```
## S3 method for class 'java.list'
print(x, ...)
```

Arguments

`x`	a java.list instance
`...`	additional parameters for consistent overriding

`print.java.object` *Print a java.object instance*

Description

The class name and the hashcode of the reference are displayed.

Usage

```
## S3 method for class 'java.object'
print(x, ...)
```

Arguments

`x`	a java.object instance
`...`	additional parameters for consistent overriding

`setJavaField` *Set the value of a public field*

Description

This function sets a particular field, which can be either static or not. If the field is static, the source should be a valid class name.

Usage

```
setJavaField(source, fieldName, value, affinity = 1)
```

Arguments

`source`	this should be either a java.list instance or a single java.object instance for non-static methods or a string representing the Java class name in case of static method
`fieldName`	the name of the field to be set
`value`	the new value of the field
`affinity`	a parameter used by the mclapply.j4r function in case of multithreading.

Details

When the source is a java.object instance, this function can be substituted for the $ operator.

`setJavaPath` *Set the path to Java*

Description

This is an option function that makes it possible to set the JAVA environment variable in R, if it is not already set. It first tests if the path ends with java or java.exe and if it is actually a file. Note that if an empty character is passed to this function. It resets the JAVA environment variable.

Usage

```
setJavaPath(path)
```

Arguments

`path`	the complete path to Java as in the example below. The file.path function should be used to define the path

See Also

file.path

Examples

```
myPath <- file.path("C:","Program Files (x86)","Java", "jre1.8.0_221", "bin", "java.exe")
# setJavaPath(myPath)  ### not run
```

`settingEnv` *The settings environment for this package*

Description

This environment contains the general settings of the package.

Usage

```
settingEnv
```

Format

An object of class `environment` of length 2.

`setValueInArray` *Set a value in an array*

Description

This function sets the value at the location given by the index parameter. It relies on the reflexive methods the Java class Array.

Usage

```
setValueInArray(object, value, index = NULL, affinity. = 1)
```

Arguments

`object`	a java.object that represents an array
`value`	the value to be set
`index`	the index of the location at which the value is set. Note that in Java the first index is 0. If this argument is set to NULL, then it is assumed that the value is set to index 0. In case of vectorization, the values are set from 0 to length(value) - 1 if this argument is left to NULL.
`affinity.`	an optional parameter for multithreading (see the mclapply.j4r function)

`shutdownJava` *Shut down Java*

Description

This function shuts down Java and the gateway server.

Usage

```
shutdownJava()
```

See Also

J4R webpage

Index