SimpleSftpFileDownload

Back to the VfsCookbook

Overview

This is a basic example to use VFS to retrieve files from a remote system using the SFTP protocol. Files matching a specified regular expression are retrieved.

Example Configuration

For the purposes of this example the remote system is named "**sftpremote.example.com**". The files to be retrieved are in a directory named **/data/source/fires/smoke** and the files are named **smoke***YearMoDy*_**wkt.txt**. Thus the data file for March 25, 2008 is named "smoke20080325_wkt.txt".

The downloaded files will be received in the local directory **/local/received/fires/smoke**.

Connect to the remote sftpremote.example.com system using the login "smokey" and password "bear".

Key Concepts

Access to a remote system using SFTP uses the SSH secure shell protocol. Although the behavior is similar to FTP, it is *not* FTP run over a secure connection. There are some differences between FTP and SFTP that should be noted. One of these it the lack of the FTP binary/ASCII transfer mode, in SFTP all transfers are binary as if they were executed with an "scp" (secure copy) command.

This example code uses a regular expression to match files on the remote system, so that not all of the files in the source directory are transferred. The `filePatternString` is set to ".*/smoke\\d{8}_wkt
.txt". This has the regular expression components of:

- ".*/" matches any path that precedes the filename. The "." is wildcard character, "*" specifies that 0 or more of these may be present. The "/" matches the directory separator.
- "
  d{8}" matches 8 digits (4 for the year, 2 for the month number, and 2 for the day of the month). The doubled backslash is to escape a single backslash for Java string literals. "\d" indicates any digit, and "{8}" specifies exactly 8 of the preceding character or pattern.
- "
  ." is an escaped "\." which means a literal period instead of a wild card interpretation of "."

It is important to call the close() method of the DefaultFileSystemManager to clean up any temporary files and close all providers. Otherwise the program will appear to hang after downloading files.

Source Code

The code provided below is for a Maven 2 project.

pom.xml Project File

The **`pom.xml`** file defines how the project is built for maven 2:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>gov.noaa.eds.byExample</groupId>
    <artifactId>trySimpleVfsSftp</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>trySimpleVfsSftp</name>
    <url>http://maven.apache.org</url>
    <build>
        <extensions>
            <extension>
                <groupId>org.apache.maven.wagon</groupId>
                <artifactId>wagon-ssh-external</artifactId>
                <version>1.0-beta-2</version>
            </extension>
        </extensions>
        <plugins>
            <plugin>
```

```
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.5</source>
                    <target>1.5</target>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                    <archive>
                        <manifest>
                            <mainClass>gov.noaa.eds.byExample.trySimpleVfsSftp.App</mainClass>
                        </manifest>
                    </archive>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-antrun-plugin</artifactId>
                <configuration>
                    <tasks>
                        <java classname="gov.noaa.eds.byExample.trySimpleVfsSftp.App" classpathref="maven.
runtime.classpath">
                        </java>
                    </tasks>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <dependencies>
        <!-- Supports VFS SFTP -->
        <dependency>
            <groupId>com.jcraft</groupId>
            <artifactId>jsch</artifactId>
            <version>0.1.23</version>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
            <version>1.1</version>
        </dependency>
        <dependency>
            <groupId>commons-vfs</groupId>
            <artifactId>commons-vfs</artifactId>
            <version>1.0</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

Code Listing

Content of **`src/main/java/gov/noaa/eds/byExample/trySimpleVfsSftp/App.java`** (yes, maven does expect the file to be nested down in this directory hierarchy). Be sure to customize the variables near the top of the listing to match your environment if you intend to run this code. The variables to modifiy are `host, user, password, remoteDir, filePatternString` and `localDir`.

```
/*
 * App.java
 */
package gov.noaa.eds.byExample.trySimpleVfsSftp;
```

```
import java.io.File;
import java.util.regex.Pattern;
import org.apache.commons.vfs.AllFileSelector;
import org.apache.commons.vfs.FileObject;
import org.apache.commons.vfs.FileSystemException;
import org.apache.commons.vfs.FileSystemManager;
import org.apache.commons.vfs.FileSystemOptions;
import org.apache.commons.vfs.FileType;
import org.apache.commons.vfs.UserAuthenticator;
import org.apache.commons.vfs.VFS;
import org.apache.commons.vfs.auth.StaticUserAuthenticator;
import org.apache.commons.vfs.impl.DefaultFileSystemConfigBuilder;
import org.apache.commons.vfs.impl.DefaultFileSystemManager;
import org.apache.commons.vfs.impl.StandardFileSystemManager;
import org.apache.commons.vfs.provider.local.LocalFile;


/**
 * Example use of VFS sftp
 *
 */
public class App {

    // Set these variables for your testing environment:
    private String host = "sftpremote.example.com";  // Remote SFTP hostname
    private String user = "smokey";      // Remote system login name
    private String password = "bear";    // Remote system password
    private String remoteDir = "/data/source/fires/smoke";
    // Look for a file path like "smoke20070128_wkt.txt"
    private String filePatternString = ".*/smoke\\d{8}_wkt\\.txt";
    // Local directory to receive file
    private String localDir = "/local/received/fires/smoke";


    private File localDirFile;
    private Pattern filePattern;
    private FileSystemManager fsManager = null;
    private FileSystemOptions opts = null;
    private FileObject sftpFile;

    private FileObject src = null; // used for cleanup in release()

    public static void main(String[] args) {
        System.out.println("SFTP download");
        App app = new App();

        app.initialize();

        app.process();

        app.release();

    } // main( String[] args )


    /**
     * Creates the download directory localDir if it
     * does not exist and makes a connection to the remote SFTP server.
     *
     */
    public void initialize() {
        if (localDirFile == null) {
            localDirFile = new File(localDir);
        }
        if (!this.localDirFile.exists()) {
            localDirFile.mkdirs();
        }

        try {
            this.fsManager = VFS.getManager();
        } catch (FileSystemException ex) {
```

```
        throw new RuntimeException("failed to get fsManager from VFS", ex);
    }

    UserAuthenticator auth = new StaticUserAuthenticator(null, this.user,
            this.password);
    this.opts = new FileSystemOptions();
    try {
        DefaultFileSystemConfigBuilder.getInstance().setUserAuthenticator(opts,
                auth);
    } catch (FileSystemException ex) {
        throw new RuntimeException("setUserAuthenticator failed", ex);
    }

    this.filePattern = Pattern.compile(filePatternString);
} // initialize()


/**
 * Retrieves files that match the specified FileSpec from the SFTP server
 * and stores them in the local directory.
 */
public void process() {

    String startPath = "sftp://" + this.host + this.remoteDir;
    FileObject[] children;

    // Set starting path on remote SFTP server.
    try {
        this.sftpFile = this.fsManager.resolveFile(startPath, opts);

        System.out.println("SFTP connection successfully established to " +
                startPath);
    } catch (FileSystemException ex) {
        throw new RuntimeException("SFTP error parsing path " +
                this.remoteDir,
                ex);
    }


    // Get a directory listing
    try {
        children = this.sftpFile.getChildren();
    } catch (FileSystemException ex) {
        throw new RuntimeException("Error collecting directory listing of " +
                startPath, ex);
    }

    search:
    for (FileObject f : children) {
        try {
            String relativePath =
                    File.separatorChar + f.getName().getBaseName();

            if (f.getType() == FileType.FILE) {
                System.out.println("Examining remote file " + f.getName());

                if (!this.filePattern.matcher(f.getName().getPath()).matches()) {
                    System.out.println("  Filename does not match, skipping file ." +
                            relativePath);
                    continue search;
                }

                String localUrl = "file://" + this.localDir + relativePath;
                String standardPath = this.localDir + relativePath;
                System.out.println("  Standard local path is " + standardPath);
                LocalFile localFile =
                        (LocalFile) this.fsManager.resolveFile(localUrl);
                System.out.println("    Resolved local file name: " +
                        localFile.getName());

                if (!localFile.getParent().exists()) {
```

```
                    localFile.getParent().createFolder();
                }

                System.out.println("  ### Retrieving file ###");
                localFile.copyFrom(f,
                        new AllFileSelector());
            } else {
                System.out.println("Ignoring non-file " + f.getName());
            }
        } catch (FileSystemException ex) {
            throw new RuntimeException("Error getting file type for " +
                    f.getName(), ex);
        }
    } // for (FileObject f : children)

    // Set src for cleanup in release()
    src = children[0];
} // process(Object obj)


/**
 * Release system resources, close connection to the filesystem.
 */
public void release() {
    FileSystem fs = null;

    fs = this.src.getFileSystem(); // This works even if the src is closed.
    this.fsManager.closeFileSystem(fs);
} // release()
} // class App
```

Detlev Moerk brought to my attention that there is an alternative method to casting fsManager to DefaultFileSystemManager and then calling close(). The release method could instead use `this.fsManager.closeFileSystem(fs);`, you would have to declare **fs** as a FileSystem object in the class and set it from one of the FileObjects in the process routine:

```
FileObject src = null;

<-- In processing(), do your things, including src = children[0] -->

public void release() {
    FileSystem fs = null;

    this.src.close(); // Seems to still work even if this line is omitted
    fs = this.src.getFileSystem(); // This works even after the src is closed.
    this.fsManager.closeFileSystem(fs);
}
```

I have incorporated Detlev's suggestion into the example above, because the original release method used "`((DefaultFileSystemManager) this.fsManager).close();`" which worked once, but if you reinitialize and run the process again you will get an error containing

```
org.apache.commons.vfs.FileSystemException: Unknown scheme "sftp" in URI "sftp://sftpremote.example.com/data
/source/fires/smoke/"
```

Detlev's method doesn't have this problem.

Compiling

Compile the source code with

```
mvn assembly:assembly
```

This will create an executable jar file in the standard `target` directory.

Running

Use a command like this to run the example

```
java -jar target/trySimpleVfsSftp-1.0-SNAPSHOT-jar-with-dependencies.jar
```

Sample Output

If the remote system has the following files in the /data/source/fires/smoke directory:

README.txt
smoke20070328_wkt.txt
smoke20070426_wkt.txt
smoke20070430.txt

The middle two with "_wkt" in their names should be retrieved.

```
SFTP download
Mar 25, 2008 1:00:44 PM org.apache.commons.vfs.VfsLog info
INFO: Using "/tmp/vfs_cache" as temporary files store.
SFTP connection successfully established to sftp://sftpremote.example.com/data/source/fires/smoke
Examining remote file sftp://sftpremote.example.com/data/source/fires/smoke/README.txt
  Filename does not match, skipping file ./README.txt
Examining remote file sftp://sftpremote.example.com/data/source/fires/smoke/smoke20070328_wkt.txt
  Standard local path is /local/received/fires/smoke/smoke20070328_wkt.txt
    Resolved local file name: file:///local/received/fires/smoke/smoke20070328_wkt.txt
  ### Retrieving file ###
Examining remote file sftp://sftpremote.example.com/data/source/fires/smoke/smoke20070426_wkt.txt
  Standard local path is /local/received/fires/smoke/smoke20070426_wkt.txt
    Resolved local file name: file:///local/received/fires/smoke/smoke20070426_wkt.txt
  ### Retrieving file ###
Examining remote file sftp://sftpremote.example.com/data/source/fires/smoke/smoke20070430.txt
  Filename does not match, skipping file ./smoke20070430.txt
```

There should now be files matching the `filePatternString` in the local machine directory "/local/received/fires/smoke".