Jargon

From IRODS

Contents

Jargon (http://www.sdsc.edu/srb/jargon) , A Java client API for the DataGrid

```
       A new grid API, that you already know.
```

The data grid is a new and exciting development in computing, utilizing transparent replication, archiving, caching heterogenous storage, aggregated data movement, shadow objects and a lot of other things most people don't know, or want to know, much about. That's why JARGON has been designed from the ground up to make programming for the grid as straightforward as possible.

Downloads (http://www.sdsc.edu/srb/jargon)

Example programs

- Test.java (http://www.sdsc.edu/srb/jargon/Test.java)
- FactoryTest.java (http://www.sdsc.edu/srb/jargon/FactoryTest.java)
- MetaDataTest.java (http://www.sdsc.edu/srb/jargon/MetaDataTest.java)

CVS (http://www.sdsc.edu/srb/matrix/cvs.cgi/jargon/)

How To: Using Jargon

Synergizing the paradigm with your core skills.

JARGON is a pure java API for developing programs with a data grid interface. The API currently handles file I/O for local and SRB file systems, as well as querying and modify SRB metadata. The API is also easily extensible to other file systems. File handling with JARGON closely matches file handling in Sun's java.io API (http://java.sun.com/j2se/1.4.1/docs/api/java/io/package-summary.html) , hopefully a familiar API to most java programmers.

The top level of the tree has a general class, which contains all the methods independent of the peculiarities of particular file systems. For example, GeneralFile (http://www.sdsc.edu/srb/jargon/doc/edu/sdsc/grid/io/GeneralFile.html) which has available all the methods from the java.io.File (http://java.sun.com/j2se/1.4.1/docs/api/java/io/File.html) class, or GeneralRandomAccessFile (http://www.sdsc.edu/srb/jargon/doc/edu/sdsc/grid/io/GeneralRandomAccessFile.html) with all the methods of the java.io.RandomAccessFile (http://java.sun.com/j2se/1.4.1/docs/api/java/io/RandomAccessFile.html) .

The next level down splits into local and remote classes. The local classes are basically wrappers of the java.io classes, e.g., LocalFile (http://www.sdsc.edu/srb/jargon/doc/edu/sdsc/grid/io/local/LocalFile.html) which wraps java.io.File (http://java.sun.com/j2se/1.4.1/docs/api/java/io/File.html) . The remote classes add functionality common to remote systems, e.g. RemoteRandomAccessFile (http://www.sdsc.edu/srb/jargon/doc/edu/sdsc/grid/io/RemoteRandomAccessFile.html) .

Subclassed off the Remote classes are the remote file system specific classes, e.g. SRBFile (http://www.sdsc.edu/srb/jargon/doc/edu/sdsc/grid/io/srb/SRBFile.html) and SRBRandomAccessFile (http://www.sdsc.edu/srb/jargon/doc/edu/sdsc/grid/io/srb/SRBRandomAccessFile.html) . These classes add the functionality that is only supported by the SRB. For example, SRBFile (http://www.sdsc.edu/srb/jargon/doc/edu/sdsc/grid/io/srb/SRBFile.html) adds a *replicate (http://www.sdsc.edu/srb/jargon/doc/edu/sdsc/grid/io/srb/SRBFile.html#replicate(java.lang.String))* method.

Class structure in JARGON conforms to the following hierarchy.

```
java.lang.Object
   |
   +-edu.sdsc.grid.io.General
      |
      +-edu.sdsc.grid.io.Local
      |
      +-edu.sdsc.grid.io.Remote
         |
         +-edu.sdsc.grid.io.srb.SRB
         |
         +-edu.sdsc.grid.io.irods.iRODS
         |
         +-edu.sdsc.grid.io.ftp.FTP
         |
         +-edu.sdsc.grid.io.http.HTTP
         |
         +-...(future filesystems)
```

The various General classes are abstract classes that get instantiated by one of the subclasses. However factory methods can be used to hide those complexities when not performing tasks special to a particular file system, such as SRBFile.replicate. But I always prefer to just look at example code, so let's move on to that. (More example code is available in the following files: MetaDataTest.java (http://www.sdsc.edu/srb/jargon/MetaDataTest.java) , FactoryTest.java (http://www.sdsc.edu/srb/jargon/FactoryTest.java) and Test.java (http://www.sdsc.edu/srb/jargon/Test.java) .)

Connecting to a File System

This simplest way is if the user tells us what file they want as a uri string, "srb://myName.sdsc@srb.sdsc.edu:5544/srbtest" or "file:/C:/test", This is a good way to instantiate a file object, because the uri contains all the necessary login information and especially as then you (the developer) don't have to pay attention to the specific requirements of the particular file system you are trying to access. (You might want to look at java.net.URI (http://java.sun.com/j2se/1.4.1/docs/api/java/net/URI.html) and the SRB URI protocol.) A few examples (passwords can be included in the URI or [more safely] sent separately):

```
URI uri = new URI( "file:/C:/test" );
GeneralFile file = FileFactory.newFile( uri );
```

```
uri = new URI( "srb://myName.sdsc@srb.sdsc.edu:5544/srbtest" );
GeneralFile file2 = FileFactory.newFile( uri, myPassword );
```

```
GeneralFile file3 = FileFactory.newFile( new URI("ftp://ftp.gnu.org/welcome.msg") );
```

```
GeneralFile file4 = FileFactory.newFile( new URI("irods://anonymous:fakePassword@irods.org:1247/irodstest") );
```

Of course, it is also possible to directly instantiate a specific filesystem. For example, the SRBFileSystem object represents a connection to the SRB. Certain information is used to establish a connection to the SRB, like username and password. The default constructor of the SRBFileSystem tries to find this information in the "home directory"/.srb/ on the local machine. This information may not always be available, in that case an account object must be constructed to hold the necessary information, e.g.

```
SRBAccount account = new SRBAccount(
   host, port, userName, password, homeDirectory, mdasDomainName, defaultStorageResource );
SRBFileSystem srbFileSystem = new SRBFileSystem( account );
```

or:

```
IRODSAccount account = new IRODSAccount (
   host, port, userName, password, homeDirectory, zone, defaultStorageResource );
IRODSFileSystem irodsFileSystem = new IRODSFileSystem( account );
```

This SRB filesystem object can be used to locate files on that system, the constructors for SRBFile objects are the same as java.io.File plus a variable to pass the filesystem object, e.g.,

```
GeneralFile file = FileFactory.newFile( irodsFileSystem, "file2.ext" );
```

```
SRBFile srbFile = new SRBFile( srbFileSystem, "filepath", "file.ext" );
```

```
IRODSFile irodsFile = (IRODSFile) FileFactory.newFile( file, "file.ext" );
```

For third example, the filesystem is contained within the parent file.

Don't forget, the path separator can vary on different filesystems, such as Windows or linux systems:

```
localFile = new LocalFile( System.getProperty("user.home"), "mySubDir" + LocalFile.PATH_SEPARATOR + fileName );
```

GSI authentication

GSI authentication to the SRB and iRODS is also possible with the JARGON API. In place of the standard password in the account object, pass in the filepath of the user's proxy file. For the SRBAccount set the options to GSIAUTH and the locations of the certificate authorities (CA).

```
account = new SRBAccount();
account.setPassword( "/tmp/x509up_u555" );
account.setOptions( SRBAccount.GSI_AUTH );
account.setCertificateAuthority(
   "/etc/grid-security/certificates/b89793e4.0, "+
   "/etc/grid-security/certificates/42864e48.0" );
GeneralFileSystem fileSystem = FileFactory.newFileSystem( account );
```

iRODS GSI can be set using the irodsEnv file or like so:

```
irodsAccount = new IRODSAccount();
irodsAccount.setPassword( "/tmp/x509up_u555" );
//comma seperated list:
irodsAccount.setCertificateAuthority( certificatesAuthorities );
GeneralFileSystem fileSystem = FileFactory.newFileSystem( account );
```

Common File Operations

Once you have a GeneralFile (http://www.sdsc.edu/srb/jargon/doc/edu/sdsc/grid/io/GeneralFile.html) object, all the methods common to the java.io.File class are available. Such as making a directory, checking read access, or listing the files in the directory.

```
file.mkdir();
boolean read = file.canRead();
String[] list = file.list();
```

All of which can be used without knowing if the file is local or remote. Of course, to use one of the SRB specific functions would require an SRB or iRODS declaration or casting the General object to an SRB object.

```
((SRBFile) file).replicate( );
```

Reading and Writing a File

There are a couple ways to manipulate the file's data. You can copy the whole file using the GeneralFile method GeneralFile.copyTo(GeneralFile) (http://www.sdsc.edu/srb/jargon/doc/edu/sdsc/grid/io/GeneralFile#copyTo(edu.sdsc.grid.io.GeneralFile)) . This method will copy this general file object to the general file given as the argument.

```
uri = new URI( args[1] );
file.copyTo( FileFactory.newFile( uri ) );
```

I'm sure you'll be happy to know, that by using the copyTo/From methods certain optimizations have been include which speed up transfers to and from the SRB and iRODS systems using a parallel file transfer protocol. SRB also uses a bulk load method which copies a directory of files much faster then can be done in serial on the SRB.

The GeneralRandomAccessFile (http://www.sdsc.edu/srb/jargon/doc/edu/sdsc/grid/io/GeneralRandomAccessFile.html) and subclasses provide random access support, read, write, seek, modeled after the java.io.RandomAccessFile (http://java.sun.com/j2se/1.4.1/docs/api/java/io/RandomAccessFile.html) .

```
GeneralRandomAccessFile randomAccessFile = FileFactory.newRandomAccessFile( file, "rw" );
randomAccessFile.write( new String( "This is a test file.\n" ) );
```

MetaData

Querying the Metadata

Metadata querying for the SRB was added to the JARGON1.1 release. Here is an example program on metadata querying, MetaDataTest.java.

Queries are roughly like a very simplified SQL, only the equivalents of SELECT and WHERE are used. For a query like, "For all files with size between 0 and 10,000 return the filename."

WHERE:

```
MetaDataSet.newCondition( FileMetaData.SIZE, MetaDataCondition.BETWEEN, 0, 10000 );
```

SELECT:

```
MetaDataSet.newSelection( FileMetaData.FILE_NAME );
```

An array of these conditions and selections forms the query.

A query returns the results as MetaDataRecordList objects.

```
MetaDataRecordList[] recordList = fileSystem.query( conditions, selects );
```

By default a query will only return the first 300 values. The various MetaDataRecordList methods are used to retrieve further results from the query.

Non-standard Metadata

Querying iRODS AVU metadata, metadata defined by each user, uses the above methods but the metadata attributes are not predefined.

WHERE:

```
MetaDataSet.newCondition( "myiRODSAVUMetaData", MetaDataCondition.BETWEEN, 0, 10000 );
```

SELECT:

```
MetaDataSet.newSelection( "myiRODSAVUMetaData" );
```

Querying the SRB definable metadata is simliar to querying other metadata, but the MetaDataTable class is used instead of a scalar value.

```
String[][] definableMetaDataValues = new String[2][2];
definableMetaDataValues[0][0] = "zxcv";
definableMetaDataValues[0][1] = "123";
definableMetaDataValues[1][0] = "asdf";
definableMetaDataValues[1][1] = "999";
int[] operators = new int[definableMetaDataValues.length];
operators[0] = MetaDataCondition.EQUAL;
operators[1] = MetaDataCondition.LESS_THAN;
```

This query would find those items with user defined metadata matching: zxcv = 123 asdf < 999

```
MetaDataTable metaDataTable = new MetaDataTable( operators, definableMetaDataValues );
MetaDataSet.newCondition(
   SRBMetaDataSet.DEFINABLE_METADATA_FOR_FILES, metaDataTable );
```

SRB also has Extensible Metadata, which can be accessed like standard system metadata after setting the extensible schema.

```
SRBMetaDataSet.setExtensibleSchema( fileSystem, "mySchema" );
conditions = new MetaDataCondition[] {
   MetaDataSet.newCondition( "myExtensibleAttribute", MetaDataCondition.EQUAL, "127.0.0.1" ) };
selects = new MetaDataSelect[] {
   MetaDataSet.newSelection( "myExtensibleOtherAttribute" ),
   //System metadata conditions and selects can still work along side
   MetaDataSet.newSelection( SRBMetaDataSet.USER_NAME )
};
//Send the query as before.
MetaDataRecordList[] rl = fileSystem.query( conditions, selects );
```

Modifying the Metadata

Since only iRODS AVUs are modifiable, iRODSFile.modifyMetaData accepts a String array of attributes, values, units. (Units are optional)

```
file.modifyMetaData( new String[]{ newAttribute, newValue, newUnits );
```

The SRB system metadata can be changed by creating a new, or modifying an existing, MetaDataRecordList and then sending it back to the file system. IRODS does not allow modifying system metadata.

```
recordList.addRecord(
   SRBMetaDataSet.getField( FileMetaData.FILE_COMMENTS ), "new comments go here." );
```

```
file.modifyMetaData( recordList );
```

Setting the definable metadata is the same as other metadata.

```
recordList = new SRBMetaDataRecordList( SRBMetaDataSet.getField( SRBMetaDataSet.DEFINABLE_METADATA_FOR_FILES );

file.modifyMetaData( recordList );
```

Be sure to take a look at the example code in, MetaDataTest.java (http://www.sdsc.edu/srb/jargon/MetaDataTest.java) , FactoryTest.java (http://www.sdsc.edu/srb/jargon/FactoryTest.java) and Test.java (http://www.sdsc.edu/srb/jargon/Test.java)
. Even more examples are available in the jargon/src/test directory of the source code.

JavaDocs (http://www.sdsc.edu/srb/jargon/doc)

License Info

The iRODS Clients are distributed by the University of California under the BSD License:

Debugging Help

To get extra debugging information launch your JVM with the jargon.debug variable set from 0-6, e.g.

```
java -Djargon.debug=1
```

Or add the following line of code before instantiating any Jargon java objects.

```
System.setProperty("jargon.debug", "1");
```

The extent of debugging information increases with higher values.

0: No debuging information.

1: A number of methods in GeneralFile can actually result in IOExceptions due to failures communicating to the server. These exceptions are caught and handled within that GeneralFile method. However sometimes it can be useful to see what exceptions did occur. The stack trace of those exceptions are printed to System.err. As well as the name of each low level method call to the server.

2: Adds information about connection and authentication. Helps to insure you are connecting to the expected server as the correct user.

- Higher debugging levels get especially technical...

3: Adds information about the query return value.

4: Adds even more information about the query result and the actual query that was sent to the server.

- Debugging levels beyond here are probably only useful to a developer expanding the Jargon API.

5: Adds the actual bytes sent to the sever by a query and the bytes returned from the server after a query.

6: Adds every byte sent on the socket connection to the server and every byte returned by the server.

See Also:

- Introductory Jargon API PowerPoint presentation (http://www.sdsc.edu/srb/jargon/Intro_Jargon.ppt)

Retrieved from "https://www.irods.org/index.php/Jargon"

- THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL
- UCSD
- This page was last modified 20:03, 18 February 2009.