Hadoop HDFS Java API

Hadoop's org.apache.hadoop.fs.FileSystem is generic class to access and manage HDFS files/directories located in distributed environment. File's content stored inside datanode with multiple equal large sizes of blocks (e.g. 64 MB), and namenode keep the information of those blocks and Meta information. FileSystem read and stream by accessing blocks in sequence order. FileSystem first get blocks information from NameNode then open, read and close one by one. It opens first blocks once it complete then close and open next block. HDFS replicate the block to give higher reliability and scalability and if client is one of the datanode then it tries to access block locally if fail then move to other cluster datanode. FileSystem uses FSDataOutputStream and FSDataInputStream to write and read the contents in stream. Hadoop has provided various implementation of FileSystem as described below:

- ***DistributedFileSystem:*** *To access HDFS File in distributed environment*
- ***LocalFileSystem:*** *To access HDFS file in Local system*
- ***FTPFileSystem:*** *To access HDFS file FTP client*
- ***WebHdfsFileSystem:*** *To access HDFS file over the web*

URI and Path:
Hadoop's URI locate file location in HDFS. It uses hdfs://host: port/location to access file through FileSystem.
Below code show how to create URI

```
e.g. hdfs://localhost:9000/TestFile.txt
URI uri=URI.create ("hdfs://host: port/path");
```

Path consist URI and resolve the OS dependency in URI e.g. Windows uses \path whereas linux uses //. It also uses to resolve parent child dependency.
It could be created as below

```
Path path=new Path (uri); //It constitute URI
```

Configuration
Configuration class passes the Hadoop configuration information to FileSystem. It loads the core-site and core-default.xml through class loader and keeps Hadoop configuration information such as fs.defaultFS, fs.default.name etc. You can create the Configuration class as below

```
Configuration conf = new Configuration ();
```

You can also set the configuration parameter explicitly as below

```
conf.set("fs.default.name", “hdfs://localhost:9000”);
```

FileSystem

Below code describe how to create Hadoop’s FileSystem

```
public static FileSystem get(Configuration conf)
public static FileSystem get(URI uri, Configuration conf)
public static FileSystem get(URI uri, Configuration conf, String user)
```

FileSystem uses NameNode to locate the DataNode and then directly access DataNodes block in sequence order to read the file. FileSystem uses Java IO FileSystem interface mainly DataInputStream and DataOutputStream for IO operation.
If you are looking to get local filesystem we can directly use getLocal method as mentioned below

```
public static LocalFileSystem getLocal(Configuration conf)
```

FSDataInputStream
FSDataInputStream wraps the DataInputStream and implements Seekable, PositionedReadable interfaces which provide method like getPos(), seek() method to provide Random Access on HDFS file.
FileSystem have open() method which return FSDataInputStream as below:

```
URI uri = URI.create (“hdfs://host: port/file path”);
Configuration conf = new Configuration ();
FileSystem file = FileSystem.get (uri, conf);
FSDataInputStream in = file.open(new Path(uri));
```

Above method get FSDataInputStream with default buffer size 4096 byte i.e. 4KB. We can also define the buffer size while creating Input Stream as below code.
public abstract FSDataInputStream open(Path path, int sizeBuffer)
FSDaraInputStream implements seek (long pos) and getPos () method of Seekable interface.

```
public interface Seekable {
  void seek(long pos) throws IOException;
  long getPos() throws IOException;
 boolean seekToNewSource(long targetPos) throws IOException;
}
```

seek() method seek the file to the given offset from the start of the file so that read () will stream from that location whereas getPos() method will return the current position on the InputStream.
Below sample code uses seek (), getPos () and read() method

```
FileSystem file = FileSystem.get (uri, conf);
FSDataInputStream in = file.open(new Path(uri));
byte[] btbuffer = new byte[5];
in.seek(5); // sent to 5th position
Assert.assertEquals(5, in.getPos());
in.read(btbuffer, 0, 5);//read 5 byte in byte array from offset 0
System.out.println(new String(btbuffer));//print 5 character from 5th position
in.read(10,btbuffer, 0, 5);// print 5 character staring from 10th position
```

FSDataInputStream also implements PositionedReadable, which provide read, & readFully method to read part of file content from seek position as mentioned below read(long position, byte[] buffer, int offset, int length)

FSDataOutputStream
Filesystem's create () method return FSDataOutputStream, which use to create new HDFS file or write the content at the EOF. It doesn't provide seek because of HDFS limitation to write to content at the EOF only. It wrap Java IO's DataOutputStream and add method such as getPos() to get the position of the file and write() to write the content at the last position.

Below method signature provide FSDataOutputStream:
Create method on FileSystem create file e.g.

```
public FSDataOutputStream create(Path f) create empty file.
public FSDataOutputStream append(Path f) will append existing file
```

Create method also pass Progressable interface to track the status during file creation.

```
public FSDataOutputStream create(Path f, Progressable progress)
```

FileStatus
As describe below code getStatus() method of FileSystem provide HDFS file's meta information of HDFS file

```
URI uri=URI.create(strURI);
        FileSystem fileSystem=FileSystem.get(uri,conf);
        FileStatus fileStatus=fileSystem.getFileStatus(new Path(uri));
        System.out.println("AccessTime:"+fileStatus.getAccessTime());
```

```
System.out.println("AccessTime:"+fileStatus.getLen());
System.out.println("AccessTime:"+fileStatus.getModificationTime());
System.out.println("AccessTime:"+fileStatus.getPath());
```

If your uri is directory not file then listStatus() will give you array of FileStatus[] as below

```
public FileStatus[] listStatus(Path f)
```

Directories

FileSystem provide method public boolean mkdirs (Path f) to create directory and its entire necessary child directory if not available. It returns true if directory created successfully. This is not mandatory as whenever you create file it will try to create necessary sub directories.

Delete file

Delete method on FileSystem remove the file/directory permanently
public boolean delete(Path f, boolean recursive) throws IOException
If recursive is true then it will delete a non-empty directory