Quick Demo

A quick demo of Skwish.

Table of contents

1. Overview

This short article introduces the library's capabilities through a simple demo program. The demo program allows the user to import documents from the file system into a "container" and/or serve documents from the container over HTTP.

2. The FileStore program

Source code for the demo program (`com.faunos.skwish.eg.FileStore`) is included with the distribution.

Note:

See `src/skwish/src/test/java/com/faunos/skwish/eg/FileStore.java` in your unpacked skwish download.

FileStore maintains a simple container for "files" consisting of 2 `SegmentStore`s: one, called *meta*, holds meta information about the files; the other, called *cache*, contains the raw contents of the files. So each file is represented using 2 entries: one, a cache entry containing its byte contents, and two, a meta entry that records a file path and a numeric reference to the associated cache entry. In a real application, the meta entry might contain additional information, such as the content type, content encoding, etc.

Run FileStore first with the help option '-h'. This gives a brief description of the program and the arguments it accepts. The program's console output is displayed below. (Classpath settings are omitted: set it to include the skwish libs.)

```
>$ java com.faunos.skwish.eg.FileStore -h

Usage:
 java com.faunos.skwish.eg.FileStore [-chw] <storage dir> [source
file/dir]*

 Arguments:
   <storage dir>      The storage directory. Required.
   [source file/dir]* Zero or more files or directories for import. Each
                      file or directory is imported in a separate thread.

 Options:
   -c  Creates a new storage dir
   -h  Prints this message
   -w  Exposes storage contents via HTTP

 Description:
   A Skwish demo program that imports files into a "storage" container.
```

```
   The container consists of 2 segment stores:
     1. cache. Contains file contents.
     2. meta.  Contains meta information about the files.
   Both segment stores are accessible through HTTP (option -w).

>$
```

Now let's give the program something to do. Identify 2 directories for import. In our example, we'll target the root directory of the javadoc-generated files for the skwish project, and also the root directory for the project's source files, but any 2 sufficiently populated directory structures will do for demo purposes. Here's what we type on the command line:

```
  >$ java -cp <classpath-settings> com.faunos.skwish.eg.FileStore -c
MyFileStore ../doc ../src/ -w
```

This creates (writes) a new container directory `MyFileStore`, starts a read-only web server on the container, and imports all files under the `doc` and `src` subdirectories. Each argument is imported under a separate thread and is scoped within its own transactions.

Let's see what just happened. Point your browser to the following URL:

```
http://localhost:8880/meta?id=0
```

This returns the entry with id 0 in the meta store. It contains meta information about the first file imported in the first committed transaction. The plain text response from the server looks something like the following..

```
uri=/src/.svn/all-wcprops
c.id=0
c.tid=1
```

This tells us the file path ended with `src/.svn/all-wcprops`. It also provides us a way to reference the file contents in the *cache* store. Let's do that:

```
http://localhost:8880/cache?id=0&tid=1
```

In this case, the plain text response is:

```
K 25
svn:wc:ra_dav:version-url
V 43
/svnroot/skwish/!svn/ver/6/skwish/trunk/src
END
```

You can "browse" other files in the "container" the same way. You look up the meta entry with id *x* (`http://localhost:8880/meta?id=x`) and then retrieve its contents using `http://localhost:8880/cache?id=c.id&tid=c.tid` where *c.id* and *c.tid* stand for the values of `c.id` and `c.tid` noted in the meta entry.

This pretty much sums up what Skwish can do on its own. You can browse the "container", but not search it. For that you would need something like Lucene (see next article). Skwish is meant to complement such tools, not replace them, and considers the task of indexing

orthogonal to task of managing blobs of data.

(Skwish's experimental HTTP server is not really germane to the project, but it does accept some additonal query string parameters that you can read about here.)

3. Source listing

The contents of `FileStore.java` (included in the distribution) is listed below.

```
package com.faunos.skwish.eg;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.PrintStream;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
import java.util.List;

import com.faunos.skwish.SegmentStore;
import com.faunos.skwish.TxnSegment;
import com.faunos.skwish.ext.http.SkwishHttpMountPoint;
import com.faunos.skwish.ext.http.SkwishHttpServer;
import com.faunos.util.io.BufferUtil;
import com.faunos.util.io.file.DirectoryOrdering;
import com.faunos.util.io.file.FileSystemTraverser;
import com.faunos.util.main.Arguments;
import com.faunos.util.test.AbbreviatedFilepath;
import com.faunos.util.tree.TraverseListener;

/**
 *
 *
 * @author Babak Farhang
 */
public class FileStore {


    public final static String HELP_OPT = "h";

    public final static String CREATE_OPT = "c";

    public final static String WEB_OPT = "w";



    public static void main(String[] args) throws Exception {
        try {
            mainImpl(args);
        } catch (Exception e) {
            printUsage(System.err);
```

```
            printCurrentDir(System.err);
            throw e;
        }
    }



    private static void mainImpl(String[] args) throws Exception {
        Arguments arguments = new Arguments();
        arguments.getOptions().getIdentifiers().add(HELP_OPT);
        arguments.getOptions().getIdentifiers().add(CREATE_OPT);
        arguments.getOptions().getIdentifiers().add(WEB_OPT);
        arguments.parse(args);
        if (arguments.getOptionList().contains(HELP_OPT)) {
            printUsage(System.out);
            return;
        }
        List<String> filepaths = arguments.getArgumentList();
        if (filepaths.size() < 1)
            throw new IllegalArgumentException(filepaths.toString());

        File storageDir = new File(filepaths.get(0));
        final boolean create =
arguments.getOptionList().contains(CREATE_OPT);
        if (create)
            storageDir.mkdirs();
        if (!storageDir.isDirectory())
            throw new IllegalArgumentException(
                    storageDir + " is not a directory");

        final SegmentStore cache, meta;
        {
            File cacheDir = new File(storageDir, "cache");
            File metaDir  = new File(storageDir, "meta");
            if (create) {
                cache = SegmentStore.writeNewInstance(cacheDir.getPath());
                meta = SegmentStore.writeNewInstance(metaDir.getPath());
            } else {
                cache = SegmentStore.loadInstance(cacheDir.getPath());
                meta = SegmentStore.loadInstance(metaDir.getPath());
            }
        }
        SkwishHttpServer http;
        if (arguments.getOptionList().contains(WEB_OPT)) {
            http = SkwishHttpServer.newInstance(8880);
            SkwishHttpMountPoint mountPoint =
http.newSkwishMountPoint("/");
            mountPoint.mapRelativeUriToSkwish("cache", cache);
            mountPoint.mapRelativeUriToSkwish("meta", meta);
            http.start();
        } else
            http = null;
```

```
        for (int i = 1; i < filepaths.size(); ++i) {
            FileImporter importer
                = new FileImporter(cache, meta, new
File(filepaths.get(i)));
            new Thread(importer).start();
        }

        try { Thread.sleep(120000); } catch (InterruptedException ix) { }

//        if (http != null)
//            http.stop();
//
//        cache.close();
//        meta.close();
    }

    private static void printUsage(PrintStream ps) {
        ps.println();
        ps.println("Usage:");
        ps.println(" java " + FileStore.class.getName() +
                           " [-chw] <storage dir> [source file/dir]*");
        ps.println();
        ps.println(" Arguments:");
        ps.println("   <storage dir>      The storage directory.
Required.");
        ps.println("   [source file/dir]* Zero or more files or directories
" +
                                         "for import. Each");
        ps.println("                       file or directory is imported in
a " +
                                         "separate thread.");
        ps.println();
        ps.println(" Options:");
        ps.println("   -c  Creates a new storage dir");
        ps.println("   -h  Prints this message");
        ps.println("   -w  Exposes storage contents via HTTP");
        ps.println();
        ps.println(" Description:");
        ps.println("   A Skwish demo program that imports files into a " +
                   "\"storage\" container.");
        ps.println("   The container consists of 2 segment stores:");
        ps.println("     1. cache. Contains file contents.");
        ps.println("     2. meta.  Contains meta information about the
files.");
        ps.println("   Both segment stores are accessible through HTTP " +
                  "(option -w).");
        ps.println();
    }

    private static void printCurrentDir(PrintStream ps) throws IOException
{
        File currentDir = new File(".").getCanonicalFile();
        ps.println("current dir: " + new AbbreviatedFilepath(currentDir));
        ps.println("    " + currentDir);
```

```
        ps.println();
    }


    static class FileImporter implements Runnable {

        private final TxnSegment cacheTxn;
        private final long cacheTxnId;
        private final TxnSegment metaTxn;
        private final File root;
        private final String parentFilepath;
        private final boolean escapeBackslash;

        private int count;

        FileImporter(SegmentStore cache, SegmentStore meta, File src)
                throws IOException
        {
            if (!src.exists())
                throw new IllegalArgumentException(
                        "no such file or directory: " + src);
            cacheTxn = cache.newTransaction();
            cacheTxnId = cacheTxn.getTxnId();
            metaTxn = meta.newTransaction();
            this.root = src.getCanonicalFile();
            this.parentFilepath = root.getParent();
            this.escapeBackslash = File.separatorChar == '\\';
        }

        public void run() {
            FileSystemTraverser traverser = new FileSystemTraverser(root);
            traverser.setSiblingOrder(DirectoryOrdering.FILE_FIRST);
            traverser.setListener(
                    new TraverseListener<File>() {

                        public void preorder(File file) {
                            try {
                                processFile(file);
                            } catch (IOException iox) {
                                throw new RuntimeException(iox);
                            }
                        }

                        public void postorder(File file) {   }

                    });

            System.out.println(name() + "starting import..");
            try {
                System.out.println(name() + "c.tid=" +
cacheTxn.getTxnId());
                long time = System.currentTimeMillis();
                traverser.run();
```

```
                cacheTxn.commit();
                metaTxn.commit();

                time = System.currentTimeMillis() - time;
                System.out.println(
                        name() + "imported " +
                        (count == 1 ? "1 file" : count + " files") +
                        " in " + time + " msec [" +
                        (time + 500) / 1000  + " sec]");
            } catch (IOException iox) {
                throw new RuntimeException(iox);
            }
        }

        private String name() {
            return "[" + Thread.currentThread().getName() + "]: ";
        }

        private void processFile(File file) throws IOException {
            // set the URI we will use for the file in our "file store"
            // the URI will start with the simple name of the root file
            String uri =
file.getPath().substring(this.parentFilepath.length());
            if (escapeBackslash)
                uri = uri.replace('\\', '/');
            if (file.isDirectory()) {
                System.out.println(name() + "skipping directory entry " +
uri);
                return;
            }
            System.out.println(name() + uri);
            // write the file contents into the cache store
            // and note the resulting entry's transaction-scoped id
            // (which together with this.cacheTxnId allows us to resolve
the
            // final, absolute entry id, after the cacheTxn has been
committed)
            FileChannel input = new FileInputStream(file).getChannel();
            long cid = cacheTxn.insertEntry(input);
            input.close();

            // write a "meta entry" for the file in the meta store
            StringBuilder metaString = new StringBuilder(uri.length() +
32);
            metaString.append("uri=").append(uri)
                      .append("\nc.id=").append(cid)
                      .append("\nc.tid=").append(cacheTxnId);
            ByteBuffer metaEntry =
BufferUtil.INSTANCE.asciiBuffer(metaString);
            metaTxn.insertEntry(metaEntry);
            ++count;
        }
```

```
    }
}
```

4. What next?

This is a really small lib. So if you worked through this example, you already have a good picture of what the library has to offer. Hopefully, any additional information you need can be obtained by browsing the following javadoc files:

- `SegmentStore`,
- `Segment`, and
- `TxnSegment`

But perhaps it would be better to demo Skwish by plugging it into other tools. The next article is exactly about that.