# FsShell

提供到FileSystem的命令行访问

属性：

1. **protected** FileSystem fs;
2. **private** Trash trash;

方法：

1. **private** **void** copyFromStdin(Path dst, FileSystem dstFs)

Copies from stdin to the indicated file.（即源地址参数srcs为"-"）

* 1. 如果dst是目录，则抛出IOException，说明stdin的操作目标只能是文件
  2. 如果指定的文件已经存在，也抛出IOException
  3. 创建文件并打开流：FSDataOutputStream out = dstFs.create(dst);
  4. 把stdin写到流中IOUtils.*copyBytes*(System.*in*, out, getConf(), **false**);

1. **void** copyFromLocal(Path[] srcs, String dstf)

Add local files to the indicated FileSystem name. src is kept.

负责处理-put和-copyFromLocal两个命令

* 1. 如果传入的源地址参数srcs为"-"，则表示从标准输入读取数据，并上传到DFS，调用copyFromStdin(dstPath, dstFs);
  2. 否则，就是把文件上传，调用dstFs.copyFromLocalFile(**false**, **false**, srcs, dstPath);
  3. FileSystem的copyFromLocalFile方法实际上是调用FileUtil的copy方法（最终还是要使用DistributedFileSystem和DFSClient处理数据）

1. **void** moveFromLocal(Path[] srcs, String dstf)

Add local files to the indicated FileSystem name. src is removed.

* 1. 调用dstFs.moveFromLocalFile(srcs, dstPath);
  2. 实际就是调用copyFromLocal，只是添加了一个参数，把源文件删除

1. **void** copyToLocal(String[]argv, **int** pos)
   1. 初始化CommandFormat cf = **new** CommandFormat("copyToLocal", 2,2,"crc","ignoreCrc");（"crc","ignoreCrc"为可选参数，默认为false）
   2. 解析所有参数：List<String> parameters = cf.parse(argv, pos);（这一步会结合上个步骤传入的选项，解析参数，因为传入的是2，2,即上下限都是2，所以必须有两个参数，即一个源地址，一个目的地址），如果解析出错，即参数的数目不对，则会抛出IllegalArgumentException异常
   3. 从解析出的参数中获取srcstr = parameters.get(0);和dststr = parameters.get(1);
   4. 读取选项参数：
      1. **boolean** copyCrc = cf.getOpt("crc");
      2. **final** **boolean** verifyChecksum = !cf.getOpt("ignoreCrc");
   5. 如果dststr.equals("-")，即，要把获取的结果打印到stdout上：
      1. 如果copyCrc为true，则打印出错误，因为此处没办法校验（注意此步骤只是忽略这个参数，并没有停止执行）
      2. 调用cat(srcstr, verifyChecksum);输出到stdout
   6. 否则，即，把远程文件复制到本地文件：
      1. 初始化本地文件File dst = **new** File(dststr);
      2. 简历远程路径Path srcpath = **new** Path(srcstr);
      3. 获取远程的文件系统FileSystem srcFS = getSrcFileSystem(srcpath, verifyChecksum);
      4. 如果copyCrc为true，但是srcFS不是ChecksumFileSystem类型，则打印出错误，并设置copyCrc为false
      5. 查询远端文件状态：FileStatus[] srcs = srcFS.globStatus(srcpath);（此处使用通配符查询，所以可以看出，这里的复制操作可以同时复制多个文件）
      6. 如果srcs不止一个文件，但是本地文件dst不是目录，则抛出IOException，因为当复制多个文件的时候，目的地址必须是目录
      7. 遍历srcs，对每个FileStatus status，调用copyToLocal(srcFS, p, f, copyCrc);复制文件到本地
2. **private** **void** copyToLocal(**final** FileSystem srcFS, **final** Path src, **final** File dst, **final** **boolean** copyCrc)

Copy a source file from a given file system to local destination.

* 1. 如果远端文件srcFS不是目录：
     1. 如果本地文件已经存在，则抛出IOException
     2. 创建临时文件File tmp = FileUtil.*createLocalTempFile*(dst.getAbsoluteFile(),*COPYTOLOCAL\_PREFIX*, **true**);
     3. 把远程文件写到临时文件中FileUtil.*copy*(srcFS, src, tmp, **false**, srcFS.getConf())
     4. 把临时文件重命名为目标文件
     5. 如果copyCrc为true，即需要赋值crc文件（默认的配置中，hdfs是没有crc文件的）：
        1. 如果srcFS不是ChecksumFileSystem类型，则抛出IOException
        2. 强转ChecksumFileSystem csfs = (ChecksumFileSystem) srcFS;
        3. 递归调用本方法，把crc文件赋值过来

1. **void** copyMergeToLocal(String srcf, Path dst, **boolean** endline)

负责处理-getmerge，Get all the files in the directories that match the source file pattern and merge and sort them to only one file on local fs srcf is kept.

* 1. 实际是调用FileUtil.*copyMerge*合并多个文件，根据传入的endline参数判断是都在每个文件末尾追加分隔符

1. **void** cat(**final** String src, **boolean** verifyChecksum)

获取所有指定的文件，并打印他们，src可以是通配符：

* 1. 使用DelayedExceptionThrowing处理，直接打印出所有文件，对所有匹配的文件调用printToStdout(srcFs.open(p));
  2. printToStdout其实就是调用IOUtils.*copyBytes*(in, System.*out*, getConf(), **false**);

1. **void** text(String srcf)

与cat类似，调用forMagic输出文件

1. **private** **void** setReplication(String[] cmd, **int** pos)

负责处理-setrep：

1. 初始化CommandFormat c = **new** CommandFormat("setrep", 2, 2, "R", "w");其中R代表recursive，w代表waitList
2. 解析参数List<String> parameters = c.parse(cmd, pos);
3. 读取解析后的参数：rep = Short.*parseShort*(parameters.get(0));
4. dst = parameters.get(1);
5. 调用setReplication(rep, dst, c.getOpt("R"), waitList);
6. 如果waitList不为null，则等待waitForReplication(waitList, rep);
7. **void** waitForReplication(List<Path> waitList, **int** rep)

Wait for all files in waitList to have replication number equal to rep.

遍历waitList，对每个Path f：

* 1. 获取文件状态FileStatus status = fs.getFileStatus(f);
  2. 执行死循环，**for**(**boolean** done = **false**; !done; )：
     1. 获取所有block副本状态BlockLocation[] locations = fs.getFileBlockLocations(status, 0, len);
     2. 对每个block执行循环，从第一个开始，等待locations[i].getHosts().length == rep;，直到所有的locations（即block位置）的数目都等于期望的rep才会停止循环（设置done为true）
  3. 而只有waitList中的每个文件都完成自己的循环后，才会退出最外层的循环
  4. 代码里，打印WARNING的位置有问题

1. **private** **void** setReplication(**short** newRep, FileSystem srcFs, Path src, **boolean** recursive, List<Path> waitingList)
   1. 如果src不是目录：
      1. 调用setFileReplication(src, srcFs, newRep, waitingList);
      2. Return
   2. 如果是目录，遍历目录下所有文件：
      1. 如果子文件不是目录，则调用setFileReplication(items[i].getPath(), srcFs, newRep, waitingList);
      2. 如果子文件也是目录：
         1. 如果recursive为false，则什么都不执行
         2. 否则，递归调用setReplication(newRep, srcFs, items[i].getPath(), recursive, waitingList);
2. **void** du(String src)

处理-du类型的操作:

* 1. 对于src指定的文件，直接显示文件长度
  2. 如果src指定的是目录，则显示目录下的所有文件的长度，子文件夹会调用srcFs.getContentSummary(items[i].getPath())获取整个的长度（不递归）

1. **void** dus(String src)

类似du操作，对匹配src的所有文件调用srcFs.getContentSummary(status[i].getPath()).getLength();获取summary信息

1. **void** mkdir(String src)

处理-mkdir类型：Create the given dir

* 1. 调用srcFs.mkdirs(f)

1. **void** touchz(String src)

处理-touchz类型：(Re)create zero-length file at the specified path. This will be replaced by a more UNIX-like touch when files may be modified.

新建FSDataOutputStream out = srcFs.create(f);，如果文件存在且长度不为0则抛出IOException，这和linux的touch不同，不能修改访问时间

1. **int** test(String argv[], **int** i)

处理-test类型：Check file types.

检查指定的文件，e表示存在，z表示长度是否为0，d表示目录，貌似没有任何效果？

1. **void** stat(**char**[] fmt, String src)

处理-stat类型：Print statistics about path in specified format.，format可以是以下组合的组合，默认是%y：（注意不能指定多个文件，但是可以使用通配符）

\* %b: Size of file in blocks

\* %n: Filename

\* %o: Block size

\* %r: replication

\* %y: UTC date as &quot;yyyy-MM-dd HH:mm:ss&quot;

\* %Y: Milliseconds since January 1, 1970 UTC

* 1. 获取所有文件FileStatus glob[] = srcFs.globStatus(srcPath);
  2. 遍历**for** (FileStatus f : glob)：
     1. 遍历格式控制参数：**for** (**int** i = 0; i < fmt.length; ++i)
     2. 对于上面提到的控制符，使用相应的文件属性填充
     3. 对于上面没有提到的控制符，或者其他字符串，则直接保持原样打印
  3. 注意，%相当于转义符，%%则会输出一个%

1. **void** rename(String srcf, String dstf)
   1. 确保src和dst在同一个文件系统
   2. 如果srcs指向多个文件，则要求dstf必须是目录
   3. 对所有源文件调用srcFs.rename
2. **private** **int** rename(String argv[], Configuration conf)

处理-mv类型的操作，Move/rename file(s) to a destination file. Multiple source files can be specified. The destination is the last element of the argvp[] array. If multiple source files are specified, then the destination must be a directory. Otherwise, IOException is thrown.

1. **void** copy(String srcf, String dstf, Configuration conf)
   1. 调用FileUtil.*copy*(srcFs, srcs[i], dstFs, dstPath, **false**, conf);
   2. 可以看出这里并没有要求一定要在一个文件系统上,和mv的不同。默认情况下，使用配置中的文件系统（即hdfs），如果想复制到本地，可以使用<file:///>这样的uri
2. **private** **int** copy(String argv[], Configuration conf)

处理-cp"类型的操作，Copy file(s) to a destination file. Multiple source files can be specified. The destination is the last element of the argvp[] array.

1. **void** delete(String srcf, **final** **boolean** recursive, **final** **boolean** skipTrash)

处理-rm类型的请求：Delete all files that match the file pattern srcf

* 1. 使用DelayedExceptionThrowing调用delete(p, srcFs, recursive, skipTrash);

1. **private** **void** delete(Path src, FileSystem srcFs, **boolean** recursive, **boolean** skipTrash)
   1. 如果recursive为true，要求src必须为目录
   2. 获取Trash trashTmp = **new** Trash(srcFs, getConf());
   3. 如果skipTrash为false，调用trashTmp.moveToTrash(src)把要删除的文件移动到回收站，然后return
   4. 如果skipTrash为true，则调用srcFs.delete(src, **true**)（因为之前已经检查过src是目录的情况，所以此处可以递归删除）
2. **private** **void** tail(String[] cmd, **int** pos)

处理-tail操作，显示指定文件结尾的1k数据，如果加上参数f，则表示动态的显示文件的更新，类似于linux的tail命令

* 1. 初始化CommandFormat c = **new** CommandFormat("tail", 1, 1, "f");
  2. 解析参数List<String> parameters = c.parse(cmd, pos);
  3. 读取文件对象src = parameters.get(0);
  4. 读取参数**boolean** foption = c.getOpt("f") ? **true**: **false**;
  5. 读取文件长度**long** fileSize = srcFs.getFileStatus(path).getLen();，并根据长度设置要打印的起始位置：**long** offset = (fileSize > 1024) ? fileSize - 1024: 0;（即从末尾往前1kb开始打印）
  6. 死循环**while** (**true**)：
     1. 打开流FSDataInputStream in = srcFs.open(path);并seek到指定位置：in.seek(offset);
     2. 打印到输出流：IOUtils.*copyBytes*(in, System.*out*, 1024, **false**);
     3. 设置offset = in.getPos();
     4. 关闭in.close();
     5. 如果foption为false，则在此处退出循环，否则继续执行
     6. 再次查询文件的长度fileSize = srcFs.getFileStatus(path).getLen();
     7. 并设置offset = (fileSize > offset) ? offset: fileSize;（这一步，如果文件长度小于等于offset，则设置offset为文件长度，否则不变，这里设置之后又会循环到上面，1）中会seek到这里指定的offset位置，并打印到文件结尾，所以除非文件长度变长，其他情况都不可能打印更多的内容到输出流。注意，此处并没有锁住文件，所以很有可能被其他线程修改，而导致这一步执行结束后文件长度变短，从而引起下次循环中seek的出错）
     8. 睡眠5秒后继续循环Thread.*sleep*(5000);

1. **private** **int** doall(String cmd, String argv[], **int** startindex)

Apply operation specified by 'cmd' on all parametersstarting from argv[startindex].因为这里的操作都可以同时操作很多文件，所以在一起处理

* 1. 遍历剩余的所有操作数，执行cmd指定的操作

1. **public** **int** run(String argv[])
   1. 获取主命令：String cmd = argv[i++];
   2. 根据命令判断参数数量是否正确
   3. 初始化init();
   4. 根据主命令调用对应的方法
2. **public** **static** **void** main(String argv[])

调用ToolRunner.*run*(shell, argv);运行，解析参数后会调用run方法

## TextRecordInputStream

## CmdHandler

/\*\*

\* This class runs a command on a given FileStatus. This can be used for

\* running various commands like chmod, chown etc.

\*/

属性：

1. **protected** **int** errorCode = 0;
2. **protected** **boolean** okToContinue = **true**;
3. **protected** String cmdName

## DelayedExceptionThrowing

# FsShellPermissions

/\*\*

\* This class is the home for file permissions related commands.

\* Moved to this seperate class since FsShell is getting too large.

\*/

# Command

/\*\*

\* An abstract class for the execution of a file system command

\*/

## Count

/\*\*

\* Count the number of directories, files, bytes, quota, and remaining quota.

\*/

方法：

**protected** **void** run(Path path)

调用fs.getContentSummary(path)获取目录的统计信息后格式化输出