# HDFS文件系统概述

created by:liuchengwu

## 一、FileSystem

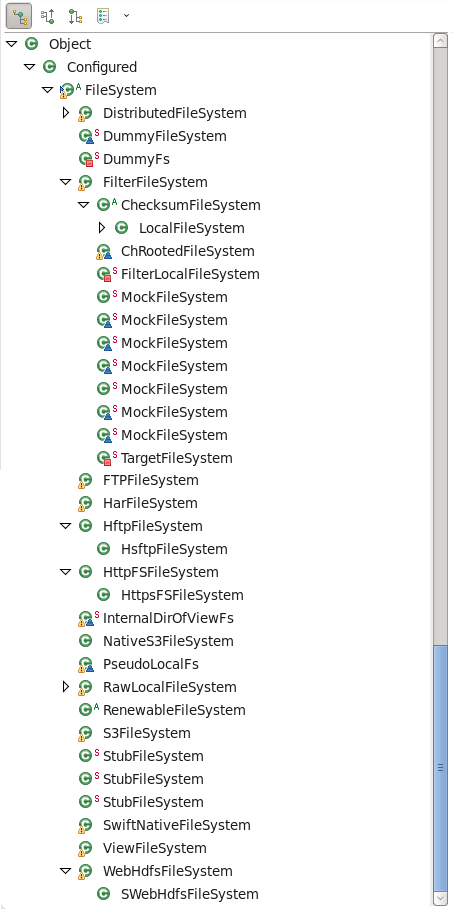
org.apache.hadoop.fs.FileSystem是hadoop的抽象文件系统，为不同的数据访问提供了统一的接口，并提供了大量具体文件系统的实现，满足hadoop上各种数据访问需求

如以下几个具体实现（原表格见《hadoop权威指南》）：

|  |  |  |  |
| --- | --- | --- | --- |
| **文件系统** | **URI方案** | **Java实现**  **（org.apache.hadoop）** | **定义** |
| Local | file | fs.LocalFileSystem | 支持有客户端校验和本地文件系统。带有校验和的本地系统文件在fs.RawLocalFileSystem中实现。 |
| HDFS | hdfs | hdfs.DistributionFileSystem | Hadoop的分布式文件系统。 |
| HFTP | hftp | hdfs.HftpFileSystem | 支持通过**HTTP方式以只读的方式访问HDFS，distcp经常用在不同的HDFS集群间复制数据。** |
| HSFTP | hsftp | hdfs.HsftpFileSystem | 支持通过**HTTPS方式以只读的方式访问HDFS。** |
| HAR | har | fs.HarFileSystem | 构建在Hadoop文件系统之上，对文件进行归档。Hadoop归档文件主要用来**减少NameNode的内存使用。** |
| KFS | kfs | fs.kfs.KosmosFileSystem | Cloudstore（其前身是Kosmos文件系统）文件系统是**类似于HDFS和Google的GFS文件系统，使用C++编写。** |
| FTP | ftp | fs.ftp.FtpFileSystem | 由FTP服务器支持的文件系统。 |
| S3（本地） | s3n | fs.s3native.NativeS3FileSystem | 基于Amazon S3的文件系统。 |
| S3（基于块） | s3 | fs.s3.NativeS3FileSystem | 基于Amazon S3的文件系统，以块格式存储解决了S3的5GB文件大小的限制。 |

在环境搭建时，我们配置fs.defaultFS属性值为hdfs://localhost:9000，即已指定文件系统为HDFS系统。

通过源码，可以查看FileSystem类的层次结构如下



## 二、java接口

文件系统的方法分为两类：一部分处理文件和目录；一部分读写文件数据。

hadoop抽象文件系统的文件操作与java、linux的对应关系（原表格见《Hadoop技术内幕 深入解析HADOOP COMMON和HDFS架构设计与实现原理》）：

|  |  |  |  |
| --- | --- | --- | --- |
| **Hadoop的FileSystem** | **Java操作** | **Linux操作** | **描述** |
| URL.openSteam  FileSystem.open  FileSystem.create  FileSystem.append | URL.openStream | open | 打开一个文件 |
| FSDataInputStream.read | InputSteam.read | read | 读取文件中的数据 |
| FSDataOutputStream.write | OutputSteam.write | write | 向文件写入数据 |
| FSDataInputStream.close  FSDataOutputStream.close | InputSteam.close  OutputSteam.close | close | 关闭一个文件 |
| FSDataInputStream.seek | RandomAccessFile.seek | lseek | 改变文件读写位置 |
| FileSystem.getFileStatus  FileSystem.get\* | File.get\* | stat | 获取文件/目录的属性 |
| FileSystem.set\* | File.set\* | Chmod等 | 改变文件的属性 |
| FileSystem.createNewFile | File.createNewFile | create | 创建一个文件 |
| FileSystem.delete | File.delete | remove | 从文件系统中删除一个文件 |
| FileSystem.rename | File.renameTo | rename | 更改文件/目录名 |
| FileSystem.mkdirs | File.mkdir | mkdir | 在给定目录下创建一个子目录 |
| FileSystem.delete | File.delete | rmdir | 从一个目录中删除一个空的子目录 |
| FileSystem.listStatus | File.list | readdir | 读取一个目录下的项目 |
| FileSystem.getWorkingDirectory |  | getcwd/getwd | 返回当前工作目录 |
| FileSystem.setWorkingDirectory |  | chdir | 更改当前工作目录 |

### 一. 获取文件系统实例

通过FileSystem的get()或newInstance()方法获取文件系统的实例。

get()和newInstance()方法分别有3个重载方法：

|  |  |
| --- | --- |
|  | //返回默认文件系统，core-site.xml中指定的，如果没有指定，则默认本地文件系统  public static FileSystem get(Configuration conf) throws IOException  public static FileSystem newInstance(Configuration conf) throws IOException    //通过给定URI方案和权限来确定要使用的文件系统，若URI中未指定方案，返回默认文件系统  public static FileSystem get(URI uri, Configuration conf) throws IOException  public static FileSystem newInstance(URI uri, Configuration conf) throws IOException    //作为给定用户来访问文件系统，对安全来说很重要  public static FileSystem get(final URI uri, final Configuration conf, final String user)  throws IOException, InterruptedException  public static FileSystem newInstance(final URI uri, final Configuration conf, final String user) throws IOException, InterruptedException |

另外可以通过getLocal()或newInstanceLocal()获取本地文件系统：

|  |  |
| --- | --- |
|  | public static LocalFileSystem getLocal(Configuration conf) throws IOException  public static LocalFileSystem newInstanceLocal(Configuration conf) throws IOException |

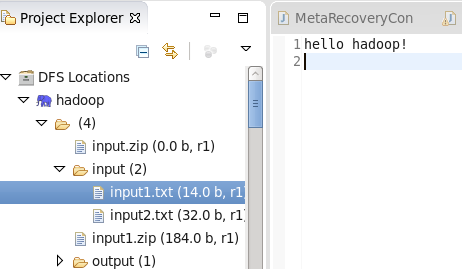
### 二. 读取数据

#### 1. 从hadoop url读取数据

读取文件最简单的方法是使用java.net.URL对象打开数据流，从中读取数据，但让java程序能识别hadoop的hdfs url需要通过FsUrlStreamHandlerFactory实例调用java.net.URL对象的setURLStreamHandlerFactory方法。

##### 例1：

 HDFS中有一个/input/input1.txt文件，文件内容“hello hadoop!”

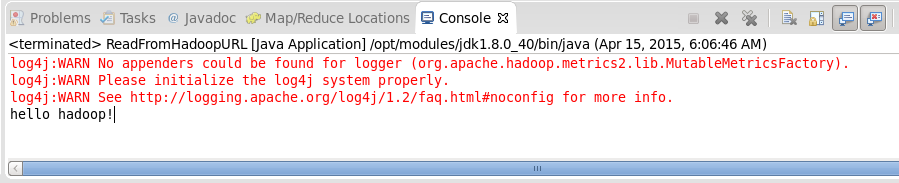


 java测试类代码：

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | public class ReadFromHadoopURL {      static {      URL.setURLStreamHandlerFactory(new FsUrlStreamHandlerFactory());    }      public static void main(String[] args) throws Exception{        String uri = "<hdfs://localhost:9000/input/input1.txt>";        InputStream in = null;          try{            in = new URL(uri).openStream();            IOUtils.copyBytes(in, System.out, 4096, false);        }finally{            IOUtils.closeStream(in);        }    }  } |

 运行结果：



**注：**

这种文件读取的方法具有一定的限制性。因为Java.net.URL的setURLStreamHandlerFactory方法每个java虚拟机最多调用一次，如果程序中有不受自己控制的第三方组件调用了这个方法，将无法使用这种方法从hadoop中读取数据。

附setURLStreamHandlerFactory源码：

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | public static void setURLStreamHandlerFactory(URLStreamHandlerFactory fac) {          synchronized (streamHandlerLock) {              if (factory != null) {                  throw new Error("factory already defined");              }                SecurityManager security = System.getSecurityManager();              if (security != null) {                  security.checkSetFactory();              }              handlers.clear();              factory = fac;          }    } |

#### 2.通过FileSystem API读取数据

hadoop文件系统中通过org.apache.hadoop.fs.Path对象来代表文件。

获取到FileSystem实例后通过open()方法获取文件的输入流

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3 | //缓冲区默认大小4KB，bufferSize指定缓冲区大小  public FSDataInputStream open(Path f) throws IOException  public abstract FSDataInputStream open(Path f, int bufferSize) throws IOException; |

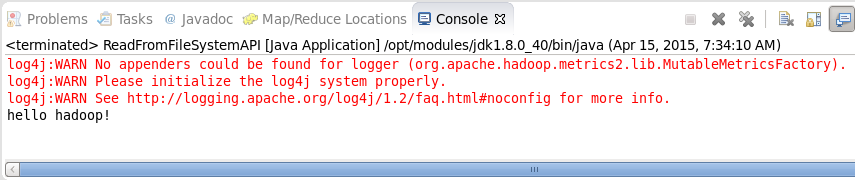
##### 例2：

java测试类代码：

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | public class ReadFromFileSystemAPI {      public static void main(String[] args) throws Exception{          String uri = "<hdfs://localhost:9000/input/input1.txt>";          Configuration conf = new Configuration();          FileSystem fs = FileSystem.get(URI.create(uri), conf);          //第二种获取文件系统的方法          //FileSystem fs = FileSystem.newInstance(URI.create(uri), conf);          InputStream in = null;          try{              in = fs.open(new Path(uri));              IOUtils.copyBytes(in, System.out, 4096, false);          }finally{              IOUtils.closeStream(in);          }      }    } |

运行结果：



#### 输入流FSDataInputStream对象介绍

FileSystem对象中的open()方法返回的是org.apache.hadoop.fs.FSDataInputStream对象，这个对象继承了java.io.DataInputStream，并支持随机访问，从流的任意位置读取数据。

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public class FSDataInputStream extends DataInputStream      implements Seekable, PositionedReadable,        ByteBufferReadable, HasFileDescriptor, CanSetDropBehind, CanSetReadahead,        HasEnhancedByteBufferAccess{    //implementation    } |

Seekable接口支持在文件中找到指定位置，并提供一个查询当前位置相对于文件起始位置偏移量的方法。注：seek()方法开销相对高，需要慎用。

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | public interface Seekable {      //定位到从文件起始位置开始指定的偏移量的位置，若偏移量超出文件位置会报异常    void seek(long pos) throws IOException;         //返回当前位置相对于文件起始位置的偏移量    long getPos() throws IOException;        //查找数据的其他副本，若找到一个新副本则返回true，否则返回false    boolean seekToNewSource(long targetPos) throws IOException;  } |

PositionedReadable接口从一个指定偏移量处读取文件的一部分。

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | public interface PositionedReadable {    //从文件指定position处读取至多length字节的数据，并存入缓冲区buffer的指定偏移量offset处   //返回值是督导的字节数，可能比length的长度小    public int read(long position, byte[] buffer, int offset, int length) throws IOException;       //从文件指定position处读取指定length的字节，并存入缓冲区buffer指定偏移量offset处    //若读到文件末尾仍不足length字节，则抛出异常    public void readFully(long position, byte[] buffer, int offset, int length) throws IOException;      //从文件指定position处读取缓冲区buffer大小的字节，并存入buffer    //若读到文件末尾仍不足length字节，则抛出异常    public void readFully(long position, byte[] buffer) throws IOException;  } |

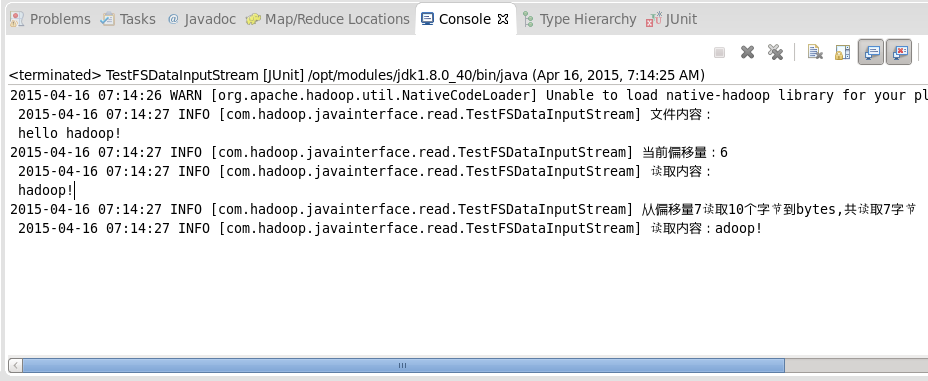
##### 例3：

测试代码：

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
|  | public class TestFSDataInputStream {      private FileSystem fs = null;      private FSDataInputStream in = null;      private String uri = "<hdfs://localhost:9000/input/input1.txt>";        private Logger log = Logger.getLogger(TestFSDataInputStream.class);      static{          PropertyConfigurator.configure("conf/log4j.properties");      }        @Before      public void setUp() throws Exception {          Configuration conf = new Configuration();          fs = FileSystem.get(URI.create(uri), conf);      }        @Test      public void test() throws Exception{          try{              in = fs.open(new Path(uri));                log.info("文件内容：");              IOUtils.copyBytes(in, System.out, 4096, false);                in.seek(6);              Long pos = in.getPos();              log.info("当前偏移量："+pos);              log.info("读取内容：");              IOUtils.copyBytes(in, System.out, 4096, false);                byte[] bytes = new byte[10];              int num = in.read(7, bytes, 0, 10);              log.info("从偏移量7读取10个字节到bytes,共读取"+num+"字节");              log.info("读取内容："+(new String(bytes)));                //以下代码会抛出EOFException  //          in.readFully(6, bytes);  //          in.readFully(6, bytes, 0, 10);          }finally{              IOUtils.closeStream(in);          }      }  } |

 运行结果：



### 三. 写入数据

#### 1.新建文件

给准备建的文件指定一个Path对象，然后通过FileSystem的create()方法返回一个用于写入数据的输出流。

Create()方法有多个重载版本，允许指定是否需要强制覆盖现有文件、文件备份数量、写入文件时缓冲区大小、文件块大小及文件权限。还可指定Progressable回调接口，这样可以把数据写入datanode的进度通知给应用。

Create()方法能为需要写入且当前不存在的文件创建父目录，若不希望这样，则应先调用exists()方法检查父目录是否存在。

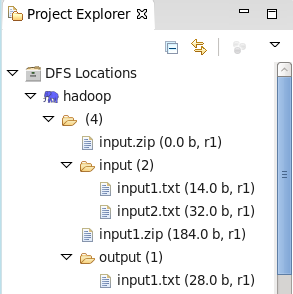
create()方法的所有重载方法：

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | //创建一个输出流，默认覆盖现有文件  public FSDataOutputStream create(Path f) throws IOException      //创建一个输出流，文件存在时，overwrite为true则覆盖现有文件，为false则抛出异常  public FSDataOutputStream create(Path f, boolean overwrite) throws IOException     //创建一个输出流，默认覆盖现有文件，progress用来报告进度  public FSDataOutputStream create(Path f, Progressable progress)  throws IOException      //创建一个输出流，默认覆盖现有文件，replication指定文件备份数  public FSDataOutputStream create(Path f, short replication) throws IOException      //创建一个输出流，默认覆盖现有文件，replication指定文件备份数，progress用来报告进度  public FSDataOutputStream create(Path f, short replication, Progressable progress)  throws IOException      //创建一个输出流，文件存在时，overwrite为true则覆盖现有文件，为false则抛出异常   //bufferSize指定写入时缓冲区大小  public FSDataOutputStream create(Path f, boolean overwrite, int bufferSize) throws IOException      //创建一个输出流，文件存在时，overwrite为true则覆盖现有文件，为false则抛出异常   // bufferSize指定写入时缓冲区大小，replication指定文件备份数，blockSize指定文件块大小  public FSDataOutputStream create(Path f, boolean overwrite, int bufferSize, short replication, long blockSize ) throws IOException      //创建一个输出流，文件存在时，overwrite为true则覆盖现有文件，为false则抛出异常   // bufferSize指定写入时缓冲区大小，replication指定文件备份数，blockSize指定文件块大小   // progress用来报告进度  public FSDataOutputStream create(Path f, boolean overwrite, int bufferSize,short replication, long blockSize,  Progressable progress ) throws IOException      //创建一个输出流，文件存在时，overwrite为true则覆盖现有文件，为false则抛出异常   // bufferSize指定写入时缓冲区大小，replication指定文件备份数，blockSize指定文件块大小   // progress用来报告进度，permission指定文件权限  public abstract FSDataOutputStream create(Path f, FsPermission permission, boolean overwrite, int bufferSize, short replication, long blockSize, Progressable progress)  throws IOException;      //创建一个输出流，permission指定文件权限， bufferSize指定写入时缓冲区大小   // replication指定文件备份数，progress用来报告进度   // flags指定创建标志，标志如下：   //          CREATE - 如果文件不存在则创建文件，否则抛出异常   //          APPEND - 如果文件存在则向文件追加内容，否则抛出异常   //          OVERWRITE - 文件存在时，覆盖现有文件，否则抛出异常   //          CREATE|APPEND - 文件不存在时创建文件，文件已存在时向文件追加内容   //          CREATE|OVERWRITE - 文件不存在时创建文件，否则覆盖已有文件   //          SYNC\_BLOCK - 强制关闭文件块，如果需要同步操作，每次写入后还需调用Syncable.hsync()方法  public FSDataOutputStream create(Path f, FsPermission permission, EnumSet<CreateFlag> flags, int bufferSize, short replication, long blockSize, Progressable progress) throws IOException      //创建一个输出流，permission指定文件权限， bufferSize指定写入时缓冲区大小   // replication指定文件备份数，progress用来报告进度，blockSize指定文件块大小   // checksumOpt指定校验和选项，若为空，则使用配置文件中的值   // flags指定创建标志，标志如下：   //          CREATE - 如果文件不存在则创建文件，否则抛出异常   //          APPEND - 如果文件存在则向文件追加内容，否则抛出异常   //          OVERWRITE - 文件存在时，覆盖现有文件，否则抛出异常   //          CREATE|APPEND - 文件不存在时创建文件，文件已存在时向文件追加内容   //          CREATE|OVERWRITE - 文件不存在时创建文件，否则覆盖已有文件   //          SYNC\_BLOCK - 强制关闭文件块，如果需要同步操作，每次写入后还需调用Syncable.hsync()方法  public FSDataOutputStream create(Path f, FsPermission permission, EnumSet<CreateFlag> flags, int bufferSize, short replication, long blockSize, Progressable progress, ChecksumOpt checksumOpt)  throws IOException |

##### 例4：

写入前HDFS中目录结构：

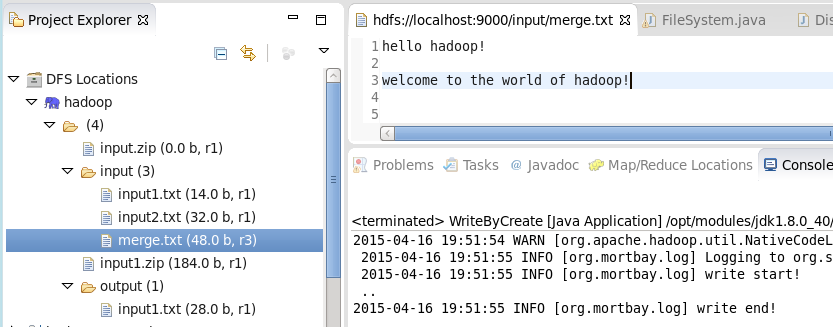


测试代码：

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | public class WriteByCreate {      static{          PropertyConfigurator.configure("conf/log4j.properties");      }        @Test      public void createTest() throws Exception {          String localSrc = "/home/hadoop/merge.txt";          String dst = "<hdfs://localhost:9000/input/merge.txt>";            InputStream in = new BufferedInputStream(new FileInputStream(localSrc));          Configuration conf = new Configuration();            FileSystem fs = FileSystem.get(URI.create(dst), conf);          OutputStream out = null;          try{              out = fs.create(new Path(dst),                      new Progressable() {                          public void progress() {                              System.out.print(".");                          }                      });              Log.info("write start!");              IOUtils.copyBytes(in, out, 4096, true);              System.out.println();              Log.info("write end!");          }finally{              IOUtils.closeStream(in);              IOUtils.closeStream(out);          }      }  } |

运行结果：



#### 2.向已存在文件末尾追加数据

 FileSystem的append()方法允许在一个已存在文件的最后偏移量处追加数据。追加操作是可选的，并不是所有hadoop文件系统都实现了该操作。

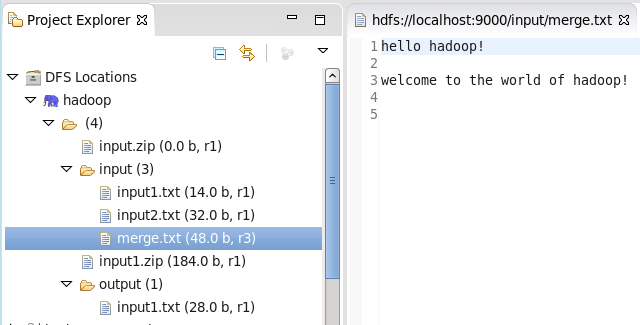
Append()的重载方法

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | //向指定文件中追加数据，默认缓冲区大小4096，文件不存在时抛出异常  public FSDataOutputStream append(Path f) throws IOException    //向指定文件中追加数据，bufferSize指定缓冲区大小，文件不存在时抛出异常  public FSDataOutputStream append(Path f, int bufferSize) throws IOException    //向指定文件中追加数据，bufferSize指定缓冲区大小，文件不存在时抛出异常，progress报告进度  public abstract FSDataOutputStream append(Path f, int bufferSize, Progressable progress) throws IOException; |

##### 例5：

追加前：

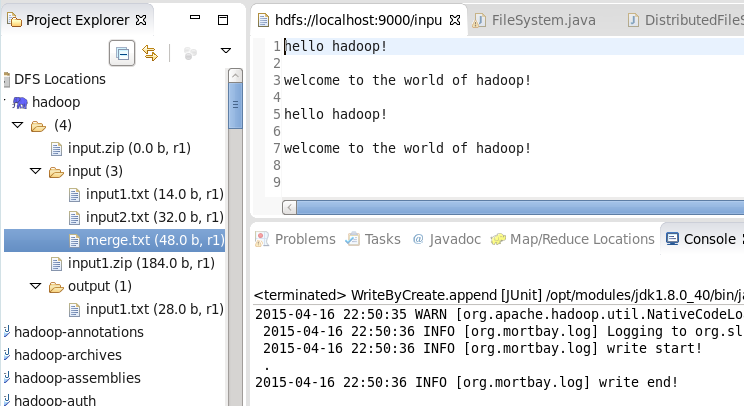


测试代码

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | public class WriteByAppend{      static{          PropertyConfigurator.configure("conf/log4j.properties");      }        @Test      public void appendTest() throws Exception {          String localSrc = "/home/hadoop/merge.txt";          String dst = "<hdfs://localhost:9000/input/merge.txt>";            InputStream in = new BufferedInputStream(new FileInputStream(localSrc));          Configuration conf = new Configuration();            FileSystem fs = FileSystem.get(URI.create(dst), conf);          OutputStream out = null;          try{              out = fs.append(new Path(dst),4096,                               new Progressable() {                                  public void progress() {                                      System.out.print(".");                                  }                              });              Log.info("write start!");              IOUtils.copyBytes(in, out, 4096, true);              System.out.println();              Log.info("write end!");          }finally{              IOUtils.closeStream(in);              IOUtils.closeStream(out);          }      }  } |

 执行结果



例6：hdfs提供的JavaAPI来编写一个上传多个文件的过程中合成一个大文件的程序

|  |
| --- |
| **import** java.io.IOException;  **import** org.apache.hadoop.conf.Configuration;  **import** org.apache.hadoop.fs.FSDataInputStream;  **import** org.apache.hadoop.fs.FSDataOutputStream;  **import** org.apache.hadoop.fs.FileStatus;  **import** org.apache.hadoop.fs.FileSystem;  **import** org.apache.hadoop.fs.Path;    **public** **class** putMerge {  **public** **static** **void** main(String[] args) **throws** IOException {  Configuration conf = **new** Configuration();  FileSystem hdfs = FileSystem.get(conf);  FileSystem fs = FileSystem.getLocal(conf);    Path inputDir = **new** Path(args[0]);  Path hdfsFile = **new** Path(args[1]);    **try** {  FileStatus[] inputFiles = fs.listStatus(inputDir);  FSDataOutputStream out = hdfs.create(hdfsFile);    **for** (**int** i = 0; i <inputFiles.length; i++) {  System.out.println(inputFiles[i].getPath().getName());  FSDataInputStream in = local.open(inputFiles[i].getPath());  **byte** buffer[] = **new** **byte**[256];  **int** bytesRead = 0;  **while** ((bytesRead = in.read(buffer)) &gt; 0) {  out.write(buffer, 0, bytesRead);  }  in.close();  }  out.close();  } **catch** (IOException e) {  e.printStackTrace();  }    }  } |

一般为了方便，我们可以类似的给这个程序起个别名，为了连贯性，我这里把该程序打包放在$HADOOP\_HOME/bin/putMerge.jar这个路径下，然后仿照[这篇里边](http://blog.pureisle.net/?p=1600)说的起别名规则，在/etc/profile里边添加一条新的别名：

alias hputm=’hadoop jar $HADOOP\_HOME/bin/putMerge.jar putMerge’

然后这个命令的使用方法就是：

hputm input(本地目录名) hdfsoutputfilename

#### 输出流FSDataOutputStream对象

FileSystem的create()方法及append()方法返回的是FSDataOutputStream对象，它也有一个查询文件当前位置的方法getPos()。与FSDataInputStream不同，FSDataOutputStream不允许在文件中定位，因为HDFS只允许对一个已打开的文件顺序写入，或在现有文件末尾追加数据，不支持在除文件末尾外的其他位置进行写入，因此写入时定位没有意义。

### 四. 创建目录

FileSystem提供了创建目录的方法。可以一次性创建所有必要但还没有的父目录。

 public boolean mkdirs(Path f) throws IOException

 public abstract boolean mkdirs(Path f, FsPermission permission ) throws IOException;

通常不需要显示创建一个目录，因为调用create()方法写入文件时会自动创建父目录。

### 五. 查询文件系统

#### 文件元数据FileStatus

FileStatus类封装了文件系统中文件和目录的元数据，FileStatus源码中可以看到如下属性

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | public class FileStatus implements Writable, Comparable {      private Path path;//文件或目录的path    private long length;//文件字节数    private boolean isdir;//是否是目录    private short block\_replication;//文件块备份数    private long blocksize;//文件块大小    private long modification\_time;//修改时间    private long access\_time;//访问时间    private FsPermission permission;//权限    private String owner;//所属用户    private String group;//所属用户组    private Path symlink; //软连接        //method  } |

FileSystem的getFileStatus()方法用于获取文件或目录的FileStatus对象

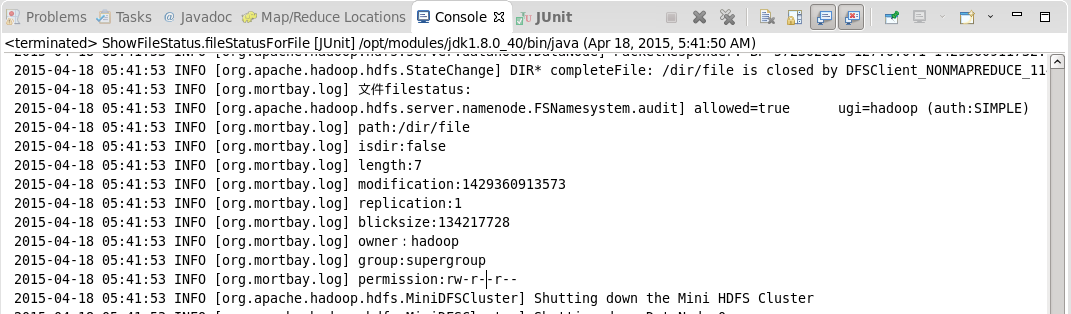
##### 例6：

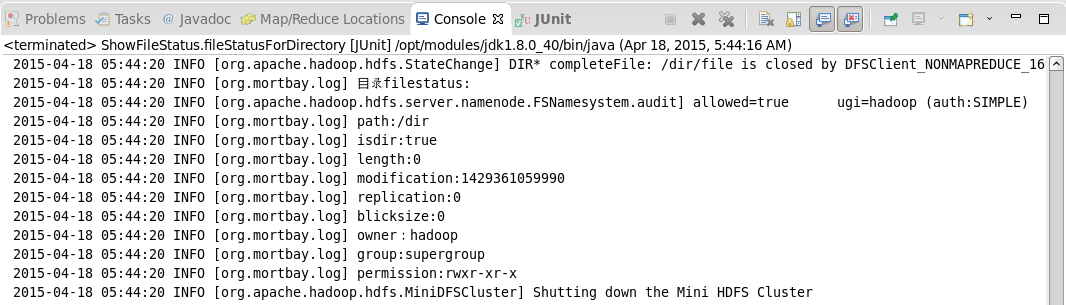
测试代码：

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64 | public class ShowFileStatus {      private MiniDFSCluster cluster; // use an in-process HDFS cluster for testing      private FileSystem fs;        @Before      public void setUp() throws IOException {          Configuration conf = new Configuration();          if (System.getProperty("test.build.data") == null) {              System.setProperty("test.build.data", "/tmp");          }          cluster = new MiniDFSCluster(conf, 1, true, null);          fs = cluster.getFileSystem();          OutputStream out = fs.create(new Path("/dir/file"));          out.write("content".getBytes("UTF-8"));          out.close();      }        @After      public void tearDown() throws IOException {          if (fs != null) {              fs.close();          }          if (cluster != null) {              cluster.shutdown();          }      }        @Test(expected = FileNotFoundException.class)      public void throwsFileNotFoundForNonExistentFile() throws IOException {          fs.getFileStatus(new Path("no-such-file"));      }        @Test      public void fileStatusForFile() throws IOException {          Path file = new Path("/dir/file");          Log.info("文件filestatus:");          FileStatus stat = fs.getFileStatus(file);          Log.info("path:"+stat.getPath().toUri().getPath());          Log.info("isdir:"+String.valueOf(stat.isDir()));          Log.info("length:"+String.valueOf(stat.getLen()));          Log.info("modification:"+String.valueOf(stat.getModificationTime()));          Log.info("replication:"+String.valueOf(stat.getReplication()));          Log.info("blicksize:"+String.valueOf(stat.getBlockSize()));          Log.info("owner："+stat.getOwner());          Log.info("group:"+stat.getGroup());          Log.info("permission:"+stat.getPermission().toString());      }        @Test      public void fileStatusForDirectory() throws IOException {          Path dir = new Path("/dir");          Log.info("目录filestatus:");          FileStatus stat = fs.getFileStatus(dir);          Log.info("path:"+stat.getPath().toUri().getPath());          Log.info("isdir:"+String.valueOf(stat.isDir()));          Log.info("length:"+String.valueOf(stat.getLen()));          Log.info("modification:"+String.valueOf(stat.getModificationTime()));          Log.info("replication:"+String.valueOf(stat.getReplication()));          Log.info("blicksize:"+String.valueOf(stat.getBlockSize()));          Log.info("owner："+stat.getOwner());          Log.info("group:"+stat.getGroup());          Log.info("permission:"+stat.getPermission().toString());      }  } |

 运行结果：





#### 列出文件

列出目录中内容，可以使用FileSystem的listStatus()方法。方法接收一个或一组路径，如果路径是文件，以数组方法返回长度为1的FileStatus对象，如果路径是目录，返回0个或多个FileStatus对象表示目录中包含的文件或目录；如果是一组路径，依次轮流对每个路径调用listStatus方法，将结果累积到一个数组

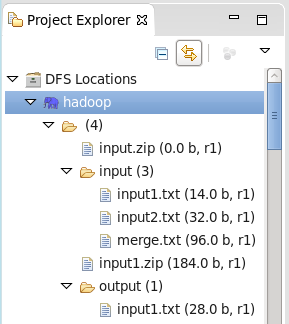
方法如下

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | //列出给定路径下的文件或目录的status  public abstract FileStatus[] listStatus(Path f)  throws FileNotFoundException, IOException;  //列出给定路径下符合用户提供的filter限制的文件或目录的status  public FileStatus[] listStatus(Path f, PathFilter filter) throws FileNotFoundException, IOException  //列出给定的一组路径下文件或目录的status  public FileStatus[] listStatus(Path[] files) throws FileNotFoundException, IOException  //列出给定的一组路径下符合用户提供的filter限制的文件或目录的status  public FileStatus[] listStatus(Path[] files, PathFilter filter)  throws FileNotFoundException, IOException |

##### 例7：

目录结构：

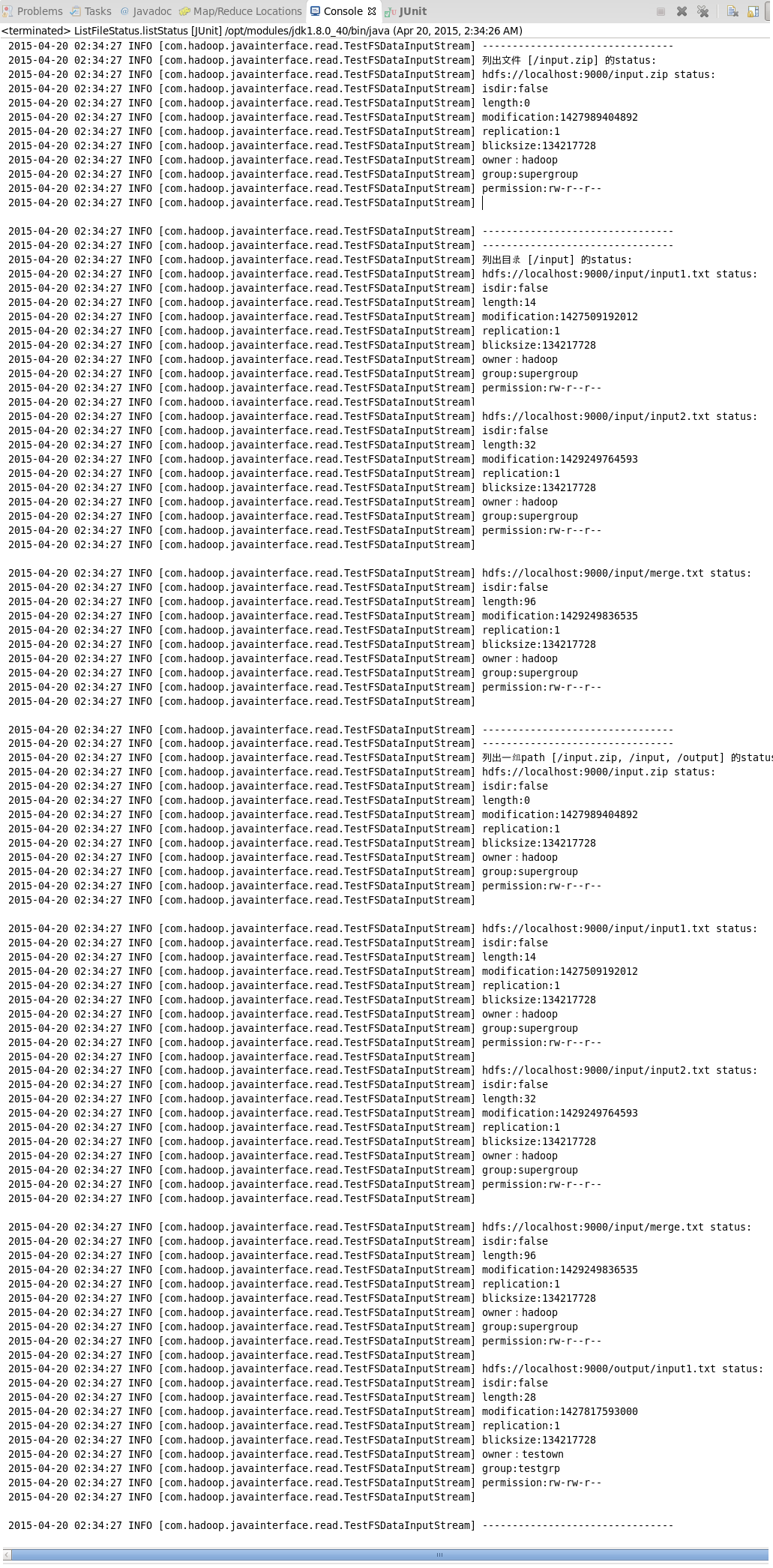


测试代码：

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52 | public class ListFileStatus {      private FileSystem fs = null;      private String uri = "<hdfs://localhost:9000/input/input1.txt>";      private Path[] paths = new Path[]{new Path("/input.zip"),new Path("/input/"),new Path("/output/")};        private Logger log = Logger.getLogger(TestFSDataInputStream.class);      static{          PropertyConfigurator.configure("conf/log4j.properties");      }        @Before      public void setUp() throws Exception {          Configuration conf = new Configuration();          fs = FileSystem.get(URI.create(uri), conf);      }        @Test      public void listStatusTest() throws Exception {          log.info("--------------------------------");          log.info("列出文件 ["+paths[0]+"] 的status:");          FileStatus[] status = fs.listStatus(paths[0]);          printFileStatus(status);          log.info("--------------------------------");            log.info("--------------------------------");          log.info("列出目录 ["+paths[1]+"] 的status:");          status = fs.listStatus(paths[1]);          printFileStatus(status);          log.info("--------------------------------");            log.info("--------------------------------");          log.info("列出一组path "+Arrays.toString(paths)+" 的status:");          status = fs.listStatus(paths);          printFileStatus(status);          log.info("--------------------------------");      }        protected void printFileStatus(FileStatus[] status){          for (FileStatus s : status) {              log.info(s.getPath()+" status:");              log.info("isdir:"+String.valueOf(s.isDir()));              log.info("length:"+String.valueOf(s.getLen()));              log.info("modification:"+String.valueOf(s.getModificationTime()));              log.info("replication:"+String.valueOf(s.getReplication()));              log.info("blicksize:"+String.valueOf(s.getBlockSize()));              log.info("owner："+s.getOwner());              log.info("group:"+s.getGroup());              log.info("permission:"+s.getPermission().toString());              log.info("\n");          }      }  } |

 测试结果：



另外，需要在一次操作中处理一批文件时，hadoop提供了通配符来匹配多个文件。

|  |  |  |
| --- | --- | --- |
| 通配符 | 名称 | 匹配 |
| \* | 星号 | 匹配0或多个字符 |
| ? | 问号 | 匹配单衣字符 |
| [ab] | 字符类 | 匹配{a,b}集合里的一个字符 |
| [^ab] | 非字符类 | 匹配非{a,b}集合里的一个字符 |
| [a-b] | 字符范围 | 匹配一个{a,b}范围内的字符，包括ab，a的字典顺序要小于等于b |
| [^a-b] | 非字符范围 | 匹配一个不在{a,b}范围内的字符，包括ab，a的字典顺序要小于等于b |
| {a,b} | 或选择 | 匹配包含a或b中一个的 |
| \c | 转义字符 | 匹配原字符c |

hadoop的FileSystem为通配提供了2个globStatus()方法，方法返回所有文件路径与给定的通配符相匹配的文件的FileStatus，filter可进一步对匹配进行限制：

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2 | public FileStatus[] globStatus(Path pathPattern) throws IOException  public FileStatus[] globStatus(Path pathPattern, PathFilter filter) throws IOException |

##### 例8：

目录结构

/

├── 2007/

│ └── 12/

│      ├── 30/

│      └── 31/

└── 2008/

      └── 01/

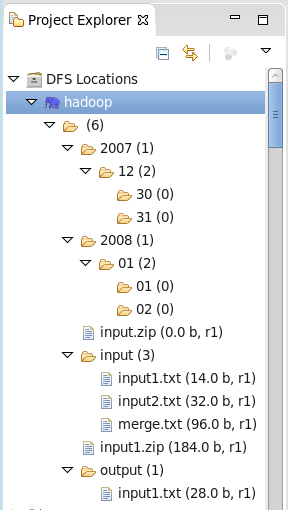
           ├── 01/

           └── 02/

通配符示例：

|  |  |
| --- | --- |
| 通配符 | Expansion |
| /\* | /2007 /2008 |
| /\*/\* | /2007/12 /2008/01 |
| /\*/12/\* | /2007/12/30 /2007/12/31 |
| /200? | /2007 /2008 |
| /200[78] | /2007 /2008 |
| /200[7-8] | /2007 /2008 |
| /200[^01234569] | /2007 /2008 |
| /\*/\*/{31,01} | /2007/12/31 /2008/01/01 |
| /\*/\*/3{0,1} | /2007/12/30 /2007/12/31 |
| /\*/{12/31,01/01} | /2007/12/31 /2008/01/01 |

目录结构：

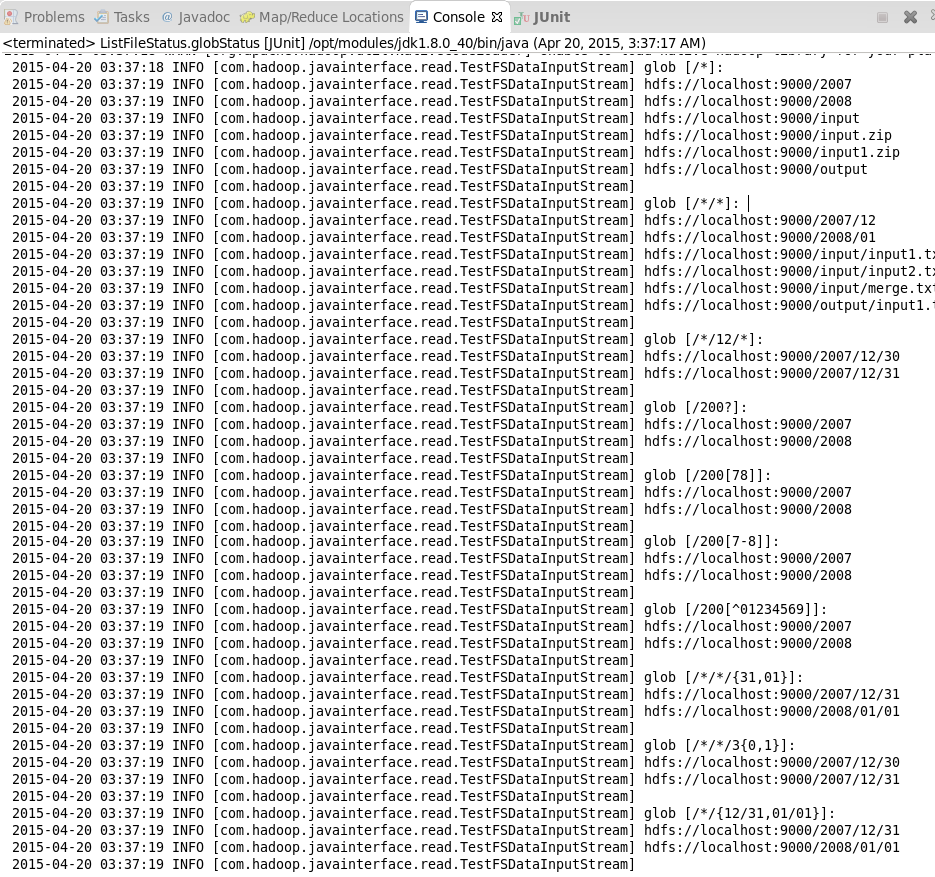


测试代码：

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | public class ListFileStatus {      private FileSystem fs = null;      private String uri = "<hdfs://localhost:9000/input/input1.txt>";      private Path[] globPaths = new Path[]{new Path("/\*"),new Path("/\*/\*"),new Path("/\*/12/\*"),new Path("/200?")                                              ,new Path("/200[78]"),new Path("/200[7-8]"),new Path("/200[^01234569]")                                              ,new Path("/\*/\*/{31,01}"),new Path("/\*/\*/3{0,1}"),new Path("/\*/{12/31,01/01}")};        private Logger log = Logger.getLogger(TestFSDataInputStream.class);      static{          PropertyConfigurator.configure("conf/log4j.properties");      }        @Before      public void setUp() throws Exception {          Configuration conf = new Configuration();          fs = FileSystem.get(URI.create(uri), conf);      }        @Test      public void globStatusTest() throws Exception {          for(Path p:globPaths){              log.info("glob ["+p+"]: ");              FileStatus[] status = fs.globStatus(p);              printFilePath(status);          }      }        protected void printFilePath(FileStatus[] status){          Path[] listedPaths = FileUtil.stat2Paths(status);          for (Path p : listedPaths) {              log.info(p);          }          log.info("");      }  } |

 运行结果：



通配符并不总能精确的描述的描述想要访问的文件集，如使用通配符排除一个特定的文件就不太可能。FileSystem的listStatus()方法和globStatus()方法提供可选的PathFilter对象，以编程方式控制通配符。过滤器只能作用于文件名，不能针对文件属性进行过滤

PathFilter接口：

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3  4 | public interface PathFilter {      boolean accept(Path path);    } |

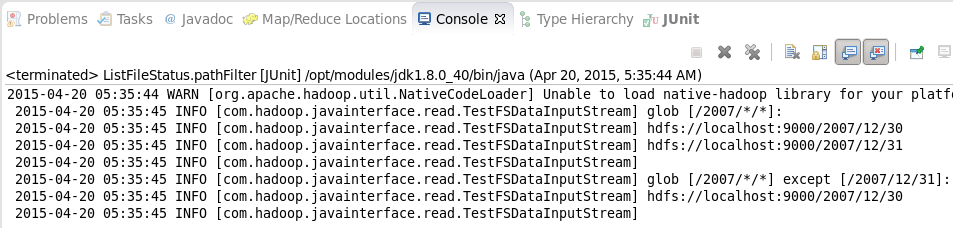
##### 例9：

测试代码

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | public class ListFileStatus {      private FileSystem fs = null;      private String uri = "<hdfs://localhost:9000/input/input1.txt>";        private Logger log = Logger.getLogger(TestFSDataInputStream.class);      static{          PropertyConfigurator.configure("conf/log4j.properties");      }        @Before      public void setUp() throws Exception {          Configuration conf = new Configuration();          fs = FileSystem.get(URI.create(uri), conf);      }        @Test      public void pathFilterTest() throws Exception {          log.info("glob [/2007/\*/\*]: ");          FileStatus[] status = fs.globStatus(new Path("/2007/\*/\*"));          printFilePath(status);            log.info("glob [/2007/\*/\*] except [/2007/12/31]: ");          status = fs.globStatus(new Path("/2007/\*/\*"), new RegexExcludePathFilter("^.\*/2007/12/31$"));          printFilePath(status);      }        protected void printFilePath(FileStatus[] status){          Path[] listedPaths = FileUtil.stat2Paths(status);          for (Path p : listedPaths) {              log.info(p);          }          log.info("");      }        class RegexExcludePathFilter implements PathFilter {          private final String regex;            public RegexExcludePathFilter(String regex) {              this.regex = regex;          }          public boolean accept(Path path) {              return !path.toString().matches(regex);          }      }  } |

 运行结果：



### 六. 删除数据

FileSystem的delete()方法可以永久删除文件或目录。

[?](http://www.cnblogs.com/zhaosk/p/4443095.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public boolean delete(Path f) throws IOException    //recursive为true时，非空目录及其内容才会被删除，否则抛出异常  public abstract boolean delete(Path f, boolean recursive) throws IOException    //标记当文件系统关闭时将删除的文件。当JVM关闭时，被标记的文件将被删除  public boolean deleteOnExit(Path f) throws IOException |