Package 'jvmr'

August 30, 2014

Type Package

Title Integration of R, Java, and Scala

Version 2.11.2.1

Date 2014-08-30

Author David B. Dahl [aut, cre], Scala developers [ctb] (see http://scala-lang.org/), beanshell2 developers [ctb] (see https://code.google.com/p/beanshell2/), jline2 developers [ctb] (see https://github.com/jline/jline2)

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

URL http://dahl.byu.edu/software/jvmr/

Imports rJava

SystemRequirements Java (>= 1.6)

Description Cross-platform, self-contained, and bi-directional interface between R and Scala, Java, and other JVM-based languages.

License GPL (>= 2) | BSD_3_clause + file LICENSE

NeedsCompilation no

Repository CRAN

Date/Publication 2014-08-30 23:14:58

R topics documented:

`interpret` *Run Java or Scala code in an interpreter embedded in R.*

Description

These functions are the main interface to the Java and Scala interpreters embedded in R. Through these functions, code in Java or Scala can be executed and data can be passed between R and these languages. There are shortcuts using square brackets for each of these functions. See the examples below.

Usage

```
interpret(interpreter, code, ..., eval.only = FALSE, simplify = TRUE, echo.output = FALSE)
interpret(interpreter, varname, type = NULL, echo.output = FALSE) <- value
```

Arguments

`interpreter`	an instance of a Java or Scala interpreter created by the `scalaInterpreter` or `javaInterpreter` functions.
`code`	a character vector of length one to be evaluated by the Java or Scala interpreter. This vector can contain many statements and span multiple lines (by using triple quotes in Scala).
`varname`	name of a Java or Scala object to be assigned `value`.
`...`	specifies up to nine substitutions to be made in the Java or Scala code. These substitutions must take the form `${d}` in the Java or Scala code, where the number `d` is the argument number 1, 2, ..., 9.
`eval.only`	When TRUE, no object is returned. If FALSE (the default value), the result of the evaluation is returned.
`simplify`	When TRUE (the default value), an R object is returned when possible. If FALSE, the reference of the resulting Java object is returned.
`echo.output`	When FALSE (the default value), output from the Scala interpreter is not printed to the R session. If TRUE, such output is printed in the R session. This is not relevant for the Java interpreter.
`type`	an optional argument, which can be used to explicitly specify the type of the Java or Scala object.
`value`	the value assigned to the Scala or Java object whose name is given by the `varname` argument.

Author(s)

David B. Dahl

See Also

`scalaInterpreter`, `javaInterpreter`, `jvmr-package`.

Examples

```
library(jvmr)

###############
## ScalaInR ##
###############

## Not run:
# Creating an instance of the Scala interpreter
a <- scalaInterpreter()

# Defining functions
interpret(a,'
  def fib( n: Int): Int = n match {
    case 0 | 1 => n
    case _ => fib( n -1) + fib( n-2)
  }
')

fib <- function(n){
  if( n > 1 ) fib(n-1) + fib(n-2)
  else n
}

system.time(a['fib(33)'])
system.time(fib(33))

# Defining a variable
interpret(a,"num") <- 3.4
# Printing the value of the variable
interpret(a,"num")

# Defining a variable using shortcut syntax
a["x"] <- "Welcome"
# Printing the value of the variable using shortcut syntax
a["x"]

# Using loops
interpret(a,"message") <- c("Hello","World","!")
interpret(a,"message")
interpret(a,'message.foreach{ x => println("<"+x+">") }',echo.output=TRUE)


# Assigning values to variables using the argument list
interpret(a,'
  val d = "Zero"
  val b = "${1}"
  val d2 = "Two"
',"One")

# Illustrating the use of 'simplify'
c <- a['Range(0,100).toArray']
```

```
c

d <- a['Range(0,100).toArray',simplify=FALSE]
d

# Illustrating the use of 'type'
library(rJava)
random <- a["new java.util.Random"]     # Can infer type

my.list <- a['List[Double](1.0, 2.3, 5.6)']
\dontrun{
a["myList"] <- my.list                  # Cannot infer type this time
}
a["myList","List[Double]"] <- my.list   # Must explicity specify type


# Illustrating the cost of the high level bridge
a["val who = 'David'"]

system.time(
  for ( i in 1:100 ) { cat(interpret(a,"who"),"\n") }
)

system.time(
  interpret(a,'for ( i <- 0 until 1000 ) println(who)')
)

## End(Not run)


#############
## JavaInR ##
#############

## Not run:
# Create an instance of the Java interpreter
b <- javaInterpreter()

# Defining functions
interpret(b,'
  int fib(int n) {
    if ( ( n == 0 ) || ( n == 1 ) ) return(n);
    return( fib(n-1) + fib(n-2) );
  }
')


fib <- function(n) {
  if ( n > 1 ) fib(n-1) + fib(n-2)
  else n
}

system.time(b['fib(33)'])
```

```
system.time(fib(33))


# Defining a variable
interpret(b,"number") <- 3.4
# Printing the value of the variable
interpret(b,"number")

# Defining a variable using shortcut syntax
b['int[] c = {0,1,2,3,4,5,6}']
# Printing a variable using shortcut syntax
b['c']

# Pulling values from Java into R
c <- b['c']
c

# Use of simplify
d <- b['c',simplify=FALSE]
d

# Using loops
interpret(b,"message2") <- c("Hello","World","Again","!")
interpret(b,"message2")
interpret(b,'for(int i = 0; i < message2.length; i++) {
  System.out.println("<" + message2[i] + ">");
}',echo.output=TRUE)

# Assigning values to variables using the argument list
b['
  String arg1 = "${1}";
  String arg2 = "${2}";
  String statement = "Skies" + arg1 + arg2;
"," are ","blue"]
b['statement']

# Illustrating the cost of the high level bridge
b['String who = "David"']

system.time(
  for ( i in 1:1000 ) { cat(interpret(b,"who"),"\n") }
)

system.time(
  interpret(b,'for ( i=0; i < 1000; i++) java.lang.System.out.println(who)')
)

## End(Not run)
```

Description

Integrates the Java and Scala interpreters in R. This provides access to existing Java and Scala libraries. This package also contains the JAR files which allow R to be embedded in Java and Scala programs.

Usage

```
.jvmr.jar
.jvmr.alljars
.jvmr.alljars.vector
.bsh.jar
```

Value

.jvmr.jar is a character vector giving the location of the jvmr JAR file. This must be in the classpath when embedding R in Scala.

.jvmr.alljars is a character vector of length 1 containing the paths to all the JARs from the jvmr package, seperated according to `.Platform$path.sep`. All these JARs must be in the classpath when embedding R in Java.

.jvmr.alljars.vector is a character vector containing the paths to all the JARs from the jvmr package. All these JARs must be in the classpath when embedding R in Java.

.bsh.jar is a character vector on length 1 containing the path of the BeanShell JAR.

Author(s)

David B. Dahl

See Also

`scalaInterpreter`, `javaInterpreter`, `interpret`, `interpret<-`.

`scalaInterpreter` *Creates an instance of a Scala or Java interpreter embedded in the R session.*

Description

Creates an instance of a Scala or Java interpreter embedded in the R session.

Usage

```
scalaInterpreter(..., use.jvmr.class.path = TRUE, include.cwd = TRUE,
                 use.java.class.path = FALSE)
javaInterpreter(..., use.jvmr.class.path = TRUE, include.cwd = TRUE)
```

Arguments

`...`	Specifies additional elements for the classpath of the interpreter. This only has an effect if `use.java.class` is FALSE.
`use.jvmr.class.path`	If TRUE (the default value), the JAR files specified in the global option `jvmr.class.path` will be included in the classpath. This only has an effect if `use.java.class` is FALSE.
`include.cwd`	If TRUE (the default value), the current working directory will be included in the classpath. These class paths will only be included if `use.java.class` is FALSE.
`use.java.class.path`	If TRUE, the default java classpath will be used and all other arguments are ignored.

Author(s)

David B. Dahl

See Also

`jvmr-package`, `interpret`, `interpret<-`.

Examples

```
library(jvmr)

## Not run:
# Creating an instance of the Scala interpreter
a <- scalaInterpreter()

# Create an instance of the Java interpreter
b <- javaInterpreter()

## End(Not run)
```

Index