Package 'rscala'

April 5, 2020

Type Package

Title Bridge Between 'R' and 'Scala' with Callbacks

Version 3.2.19

Date 2020-04-04

URL https://github.com/dbdahl/rscala

BugReports https://github.com/dbdahl/rscala/issues

Imports utils

SystemRequirements Scala (>= 2.11), Java (>= 8)

Description 'Scala' <http://www.scala-lang.org/> is embedded in 'R' and callbacks from 'Scala' to 'R' are available. Support is provided to write 'R' packages that access 'Scala'. After installation, please run 'rscala::scalaConfig()'. The vignette provides an update of the original paper <doi:10.18637/jss.v092.i04>.

Depends R (>= 3.1.0)

LazyData TRUE

License Apache License 2.0 | file LICENSE

Encoding UTF-8

RoxygenNote 7.1.0

Suggests knitr, testthat, devtools, rstudioapi, processx

VignetteBuilder knitr

NeedsCompilation no

Author David B. Dahl [aut, cre]

Maintainer David B. Dahl <dahl@stat.byu.edu>

Repository CRAN

Date/Publication 2020-04-05 05:00:12 UTC

R topics documented:

`*.rscalaBridge` *Evaluation Operator*

Description

This operator compiles and executes a snippet of Scala code. All definitions are *local* to the supplied Scala snippet. Subsequent uses of the same code snippet skips the time-consuming compilation step. The return value is a vector or matrix of R's basic types (if possible) or an rscala reference (otherwise).

Usage

```
## S3 method for class 'rscalaBridge'
bridge * snippet
```

Arguments

`bridge`	A Scala bridge.
`snippet`	String providing a Scala code snippet.

Value

Returns a vector or matrix of R's basic types (if possible) or an rscala reference (otherwise).

See Also

^.rscalaBridge, +.rscalaBridge, scala

Examples

```
s <- scala()
s * 'scala.util.Random.nextDouble() <= 0.75'
s(mean=10, sd=2.5) * 'mean + sd * scala.util.Random.nextGaussian()'
close(s)
```

+.rscalaBridge *Declaration Operator*

Description

This operator compiles and executes a snippet of Scala code *in Scala's global environment*, where subsequent uses of the same code snippet do *not* skip the time-consuming compilation step and the return value is NULL. As such, this operator is used to define *global* imports, objects, classes, methods, etc.

Usage

```
## S3 method for class 'rscalaBridge'
bridge + snippet
```

Arguments

bridge	A Scala bridge.
snippet	String providing a Scala code snippet.

Value

Returns NULL, invisibly.

See Also

*.rscalaBridge, ^.rscalaBridge, scala

Examples

```
s <- scala()
s + '
  import scala.util.Random.nextInt
  import scala.math.{Pi, log, exp, sqrt}
  val const = -log(sqrt(2*Pi))
  def dnorm(x: Double, mean: Double, sd: Double, logScale: Boolean) = {
    val z = ( x - mean ) / sd
    val result = const - log(sd) - z * z / 2
    if ( logScale ) result else exp(result)
  }
'
s $ const()
s $ nextInt(100L)
s $ dnorm(8, 10, 2, FALSE)
close(s)
```

---

close.rscalaBridge *Close a Scala Bridge*

Description

Close a Scala Bridge

Usage

```
## S3 method for class 'rscalaBridge'
close(con, ...)
```

Arguments

con	A Scala bridge.
...	Currently ignored.

Value

Returns NULL, invisibly.

is.scalaReference *Test for Scala Reference*

Description

Test for Scala Reference

Usage

```
is.scalaReference(x)
```

Arguments

x	An arbitrary R object.

Value

Logical indicating whether x is an rscala reference.

Examples

```
is.scalaReference(c(1,2))
```

scala *Instantiate a Scala Bridge*

Description

This function creates an instance of a Scala bridge. Details on this function (and the rscala package as a whole) are provided in the package vignette. The original paper was published in the *Journal of Statistical Software*. See the reference below.

Usage

```
scala(
  JARs = character(),
  serialize.output = .Platform$OS.type == "windows",
  stdout = TRUE,
  stderr = TRUE,
  port = 0L,
  heap.maximum = NULL,
  command.line.arguments = character(0),
  debug = FALSE
)
```

Arguments

JARs	Character vector describing JAR files to include in the classpath. Elements are some combination of file paths to JARs or package names which contain embedded JARs. In the case of package names, the embedded JARs of all packages that recursively depend on, import, or suggest the specified package are also included.
serialize.output	Logical indicating whether Scala output should be serialized back to R. This is slower and probably only needed on Windows.
stdout	When serialize.output == FALSE, this argument influences where "standard output" results should be sent. TRUE or "" sends output to the R console (although that may not work on Windows). FALSE or NULL discards the output. Otherwise, this is the name of the file that receives the output.
stderr	Same as stdout, except influences the "standard error".
port	If 0, two random ports are selected. Otherwise, port and port+1 are used to the TCP/IP connections.
heap.maximum	String giving Scala's heap maximum, e.g., "8G" or "512M". The value here supersedes that from scalaMemory. Without this being set by either scala or scalaMemory, the heap maximum will be 90% of the available RAM.
command.line.arguments	A character vector of extra command line arguments to pass to the Scala executable, where each element corresponds to one argument.
debug	(Developer use only.) Logical indicating whether debugging should be enabled.

Details

Multiple interpreters can be created and each runs independently with its own memory space. Each interpreter can use multiple threads/cores, but the bridge between R and Scala is itself not thread-safe, so multiple R threads/cores should not simultaneously access the same bridge.

Terminate the bridge using close.rscalaBridge.

Value

Returns a Scala bridge.

References

David B. Dahl (2019). "Integration of R and Scala Using rscala." Journal of Statistical Software, 92:4, 1-18. https://www.jstatsoft.org

See Also

close.rscalaBridge, scalaMemory scalaPushRegister, scalaPullRegister

Examples

```
s <- scala()
rng <- s $ .new_scala.util.Random()
rng $ alphanumeric() $ take(15L) $ mkString(',')
s * '2+3'
h <- s(x=2, y=3) ^ 'x+y'
h $ toString()
s(mean=h, sd=2, r=rng) * 'mean + sd * r.nextGaussian()'
close(s)
```

---

scalaConfig *Configure Scala and Java*

Description

This function installs Scala and/or Java in the user's `~/.rscala` directory.

Usage

```
scalaConfig(
  verbose = TRUE,
  reconfig = FALSE,
  download = character(0),
  require.sbt = FALSE
)
```

Arguments

`verbose`	Should details of the search for Scala and Java be provided? Or, if a Scala bridge is provided instead of a logical, the function returns a list of details associated with the supplied bridge.
`reconfig`	If `TRUE`, the script `~/.rscala/config.R` is rewritten based on a new search for Scala and Java. If `FALSE`, the previous configuration is sourced from the script `~/.rscala/config.R`. If `"live"`, a new search is performed, but the results do not overwrite the previous configuration script. `"offline"` is the same as `"live"`, except no software is ever downloaded. Finally, the value set here is superceded by the value of the environment variable `RSCALA_RECONFIG`, if it exists.
`download`	A character vector which may be length-zero or whose elements are any combination of `"java"`, `"scala"`, or `"sbt"`. Or, `TRUE` denotes all three. The indicated software will be installed in "~/.rscala".
`require.sbt`	Should SBT be required, downloading and installing it in '~/.rscala/sbt' if necessary?

Value

Returns a list of details of the Scala and Java binaries.

References

David B. Dahl (2019). "Integration of R and Scala Using rscala." Journal of Statistical Software, 92:4, 1-18. https://www.jstatsoft.org

Examples

```
scalaConfig()
```

`scalaDevelDeployJARs` *Deploy JAR Files into the Package File System*

Description

This function copies the JAR files to the appropriate directories of the R package source. Specifically, source JAR files go into `(PKGHOME)/java` and binary JAR files go into `(PKGHOME)/inst/java/scala-(VERSION)`, where `(PKGHOME)` is the package home and `(VERSION)` is the major Scala version (e.g., 2.13).

Usage

```
scalaDevelDeployJARs(name, root, srcJAR, binJARs)
```

Arguments

`name`	The package name (as a string).
`root`	The file system path to package root directory (as a string).
`srcJAR`	The file system path to source JAR file (as a string).
`binJARs`	A named character vector of file system paths, where each name is a Scala major version (e.g., `"2.13"`.)

scalaDevelDownloadJARs *Download and Deploy JAR Files into the Package File System*

Description

This function only takes effect during package installation. It is meant to be called from bare code of a package that depends on **rscala** in a script such as `zzz.R`. When called during package installation, it downloads JAR files to the appropriate directories. This avoids the need to distribute some JAR files in the source package.

Usage

```
scalaDevelDownloadJARs(
  description,
  scalaMajorVersion = "",
  prefix = "https://search.maven.org/remotecontent?filepath="
)
```

Arguments

`description`	A character vector describing the JAR files to download, e,g. `"org.apache.commons:commons-math3:3.`
`scalaMajorVersion`	A Scala major version, such as `"2.13"`, if the JAR should be placed in a directory specific to a Scala version. Otherwise, `""` is used to specify a general installation.
`prefix`	A string giving the prefix of the download URL.

Examples

```
## Not run:
## To be run in bare code of a package that depends on rscala and needs,
## for example, the Apache Commons Math Library.
rscala::scalaDevelDownloadJARs("org.apache.commons:commons-math3:3.6.1")

## End(Not run)
```

scalaDisconnect *Temporarily Disconnect Scala by Closing Connections*

Description

This function temporarily disconnects a Scala bridge by closing its associated socket connections. The primary place where this function is used is at the end of examples of packages that depend on rscala (because, under some versions of R, "`R CMD check --as-cran`" does not permit connections to persist after an example ends).

Usage

```
scalaDisconnect(bridge = scalaFindBridge())
```

Arguments

bridge	A Scala bridge.

Examples

```
showConnections()
s <- scala()
showConnections()         # No additional connections yet.
s * "3+4"
showConnections()         # Now there are two additional connections.
scalaDisconnect()
showConnections()         # The new connections are gone.
s * "3+4"
showConnections()         # New connections are established as needed.
close(s)
```

---

scalaFindBridge *Find a Scala Bridge*

Description

This function attempts to find an instance of a Scala bridge based on an rscala reference or by searching the environment path.

Usage

```
scalaFindBridge(reference = NULL)
```

Arguments

reference	Either: i. An rscala reference, or ii. NULL (in which case the environment path is searched).

Value

A Scala bridge.

scalaLast *Retrieve the Last Scala Computation*

Description

This function retrieves the last result from the supplied Scala bridge.

Usage

```
scalaLast(bridge = scalaFindBridge())
```

Arguments

bridge	A Scala bridge

See Also

scalaFindBridge

Examples

```
s <- scala()
s * "2+3"
scalaLast(s)
close(s)
```

scalaLazy *Lazily Execute Functions on a Scala Bridge*

Description

Lazily Execute Functions on a Scala Bridge

Usage

```
scalaLazy(functions, bridge = scalaFindBridge())
```

Arguments

functions	A single function or list of functions. Each function takes a Scala bridge as its only argument. These functions are called immediately after the next time the bridge is connected. These functions are where setup code should go, like *global* imports, objects, classes, methods, etc. For example, it might equal `function(s) { s + 'import scala.util.Random' }`. **Note** the use of the declaration operator + instead of the operators * or ^.
bridge	A Scala bridge from the `scala` function.

Value

Returns NULL, invisibly.

See Also

scalaFindBridge

Examples

```
s <- scala()
scalaLazy(function(s) { s + 'import scala.util.Random' })
s$.new_Random()$nextDouble()
close(s)
```

---

scalaMemory *Get or Set Memory Available to Scala*

Description

Depending on the argument type, this function has several uses related to memory in Scala.

Usage

```
scalaMemory(x)
```

Arguments

x	If the argument is a string (e.g., "8G" or "512M"), the function sets the default maximum heap size for new instances of Scala bridges created by the function scala. If the argument is missing, the current default maximum heap size for new instances is returned. Set the argument to NULL to disable this global option, and therefore use **rscala**'s default. If the argument is a Scala bridge, the function returns a numeric vector giving the current heap size and the maximum heap size, in megabytes.

See Also

scala

Examples

```
## Not run:
scalaMemory("1G")

## End(Not run)
```

scalaPull *Push and Pull Objects Between R and Scala*

Description

The push function serializes an R object to Scala and the push function does the opposite. A couple of built push and pull methods are provided, namely `"generic"` and `"list"`. The `"generic"` method serializes an arbitrary R object to an instance of `RObject` in Scala. Since the `RObject` merely contains an array of bytes, the `RObject` is really only useful as storage for later unserialization. The `"generic"` method has an optional `as.is` argument which is either `TRUE` to cause the list to serialized as a single object or `FALSE` to cause each element of the list to the serialized individually. More methods may be added using the functions `scalaPushRegister` and `scalaPullRegister`.

Usage

```
scalaPull(reference, method, ...)

scalaPush(x, method = "generic", bridge = scalaFindBridge(), ...)
```

Arguments

`reference`	An rscala reference.
`method`	A string giving the specific 'push' or 'pull' method to use.
`...`	Other arguments passed to specialized push and pull functions.
`x`	An R object.
`bridge`	A Scala bridge.

See Also

`scalaPushRegister`, `scalaPullRegister`

Examples

```
s <- scala()

s(rn=scalaPush(rnorm),n=5) * 'R.evalD1("%-(%-)",rn,n)'

mtcarsRef <- scalaPush(mtcars, "list")
mtcarsRef$names()
mtcarsRef$mpg()
mtcars2 <- scalaPull(mtcarsRef, "list")
identical(mtcars, mtcars2)

# Oops, the variable names are bad...
tryCatch(ref <- scalaPush(iris, "list"), error=function(e) e)
```

```
# ... so let's clean up the variable names.
irisCleaned <- iris
names(irisCleaned) <- gsub("\\W","_",names(iris))
irisCleaned$Species <- as.character(iris$Species)
ref2 <- scalaPush(irisCleaned, "list")
scalaType(ref2)
ref2$Sepal_Length()
irisCleaned2 <- scalaPull(ref2,"list")
identical(irisCleaned, irisCleaned2)

close(s)
```

`scalaPushRegister` *Register Functions to Push and Pull Between R and Scala*

Description

The 'rscala' package provides support for serializing objects between R and Scala. These registration functions allows additional, more-specialized push and pull methods to be added. Package developers may want to call these registration functions in the package's `.onLoad` function.

Usage

```
scalaPushRegister(pusher, method, bridge = scalaFindBridge())

scalaPullRegister(puller, method, bridge = scalaFindBridge())
```

Arguments

`pusher`	A function whose first two arguments are as shown in the example below. Other arguments can be used as additional arguments.
`method`	A string giving the name of the specific 'push' or 'pull' method.
`bridge`	A Scala bridge.
`puller`	A function whose first two arguments are as shown in the example below. Other arguments can be used as additional arguments.

See Also

`scalaPush`, `scalaPull`

Examples

```
s <- scala()

name <- "Grace"
nameAsRObject <- scalaPush(name,"generic")   # Basic serialization
scalaType(nameAsRObject)
identical(name,scalaPull(nameAsRObject,"generic"))

scalaPush.character <- function(x, bridge) {
  if ( is.character(x) && ( length(x) == 1L ) ) bridge(x=x) ^ 'x'
  else stop("'x' should be a character vector.")
}
scalaPushRegister(scalaPush.character, "character")
nameAsString <- scalaPush(name, "character", s)    # More specific serialization
scalaType(nameAsString)

scalaPull.character <- function(reference, bridge) {
  if ( scalaType(reference) == "String" ) reference$toString()
  else stop("'reference' should be a 'String'.")
}
scalaPullRegister(scalaPull.character, "character")
identical(name,scalaPull(nameAsString,"character"))

close(s)
```

`scalaSBT` *Run SBT and Deploy JAR Files*

Description

This function helps developers of packages based on rscala. It runs SBT (Scala Build Tool) to package JAR files and then copy them to the appropriate directories of the R package source.

Usage

```
scalaSBT(
  args = c("+package", "packageSrc"),
  copy.to.package = TRUE,
  only.if.newer = TRUE
)
```

Arguments

`args`	A character vector giving the arguments to be passed to the SBT command.

`copy.to.package`
: Should the JARs files be copied to the appropriate directories of the R package source?'

`only.if.newer`
: Should compilation be avoided if it appears Scala code has not changed?

Details

Starting from the current working directory and moving up the file system hierarchy as needed, this function searches for the directory containing the file `'build.sbt'`, the SBT build file. It temporarily changes the working directory to this directory. It then runs `sbt +package packageSrc` to package the cross-compiled the Scala code and package the source code. publish the JAR files locally. Finally, it copies the JAR files to the appropriate directories of the R package source. Specifically, source JAR files go into `(PKGHOME)/java` and binary JAR files go into `(PKGHOME)/inst/java/scala-(VERSION)`, where `(PKGHOME)` is the package home and `(VERSION)` is the major Scala version (e.g., 2.13). It is assumed that the package home is a subdirectory of the directory containing the `'build.sbt'` file.

Note that SBT may give weird errors about not being able to download needed dependences. The issue is that some OpenJDK builds less than version 10 do not include root certificates. The solution is to either: i. manually install OpenJDK version 10 or greater, or ii. manually install Oracle's version of Java. Both are capable with the rscala package.

Value

`NULL`

Examples

```
## Not run:
scalaSBT()  # Working directory is the root of a package based on rscala.

## End(Not run)
```

`scalaType` *Get or Specify a Scala Type*

Description

This function gets the Scala type of an rscala reference. It also, together with the associated convenience objects, specifies a Scala type for transcompilation purposes.

Usage

```
scalaType(type)

stI0

stD0
```

stL0

stR0

stS0

stI1

stD1

stL1

stR1

stS1

stI2

stD2

stL2

stR2

stS2

Arguments

type	An rscala reference or a character vector of length one giving a Scala type.

Format

See 'Value' below.

An object of class rscalaType of length 1.

An object of class rscalaType of length 1.

An object of class rscalaType of length 1.

An object of class rscalaType of length 1.

An object of class rscalaType of length 1.

An object of class rscalaType of length 1.

An object of class rscalaType of length 1.

An object of class rscalaType of length 1.

An object of class rscalaType of length 1.

An object of class rscalaType of length 1.

An object of class rscalaType of length 1.

An object of class rscalaType of length 1.

An object of class rscalaType of length 1.

An object of class rscalaType of length 1.

An object of class rscalaType of length 1.

Details

The convenience objects are of the form stXY (where X is in {I,D,L,R,S} and Y is in {0,1,2}) as as indicated below:

- I corresponds to Scala's Int and R's integer.
- D corresponds to Scala's Double and R's double.
- L corresponds to Scala's Boolean and R's logical.
- R corresponds to Scala's Byte and R's raw.
- S corresponds to Scala's String and R's character.

- 0 corresponds to a Scala primitive and an R length one vector.
- 1 corresponds to a Scala array and an R vector.
- 2 corresponds to a Scala array of arrays and an R matrix.

For example, stS2 is equivalent to Scala's scalaType("Array[Array[String]]") and R's type for matrix(character()). Also, stL1 is equivalent to Scala's scalaType("Boolean") and R's type for logical(1).

Value

An object of class rscalaType whose value is a character vector of length one indicating a Scala type.

Examples

```
scalaType("Double")
stD0
scalaType("Array[Byte]")
stR1
scalaType("Array[Array[Int]]")
stI2
```

scalaVersionJARs *JAR Files for Support Scala Versions*

Description

This function returns a named list whose elements give the file system paths of the JAR files for the supported major versions of Scala.

Usage

```
scalaVersionJARs()
```

Value

A list whose names correspond to Scala major versions and whose elements are file system paths.

Examples

```
scalaVersionJARs()
```

^.rscalaBridge *Evaluation Operator Returning a Reference and Transcompile Operator*

Description

This operator is equivalent to `*.rscalaBridge`, except the return value is always an rscala reference. This operator also allows (a small subset of) R code to be transcompiled to Scala code and produces an rscala reference to an anonymous Scala function.

Usage

```
## S3 method for class 'rscalaBridge'
bridge ^ snippet
```

Arguments

`bridge`	A Scala bridge.
`snippet`	String providing a Scala code snippet.

Value

Returns an rscala reference.

See Also

*.rscalaBridge, +.rscalaBridge, scala

Examples

```
s <- scala()
x <- s ^ 'new scala.util.Random()'  # These two lines ...
x <- s $ .new_scala.util.Random()   # ... are equivalent
s(rng=x) * 'rng.nextDouble()'
f <- s ^ function(x=scalaType('Double')) { pi - x }
f$apply(3.14)
s(n=10L, mapper=s ^ function(x=scalaType("Int")) { 2 * x }) * "Array.tabulate(n)(mapper)"
logStdNormalDensity <- s ^ function(x=scalaType("Double"), mean=0.0, sd=1.0) {
  variance <- sd^2
  -0.5*log(2*pi*variance) - 0.5/variance * (x-mean)^2
}
identical(logStdNormalDensity$apply(1.0), dnorm(1.0, log=TRUE))
close(s)
```

Index