Package 'RWebServices'

April 23, 2016

Type Package

Title Expose R functions as web services through Java/Axis/Apache

Version 1.34.0

Author Nianhua Li, MT Morgan

Maintainer Martin Morgan <mtmorgan@fredhutch.org>

Description This package provides mechanisms for automatic function prototyping and exposure of R functionality in a web services environment.

Depends SJava (>= 0.85), TypeInfo, methods, tools (>= 2.10.0), R (>= 2.5.0)

Imports RCurl

License file LICENSE

License_restricts_use no

biocViews Infrastructure

NeedsCompilation yes

R topics documented:

`RWebServices-package` *Expose R functions as web services through Java/Axis/Apache*

Description

This package provides mechanisms for automatic function prototyping and exposure of R functionality in a web services environment.

Details

Package:	RWebServices
Type:	Package
Version:	1.0
Date:	2005-12-23
License:	What license is it under?

Author(s)

Written by Nianhua Li, MT Morgan. Maintainer: Robert Gentleman <rgentle@fhcrc.org>

See Also

The `TypeInfo` package.

`ArrayAndMatrix-class` *Class "NumericMatrix" and friends*

Description

Classes providing `matrix` and `array` functionality, but allowing stronger type specification, e.g., `NumericMatrix` is a `matrix` whose members must be of type "numeric". A "matrix" is two-dimensional; an "array" is any dimensions, but R treats two-dimensional arrays as "matrix" and one-dimensional arrays as "vector" so "XxxArray" is most useful for specifying arrays with greater than 2 dimensions.

Available classes are "RawMatrix" "CharMatrix" "LogicalMatrix" "IntegerMatrix" "NumericMatrix" "ComplexMatrix" "RawArray" "CharArray" "LogicalArray" "IntegerArray" "NumericArray" "ComplexArray".

Details

The following illustrates how to use these classes in R and Java, using `NumericMatrix` as an example.

The examples assume that mapping between R and Java uses the "javalib" type mode.

Suppose you would like an R function to return a matrix of numeric (double) values, and that you are using the "javalib" mode for mapping between R and Java (`typeMode="javalib"` in `createMap`, or `typemode=javalib` in the ant configuration file `RWebServicesTuning.properties`). For this type mode, specifying "matrix" as a return type is insufficient: a matrix could contain any basic type (raw, character, integer, double, complex, etc.), so there is not enough information to map to a Java object. The solution is to specify more clearly what the type of the return value is, for instance, if the return type is a matrix of numeric (double) values, then arrange for the function to return an object of class `NumericMatrix`.

In R the basic properties of `NumericMatrix` can be seen here:

```
    > nm=new("NumericMatrix", matrix(as.double(50:1),10,5))
> dim(nm)
[1] 10  5
> nm[7,3]
[1] 24
> as.vector(nm)
 [1] 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33
[19] 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15
[37] 14 13 12 11 10  9  8  7  6  5  4  3  2  1
> as.vector(nm)[(3-1)*nrow(nm)+7]
[1] 24
```

The last line is meant to illustrate that the 'matrix' data in R is stored as a single vector, in 'column-major' order, i.e., elements `1:nrow(nm)` are the first column, `(nrow(nm)+1):(2*nrow(nm))` are the second column, etc.

In Java, `dim=getDim()` returns an `int[]`, containing the number of rows and the number of columns, e.g., `int[] = {10, 5}` for the example above of columns in the matrix (like `dim()` would do on a matrix in R).

`value=getValue()` will return `double[]`, containing all the data, like the result of as.vector above. Remembering that Java starts indexing at 0, we can get the element in row i and column j as

```
value[i * dim[0] + j]
```

We (or you) could write methods getMatrix / setMatrix returning / taking double[][] to do this, or getElement / setElement to access specific elements; it's worthwhile remembering that this does require quite a bit of memory allocation (for big arrays), so in some ways it's better to force the client to just give the data in the expected format anyway.

Objects from the Class

Objects can be created by calls of the form `new("NumericMatrix",...)`. To create an instance from an existing, untyped, "matrix" or "array", use forms like `new("NumericMatrix", m)` or unclass and update the `mode` of the object. See examples for additional detail.

Slots

None.

Extends

Class `"matrix"`, from data part. Class `"structure"`, by class "matrix", distance 2. Class `"array"`, by class "matrix", distance 2. Class `"vector"`, by class "matrix", distance 3, with explicit coerce. Class `"vector"`, by class "matrix", distance 4, with explicit coerce.

Methods

Use "NumericMatrix" and friends as replacements for "matrix" or "array".

Author(s)

Martin Morgan <mtmorgan@fhcrc.rog

Examples

```
## creating an empty instance
obj <- new("ComplexMatrix")
obj
mode(obj)
## Converting between array and typed Array
new("RawArray", array(raw(), dim=c(5,2,3)))

## Converting between types
obj <- new("NumericMatrix", matrix(numeric()))
obj <- unclass(obj)                    # retrieve underlying matrix
mode(obj) <- "raw"
obj <- new("RawMatrix", obj)
validObject(obj)
obj
```

---

converters *Convert between R objects and their Java representation.*

Description

These functions are not intended for the end user. They are used to match objects being translated between R and Java to functions that perform the conversion. They are useful for understanding the overall conversion process, and as examples for writing custom conversions.

Usage

```
# 'robject' converters
cvtCharacterFromJava(x, thisClassName)
cvtCharacterToJava(x, ...)
matchCharacterToJava(x, ...)

cvtIntegerFromJava(x, thisClassName)
cvtIntegerToJava(x, ...)
matchIntegerToJava(x, ...)

cvtLogicalFromJava(x, thisClassName)
cvtLogicalToJava(x, ...)
matchLogicalToJava(x, ...)

cvtNumericFromJava(x, thisClassName)
cvtNumericToJava(x, ...)
matchNumericToJava(x, ...)

cvtRawFromJava(x, thisClassName)
cvtRawToJava(x, ...)
matchRawToJava(x, ...)

cvtComplexFromJava(x, thisClassName)
cvtComplexToJava(x, ...)
matchComplexToJava(x, ...)

cvtVectorFromJava(x, thisClassName)
cvtVectorToJava(x, ...)
matchVectorToJava(x, ...)

cvtListFromJava(x, thisClassName)
cvtListToJava(x, ...)
matchListToJava(x, ...)

cvtArrayFromJava(x, thisClassName)
cvtArrayToJava(x, ...)
matchArrayToJava(x, ...)

cvtMatrixFromJava(x, thisClassName)
cvtMatrixToJava(x, ...)
matchMatrixToJava(x, ...)

cvtFactorFromJava(x, thisClassName)
cvtFactorToJava(x, ...)
matchFactorToJava(x, ...)

cvtDataFrameFromJava(x, thisClassName)
cvtDataFrameToJava(x, ...)
```

```
matchDataFrameToJava(x, ...)

cvtEnvFromJava(x, thisClassName)
cvtEnvToJava(x, ...)
matchEnvToJava(x, ...)

cvtUnknownFromJava(x, thisClassName)
cvtUnknownToJava(x, ...)
matchUnknownToJava(x, ...)

cvtFileReferencesFromJava(x, thisClassName)
cvtFileReferencesToJava(x, ...)
matchFileReferencesToJava(x, ...)

## 'javalib' converters
cvtCharacterToJava2(x, ...)
matchCharacterToJava2(x, ...)

cvtIntegerToJava2(x, ...)
matchIntegerToJava2(x, ...)

cvtLogicalToJava2(x, ...)
matchLogicalToJava2(x, ...)

cvtNumericToJava2(x, ...)
matchNumericToJava2(x, ...)

cvtRawToJava2(x, ...)
matchRawToJava2(x, ...)

cvtComplexFromJava2(x, thisClassName)
cvtComplexToJava2(x, ...)
matchComplexToJava2(x, ...)

cvtListToJava2(x, ...)
matchListToJava2(x, ...)

cvtFactorFromJava2(x, thisClassName)
cvtFactorToJava2(x, ...)
matchFactorToJava2(x, ...)

cvtDataFrameFromJava2(x, thisClassName)
cvtDataFrameToJava2(x, ...)
matchDataFrameToJava2(x, ...)

cvtCharArrayFromJava2(x, thisClassName)
cvtCharArrayToJava2(x, ...)
matchCharArrayToJava2(x, ...)
```

```
cvtIntegerArrayFromJava2(x, thisClassName)
cvtIntegerArrayToJava2(x, ...)
matchIntegerArrayToJava2(x, ...)

cvtNumericArrayFromJava2(x, thisClassName)
cvtNumericArrayToJava2(x, ...)
matchNumericArrayToJava2(x, ...)

cvtLogicalArrayFromJava2(x, thisClassName)
cvtLogicalArrayToJava2(x, ...)
matchLogicalArrayToJava2(x, ...)

cvtRawArrayFromJava2(x, thisClassName)
cvtRawArrayToJava2(x, ...)
matchRawArrayToJava2(x, ...)

cvtComplexArrayFromJava2(x, thisClassName)
cvtComplexArrayToJava2(x, ...)
matchComplexArrayToJava2(x, ...)

cvtCharMatrixFromJava2(x, thisClassName)
cvtCharMatrixToJava2(x, ...)
matchCharMatrixToJava2(x, ...)

cvtIntegerMatrixFromJava2(x, thisClassName)
cvtIntegerMatrixToJava2(x, ...)
matchIntegerMatrixToJava2(x, ...)

cvtNumericMatrixFromJava2(x, thisClassName)
cvtNumericMatrixToJava2(x, ...)
matchNumericMatrixToJava2(x, ...)

cvtLogicalMatrixFromJava2(x, thisClassName)
cvtLogicalMatrixToJava2(x, ...)
matchLogicalMatrixToJava2(x, ...)

cvtRawMatrixFromJava2(x, thisClassName)
cvtRawMatrixToJava2(x, ...)
matchRawMatrixToJava2(x, ...)

cvtComplexMatrixFromJava2(x, thisClassName)
cvtComplexMatrixToJava2(x, ...)
matchComplexMatrixToJava2(x, ...)

cvtEnvFromJava2(x, thisClassName)
cvtEnvToJava2(x, ...)
matchEnvToJava2(x, ...)
```

```
cvtFileReferencesFromJava2(x, thisClassName)
cvtFileReferencesToJava2(x, ...)
matchFileReferencesToJava2(x, ...)
```

Arguments

x	Either a reference to the Java object to be converted (e.g. cvtArrayFromJava), or a R object.
...	Additional arguments passed to SJava
thisClassName	Character representation of the class of the Java object that x refers to.

Details

The cvt* functions take an instance of an object of one language (R or Java) and convert it to an instance of the other language.

The match* functions take an instance of an object, and return a logical indicating whether the match criterion is satisified.

The convention used in RWebServices is to name the functions as <RObject>ToJava or <RObject>FromJava.

RWebServices installs one of two types of converters. The the function regAddonCvt installs converters for the 'robject' model, where R objects are represented in a hierarchy of Java objects that attempt to capture important attributes of the R instances. The mapping is as follows:

R	Java
raw	RRaw
logical	RLogical
character	RChar
integer	RInteger
numeric	RNumeric
complex	RComplex
list	RList
factor	RFactor
data.frame	RDataFrame
environment	REnvironment
array	RArray
matrix	RMatrix
'other'	RUnknown

The 'other' converter is invoked when no other converter matches. cvtUnknownToJava creates an instance of the Java class rservices.RUnknown to hold the class, length, and string representation (i.e., the result of print) of the contents of the R object. cvtUnknownFromJava converts rservices.RUnknown instance to a R S4 instance if the original R object is a S4 instance, otherwise the function returns a list with the same class and length as the original R object. This is likely to be very unsatisfactory.

The function regAddonCvt2 installs converters for the 'javalib' model, where R objects are represented by Java primitive types where possible; NA values are not permitted. The mapping is as follows:

R	Java
raw	byte[]
logical	boolean[]
character	String[]
integer	int[]
numeric	double[]
complex	RJComplex
list	Object[]
factor	RJFactor
data.frame	RJDataFrame
environment	java.util.HashMap
'*'Array,	RJ'*'Array
'*'Matrix,	RJ'*'Matrix
FileReferences	RJFileReferences

The '*'Array and '*'Matrix and array entries represent instances of the ArrayAndMatrix-class, which provide strong type information about the typeof elements (raw, logical, character, integer, numeric, complex) in arrays and matricies.

Value

The cvt*ToJava functions return references to Java objects. The cvt*FromJava functions return instances of R objects. match* return TRUE when the match criterion is satisfied, FALSE otherwise.

Author(s)

Nianhua Li

References

http://www.omegahat.org/RSJava/index.html

See Also

setJavaFunctionConverter, regAddonCvt, regAddonCvt2

---

createMap *Create Java function signatures from R functions*

Description

createMap extracts type information from R function definitions, and uses this to create Java-style function calls with appropriately typed arguments. Types are then converted to Java objects.

A side effect is to create or modify a directory structure in the current working directory containing Java class information.

There are three methods defined:

`createMap(standardGeneric, missing, ...)` Intended to dispatch on S4 generics, creating maps for all currently visible methods.

`createMap(character, missing, ...)` Intended to dispatch on a character vector of function names, creating maps for each function.

`createMap(missing, pkgs=character, ...)` Inteded to map all functions in `pkgs` with `typeInfo` applied. `pkgs` can be a character vecorr of package names, or in conjunction with the argument `splitPkgsToVector=TRUE` a comma-delimited list of packages. The latter behavior is meant to faciliate use with the `ant` command line, as `ant -Dpkg=caDNAcopy,caPROcess gen-map-from-package`.

Usage

```
createMap(funcs, pkgs, generateTests = TRUE,
          outputDirectory = stop("specify outputDirectory"),
          typeMode = "javalib", deployMode = "jms",
          verbose = FALSE, ...)
## S4 method for signature 'missing,character'
createMap(funcs, pkgs, generateTests = TRUE,
          outputDirectory = stop("specify outputDirectory"),
          typeMode = "javalib", deployMode = "jms",
          verbose = FALSE, splitPkgsToVector=TRUE, ...,
          extraClasses=character())
```

Arguments

`funcs`	Function or list of functions with typeInfo already applied, or S4 generic method.
`pkgs`	Character string or vector of package names. Package namespaces are parsed for functions with TypeInfo, and R-Java maps are created for these.
`generateTests`	Logical. Should test code be generated?
`outputDirectory`	Directory where Java hierarchy will be created.
`typeMode`	Character, either "robject" or "javalib". How Java objects should be generated; see `generateDataMap` and `converters`. `typemode` may be over-written by values in `system.file("rservices", "properties.R", package=pkg)`.
`deployMode`	Character, either "demo" or "jms". How the service will be deployed. "demo" is no longer supported.
`verbose`	Print information during mapping generationg; useful for debugging.
`...`	Additional arguments, e.g., `S4DefaultTypedSig` used for providing type specifications for S4 functions when `funcs` is of class `standardGeneric`, or `splitPkgsToVector=TRUE` to treat `pkgs` as a comma-delimited list of packages.
`splitPkgsToVector`	Parse `character(1)` pkgs as `split(pkgs, ",")`, primarily for ant-based integration.
`extraClasses`	A character vector of extra classes to map, in addition to those specified in the TypeInfo signature. `extraClasses` for this method are augmented by any classes defined in the file `system.file("rservices","properties.R", package=pkg)`.

Details

createMap operates either on functions with typeInfo applied, or on S4 generic methods.

Some coercion occurs with typeInfo functions. All signatures have an explict return type. IndependentTypeSpecificatio classes are 'expanded' to a complete set of typed signatures.

Function signatures for S4 methods are created from information used in generic method construction, supplemented by an optional default type signature supplied as a named argument S4DefaultTypedSig. Each method defined for a generic results in a Java signature. Arguments used for method dispatch are typed according to the dispatch rules for the function. Return types are determined by the valueClass of the generic (the valueClass of methods seem not to defined, despite what the documentation indicates). Default values for other arguments, and for return values if valueClass is not used, can be created with a TypedSignature passed as a named argument S4DefaultTypedSig. Values implied by S4 method definitions override the defaults.

See typeInfo2Java for additional information on how conversion occurs.

Value

Primarily inovkes for the side-effect of creating a java class hierarchy reflecting the data and methods present in the call to createMap. Returns a list of Java signatures function definitions..

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>

See Also

createMap,missing,character-method

Examples

```
library(RWebServices)

## S4
## Not run:
setClass("A", representation(x="character"))
setGeneric("foo", function(obj) standardGeneric("foo"))
setMethod("foo", "numeric", function(obj) new("A", x=as.character(obj)))
setMethod("foo", "character", function(obj) new("A", x=obj))

createMap(foo, outputDirectory=tempdir(), S4DefaultTypedSig=TypedSignature(returnType="A"))




## End(Not run)
```

`FileReferences` *FileReferences constructor and accessors*

Description

Functions for creating and manipulating FileRefences-class objects.

Usage

```
FileReferences(urls = character(), localNames = basename(urls),
               types = character(length(urls)), ...)
urls(object, ...)
localNames(object, ...)
types(object, ...)
## S4 method for signature 'FileReferences'
length(x)
```

Arguments

`urls`	A character vector of urls of the files to be referenced. The urls are the actual locations of the files.
`localNames`	A (optional) character vector of local names of the files to be references. Local names are the names used to refer to the files internally, e.g., by functions wanting to use file names as a label for data frame rows. The length of the local names argument must match that of the urls; the default is to take the file name portion of the url as the local name.
`types`	A (optional) character vector of length equal to that of the urls, containing the type each file represents. Type might normally come from a controlled vocabulary; the default is a length zero character string, reprsenting an unspecified type.
`object`	An object of class `FileReferences`, form which urls, local names, or types are to be extracted.
`x`	An object of class `FileReferences`, form which the length (number of file references) is to be determined.
`...`	Additional arguments, passed to the class initializaation method or (currently) ignored by the accessors.

Value

`FileReferences`	Object of class `FileReferences`.
`urls, localNames, types`	character vector of urls, local names, or file types.
`length`	integer scalar of the number of references in the object.

Author(s)

Martin Morgan mtmorgan@fhcrc.org

See Also

FileReferences-class

Examples

```
obj <- FileReferences(c("/foo/bar", "/foo/baz"), type=rep("CEL", 2))
localNames(obj)
length(obj)
```

FileReferences-class *Class "FileReferences"*

Description

FileReferences represents a collection of urls at which files can be referenced, the local names by which the files are to be refered to (to allow the file name portion of the url to differ from the name used to refer to the file), and the types (e.g., CEL, xls, etc.) of the files.

Objects from the Class

Objects can be created by calls to the constructor FileReferences. Objects are constructed with equal-length character vectors.

Elements can be accessed with urls, localNames, and types.

Slots

urls: Object of class "character" containing the urls where the files can be lcoated

localNames: Object of class "character" containing the local names by which the files will be refered.

types: Object of class "character" containing information on the file 'type', for instance indicating that the files are CEL files (for gene expression analysis). Usually file types will be from a controlled vocabulary of types defined elsewhere.

Methods

See FileReferences

Author(s)

Martin Morgan mtmorgan@fhcrc.org

See Also

FileReferences for object consturction, accessors urls, localNames, and types

Examples

```
showClass("FileReferences")
FileReferences()
```

---

generateDataMap *Create java beans and mapping functions for a R data type*

Description

The generateDataMap function generates java beans and R–Java data type conversions for a R data type if they are not available in the environment lookup. All java mappings are output as files under CurrentWorkingDir/biocJavaMap, where CurrentWorkingDir is the result of getwd(). This function also updates environments lookup and cvtImport (see arguments below) as a side effect.

Usage

```
generateDataMap(rType, javaToR, deployMode, typeMode, lookup)
```

Arguments

rType	a character vector containing a R data type name. The data type is either a S4 class or a ClassUnion.
javaToR	logical, TRUE if want to convert Java data to R, FALSE if want to convert R data to Java.
deployMode	Character, either "demo" or "jms". How the service will be deployed. "demo" is no longer supported.
typeMode	Character, either "robject" or "javalib". How Java objects should be generated; see generateDataMap and converters
lookup	environment, key is rType, value is of type RJMap. RJMap provides the java type, java package, and java–R conversion function names for the key. By default, lookup only contains mapping information for vector, list, factor, data frame, array, matrix, environment and NULL

Value

returns updated environment lookup.

Author(s)

Nianhua Li

See Also

printLookup

Examples

```
## used internally
```

---

generateDataTest *Create a Java program to test R - Java data type mapping*

Description

The generateDataTest function is used to validate the results of function generateDataMap. Given the name of a R data type, and the lookup environment updated by generateDataMap, function generateDataTest creates a test program in Java. The test program creates a java companion of the R data type, converts it to the R data type, and converts it back to Java. This round-trip can test both the Java bean and the convert functions.

Usage

```
generateDataTest(mainServ, mainPkg, lookup, addonType, verbose)
```

Arguments

mainServ	Name of web service; used to generate class name and properties files.
mainPkg	Package name, used in Java package hierarchy and usually the same as the R package name where the data object is defined.
lookup	environment containing mapping between R and Java types; see generateDataMap
addonType	character. Additional types required for creating the data test,e.g., to test conversion of an R list containing particular S4 classes requires that the particular classes be named as addonType.
verbose	logical. Report progress to console?

Value

returns NULL.

Author(s)

Nianhua Li

See Also

generateDataMap

Examples

```
## see '?generateDataMap' for an example of generating java test program
```

generateFunctionMap *Create java wrapper functions for a list of R functions*

Description

The generateFunctionMap function generates java wrapper functions for a list of R functions. It also generates java beans and R–Java data type conversions for any R data types that are related to the function but unavailable in the environment lookup. This function also updates environments lookup and cvtImport (see arguments below) as a side effect.

Usage

```
generateFunctionMap(rTypeInfo, genTest = TRUE, workDir = getwd(),
                    verbose = FALSE,
                    deployModeName = "jms", typeModeName = "javalib",
                    wsdlStyle = "WRAPPED", wsdlUse = "LITERAL",
                    extraClasses = "", pkgRoot="org.bioconductor", ...)
```

Arguments

rTypeInfo	a list, each component is an instance of RJavaSignature-class and represents the signature of a R function.
genTest	logical, TRUE if want to generate a java program to test the java wrapper functions generated by generateFunctionMap, FALSE otherwise.
workDir	a character string, the path of the directory where this function generates all its outputs.
verbose	logical, TRUE if want to print out debug information, FALSE otherwise.
deployModeName	Character, either "demo" or "jms". How the service will be deployed. "demo" is no longer supported.
typeModeName	Character, either "robject" or "javalib". How Java objects should be generated; see generateDataMap and converters
wsdlStyle	Style of wsdl to create. A value of "WRAPPED" adds "_PortType" to the short name of the main service interface; all other values do not add "_PortType".
wsdlUse	Whether SOAP bindings are document/literal (value "LITERAL" or document/encoded (any other value). Only "LITERAL" is currently supported.
extraClasses	character. Extra classes requied to convert function argument or return values, e.g., an argument of type list might require information about the classes present in the list. Usually this argument is not required, as S4 classes are parsed to their constituent parts
pkgRoot	character. Root package name of the java classes classes being created. It should be a sequence of one or more character strings delimited by dot, and no dot after the last string
...	Additional arguments, to be compatiable with createMap.

Details

The java mappings generated under `workDir` are organized as following:

- biocJavaMap The base directory
- biocJavaMap/`package1` Java mappings for functions and data types defined in R `package1`
- biocJavaMap/`package1`/data Java beans and java <-> R mapping functions (in R) for data types defined in `pacakge1`
- biocJavaMap/`package1`/function Java wrapper functions for functions defined in `package1`
- biocJavaMap/`package2` Java mappings for functions and data types defined in `package2`
- biocJavaMap/`mainService` Main java API, which invokes the java mapping functions for `package1`, `package2`, etc. `mainService` value is provided by user as input.

Value

A list, each component is an instance of `RJavaSignature-class` and represents the signature of a java wrapper function. The return value has a one-to-one mapping relation with input parameter `rTypeInfo`, i.e. the java wrapper function that is represented by the ith component in the return list, is the wrapper for the R function that is represend by the ith component in `rTypeInfo`.

Author(s)

Nianhua Li

Examples

```
##  This function is usually invoked by createMap.
```

`getRSessionInfo` *Report R session information in a web-services friendly format.*

Description

This function has TypeInfo applied, and so is mapped as a web service in the org.bioconductor.packages.rservices name space.

Usage

```
getRSessionInfo(verbose = FALSE)
```

Arguments

`verbose`	A `logical(1)`, with `FALSE` returning a character vector of packages with (for non-base packages) version number appended and `TRUE` returning a character vector capturing the output of `sessionInfo()`.

Value

A character vector containing information on all attached and loaded packages in the current R session.

Author(s)

Martin Morgan mtmorgan@fhcrc.org

Examples

```
getRSessionInfo(FALSE)
```

`printLookup` *Print R-Java data type mapping information*

Description

`printLookup` prints the R-Java data type mapping information.

Usage

```
printLookup(lookup, all = TRUE)
```

Arguments

`lookup`	environment, the environment contains the R-Java data type mapping information. It is generated by `generateDataMap`.
`all`	logical, FALSE to just print the mappings for user-defined data types, TRUE to print the mappings for basic R data types as well.

Author(s)

Nianhua Li

Examples

```
# please try the example under examples/s4Map
```

register-converters *Load R - Java data converters.*

Description

When invoking R from Java, data type conversions between R and Java are handled by SJava. But SJava only supports conversions for limited R data types, the conversions for other R data types are specified by user converters and interpreted by SJava. regAddonCvt and regTestCvt help to load user converters.

Usage

```
regAddonCvt()
regAddonCvt2()
regTestCvt()
```

Details

These functions are usually used by the auto-generated codes from generateFunctionMap. Users do not need to deal with them. regAddonCvt and regAddonCvt2 load converters for R character, logical, integer, numeric, raw, complex, array, matrix, data.frame, list, environment, and factor. The regAddonCvt converters follow the 'robject' model whereas regAddonCvt2 follows the 'javalib' model; see converters for more information. regTestCvt are used for low-level testing.

Author(s)

Nianhua Li

retrieve-methods *Retrieve file references from remote locations*

Description

This method retrieve files referenced by its first argument to locations determined by its second argument. The return value is a FileReferences object summarizing the destination.

Usage

```
retrieve(from, to, ...)
```

Arguments

from	The source from which objects will be retrieved; typically (or exclusively, depending on method) 'remote' locations accessible via, e.g., http.
to	The destination to which files will be copied; typically on the local file system.
...	Additional arguments, passed to the function performing the download.

Methods

from = "FileReferences", to = "character" `from` contains remote urls; files must be retrievable using `getBinaryURL` from the `RCurl` package. `...` arguments are passed to `getBinaryURL`. `to` is a directory name to which files will be copied; the directory is created if it does not exist. File names will be unique.

from = "FileReferences", to = "FileReferences" `from` is as in `retrieve,FileReferences,character-method`. `to` specifies destination urls; `urls(to)` must have the same number of elements as `urls{from}`, must not already exist on the local file system, and must not require creation of new directories. `localNames` and `types` of `to` are replaced by corresponding elements from `from`.

from = "FileReferences", to = "missing" `from` is as in the `retrieve,FileReferences,character-method`. The destination is created using `tempfile()`, and forwarding this as the second argument to `retrieve,FileReferences,character-method`. The effect is to retrieve files to R's temporary directory; these files will be removed when the R session ends.

Examples

```
try({ # fails when no internet available
    from <- FileReferences(c("http://bioconductor.org/biocLite.R",
                             "http://bioconductor.org/index.html"))
    ## to R's temporary directory
    to <- retrieve(from)
    urls(to)
    localNames(to)

    ## tell R where to copy files to
    toDir <- tempfile()
    dir.create(toDir, recursive=TRUE)
    toUrls <- file.path(toDir, localNames(from))
    retrieve(from, FileReferences(toUrls))
    list.files(toDir)
})
```

RJavaPkgFunctions-class

Java signatures derived from R methods

Description

Class `RJavaPkgFunctions` contains elements of Java signatures, including the source R package, the corresponding java class, and type information about the Java function.

Create this class using `createMap` or `generateFunctionMap`.

Slots

`rPackage` Character vector of R source packages

`javaClass` Character vector of corresponding Java classes

`javaTypeInfo` List of Java function signatures.

Author(s)

MT Morgan <mtmorgan@fhcrc.org>

See Also

generateFunctionMap, createMap

Examples

```
library(RWebServices)

oneWayAnova <- function( response, predictor ) {
    if ( is.character( predictor ))
        return( oneWayAnova( response, as.factor( predictor )))
    formula <- as.formula( substitute( response ~ predictor ))
    result <- lm( formula )
    anova( result )
}

typeInfo(oneWayAnova) <-
  SimultaneousTypeSpecification(
    TypedSignature(
      response = "numeric",
      predictor = "factor"),
    TypedSignature(
      response = "numeric",
      predictor = "character"),
    returnType = "anova" )

## Not run:
res <- createMap("oneWayAnova", outputDirectory=tempdir(),
typeMode="robject", verbose=TRUE)

## End(Not run)
```

RJavaSignature-class *String representations of R type signatures from R functions*

Description

Class "RJavaSignature" contains elements of java-style function signature, including return type, function name, and argument names/types

Create this class using typeInfo2Java.

Slots

`returnType` Character string describing the return type of the function. This must be defined.

`funcName`**:** Function name as character string.

`args`**:** Vector of character strings providing argument types. Argument names are stored as `name` attributes.

Author(s)

MT Morgan <mtmorgan@fhcrc.org>

See Also

link{typeInfo2Java}

Examples

```
library(RWebServices)

oneWayAnova <- function( response, predictor ) {
    if ( is.character( predictor ))
        return( oneWayAnova( response, as.factor( predictor )))
    formula <- as.formula( substitute( response ~ predictor ))
    result <- lm( formula )
    anova( result )
}

typeInfo(oneWayAnova) <-
  SimultaneousTypeSpecification(
    TypedSignature(
      response = "numeric",
      predictor = "factor"),
    TypedSignature(
      response = "numeric",
      predictor = "character"),
    returnType = "anova" )

res <- typeInfo2Java(oneWayAnova)

res

summary(res)
```

SinkOutput-class *Results captured by calls to sinkSetup and sinkRetrieve*

Description

Create this class using sinkSetup and sinkRetrieve. A typical use is to wrap function evalutation calls in such a way as to redirect screen output to character variables.

Slots

stdout Vector of type character containg each line of output destined for the 'console', excepting error and warning messages. Each element in the vector is a 'line' of string output, as determined by R.

stderr: As with stdout, but captureing warning and error messages

Author(s)

MT Morgan <mtmorgan@fhcrc.org>

See Also

link{sinkSetup}, sinkRetrieve

Examples

```
library(RWebServices)

sinkSetup()
ls()
try( fails())
f <- function() stop("fails in function f")
try( f())
sinkRetrieve()
```

---

sinkSetup, sinkRetrieve

Capture and retrieve screen output

Description

sinkSetup establishes temporary files to capture output that would normally be directed toward the screen, and redirects both standard and error output to these files. sinkRetrieve retrieves the output, and removes screen redirections so output again appears on the screen.

Usage

```
sinkSetup()
sinkRetrieve()
```

Details

These functions are used to redirect and then retrieve screen output. Redirection is to objects created with file, with no arguments (i.e., a temporary file, unlinked from the file system). Both standard output and error output are redirected.

User-level calls of sink between calls to sinkSetup and sinkRetrieve may confuse screen (especially error) capture, likely resulting in warnings when sinkRetrieve is called.

Value

sinkRetrieve returns an object of SinkOutput-class, the slots of which contain vectors of the lines of text directed toward the screen.

Author(s)

MT Morgan

Examples

```
library(RWebServices)

sinkSetup()
ls()
try( fails())
f <- function() stop("fails in function f")
try( f())
sinkRetrieve()
```

`Test utilities` *Utilities for facilitating data and method tests*

Description

These functions provide utilities that are useful when testing different aspects of R / Java communication.

Usage

```
checkJava2R(javaData, rVariable)
checkPkgVersion(pkgName, expVersion)
reflectObj(x, verbose=FALSE)
```

Arguments

`javaData`	Object translated from Java
`rVariable`	R variable containing an instance that is supposed to be identical to `javaData`. `rVariable` might have been created in previous calls, or loaded from a data file.
`pkgName`	Name of package whose version will be determined.
`expVersion`	Character string with expected package version.
`x`	Object to be returned to Java.
`verbose`	Logical, indicating whether R reports information about the object.

Value

`checkJava2R` returns `TRUE` if `javaData` is identical to `rVariable`, or signals an error otherwise.

`checkPkgVersion` returns true if the installed package matches the expected package, or signals a warning otherwise.

`reflectObj` returns the object.

Author(s)

Nianhua Li

typeInfo2Java *Create Java-like function signatures from R functions*

Description

The typeInfo2Java methods convert functions or TypeInfo structures to a Java-like signature, without coercion to Java types. See createMap to convert functions and coerce arguments.

These functions are primarily for internal use.

Usage

```
typeInfo2Java(x, ...)
```

Arguments

x	function with typeInfo already applied, S4 function, or any TypeInfo class.
...	Additional arguments specific to different methods. These are usually funcName (the name of the function) or args (a list of argument names)

Details

TypeInfo offers greater flexibility in type specification than can easily be implemented in Java. This function fails with DynamicTypeTest, and treats StrictIsTypeTest as InheritsTypeTest (issuing a warning in the process). See NamedTypeTest-class of TypeInfo for more information.

Value

An object of RJavaSignature-class, containing return type, function name, and argument type/name as slots with named character vectors.

Author(s)

MT Morgan

Examples

```
library(RWebServices)

func <- function( response, predictor ) {
    if ( is.character( predictor ))
        return( oneWayAnova( response, as.factor( predictor )))
    formula <- as.formula( substitute( response ~ predictor ))
    result <- lm( formula )
    anova( result )
}

## this indirection makes it easier to apply typeInfo
oneWayAnova <- func
```

```
typeInfo(oneWayAnova) <-
  SimultaneousTypeSpecification(
    TypedSignature(
      response = "numeric",
      predictor = "factor"),
    returnType = "anova" )

typeInfo2Java(oneWayAnova)

oneWayAnova <- func

typeInfo(oneWayAnova) <-
  SimultaneousTypeSpecification(
    TypedSignature(
      response = "numeric",
      predictor = "factor"),
    TypedSignature(
      response = "numeric",
      predictor = "character"),
    returnType = "anova" )

typeInfo2Java(oneWayAnova)


oneWayAnova <- func

typeInfo(oneWayAnova) <-
  IndependentTypeSpecification(
    response = c("numeric"),
    predictor = c( "factor", "character", "numeric" ),
    returnType = "anova"
  )

typeInfo2Java(oneWayAnova)

oneWayAnova <- func

typeInfo(oneWayAnova) <-
  SimultaneousTypeSpecification(
    TypedSignature(
      response = "numeric",
      predictor = "factor",
      returnType = "matrix"),
    TypedSignature(
      response = "numeric",
      predictor = "character"),
    returnType = "anova" )

typeInfo2Java(oneWayAnova)

oneWayAnova <- func
```

```
typeInfo(oneWayAnova) <-
  IndependentTypeSpecification(
    response = c("numeric"),
    predictor = c( "factor", "character", "numeric" ),
    returnType = c("anova")
  )

typeInfo2Java(oneWayAnova)

# Warning about StrictIsTypeTest
oneWayAnova <- func
typeInfo(oneWayAnova) <-
  IndependentTypeSpecification(
    response = StrictIsTypeTest("numeric"),
    predictor = c( "factor", "character", "numeric" ),
    returnType = c("anova")
  )
typeInfo2Java(oneWayAnova)

oneWayAnova <- func
typeInfo(oneWayAnova) <-
  IndependentTypeSpecification(
    response = "numeric",
   predictor = quote( ( length(predictor) == length(response)) && is( predictor, "factor" )),
    returnType = c("anova")
  )
typeInfo2Java(oneWayAnova)
```

`unpackAntScript` *Unpack script for building, testing, and installing packages.*

Description

RWebServices includes a number of scripts and files to be used to develop, test, and deploy services. This function unpacks a 'master' script and partly configured properties files to a convenient directory location.

Usage

```
unpackAntScript(toDir = stop("specify destination directory for unpacking"), overwrite=FALSE)
```

Arguments

`toDir`	Destination (top-level) directory. Warnings are issued if files already exist.
`overwrite`	Overwrite output?

Value

This function is used for its side effect.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>

Examples

```
## Not run: unpackAntScript("/tmp/ants")
## Not run:
## From the command line
echo "library(RWebServices); unpackAntScript('~/tmp/ants')" | R --vanilla

## End(Not run)
```

Index