School of Computer Science and Technology, China

University of Mining and Technology

Special Report for Emerging Technology in Computer Science

Tile of Report	**Coding Assignment for Apache Hadoop Tutorial**
Reporting Time	2020-9-17
Student Name	Guo Ziyang
Class	6
Student ID	08173042
Teaching staff	Zhang Bo

1. If a datanode fails while data is being writing to it, what actions HDFS would take to handle this error?

To explain this problem, please let me start with an introduction to the Hadoop file writing process.

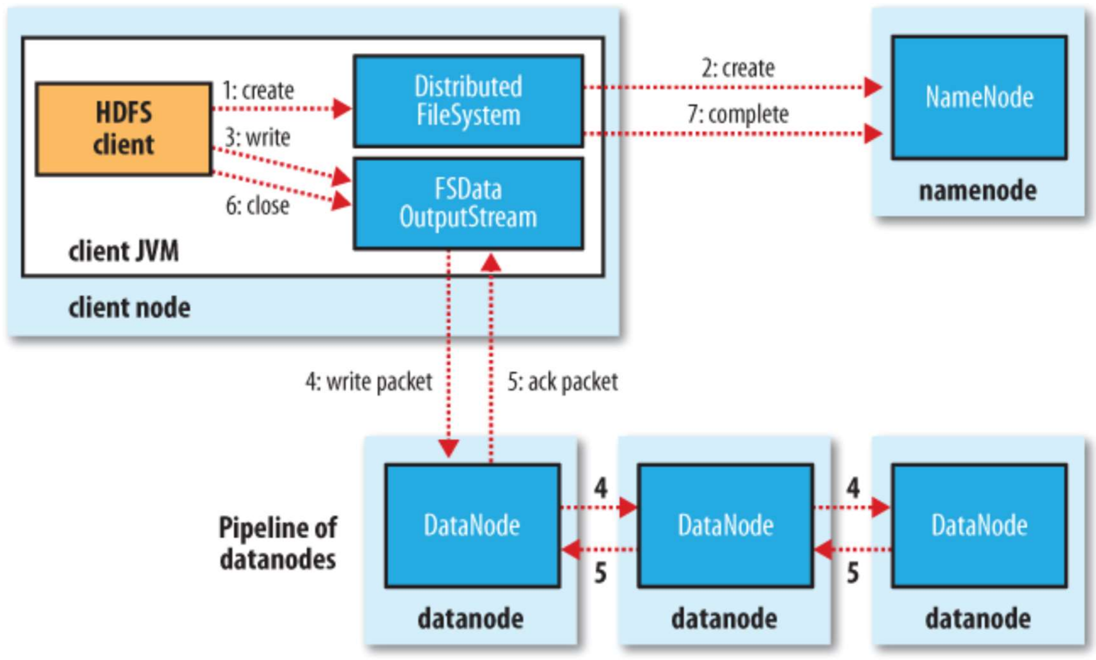


Figure 1-1 A client writing data to HDFS

Step1. The client calls the Create() function through the DistributedFileSystem object to create a new file

Step2. DistributedFIleSystem creates an RPC call to the namenode, creating a file in the namespace of the file system where there is no corresponding data block

Step3. When a client writes data, the DFSOutputStream splits the data into packets and writes to Data queue. DataStreamer deals with data queues by inquiring namenode to find the datanodes which are most suitable to store the new blocks and then arranging them into a pipeline.

Step4. A DataStreamer will export Packet as a queue to the first datanode of the pipeline, the first datanode will export Packet to the second datanode.

Step5. DFSOutputStream also has a queue called ack queue, which is also composed of packet. It waits for the response of the datanode. When all datanodes in the pipeline represent received packets, akc Queue will remove the corresponding packet.

Step6. When the client has finished writing data it calls close() on the stream.

Step7. Writes all remaining packets to the datanode and waits for confirmation before contacting the namenode to tell it that the file write is complete.

The error happens in step5, if a datanode fails while data is being written to it, then the following actions are taken, which are transparent to the client writing the data. First the pipeline is closed, and any packets in the ack queue are added to the front of the data queue so that datanodes that are downstream from the failed node will not miss any packets. The current block on the good datanodes is given a new identity, which is communicated to the namenode, so that the partial block on the failed datanode will be deleted if the failed datanode recovers later on. The failed datanode is removed from the pipeline and the remainder of the block's data is written to the two good datanodes in the pipeline. The namenode notices that the block is under-replicated, and it arranges for a further replica to be created on another node. Subsequent blocks are then treated as normal.

According to my personal experience, since the namenode finds that the number of copies is less than the number configured by us, it is necessary to find a new datanode and then copy the data blocks with insufficient copies to the new datanode. After processing these data blocks, the new data blocks can be processed in the way of pipelines.

It's possible, but unlikely, that multiple datanodes fail while a block is being written. As long as dfs.replication.min replicas (default one) are written the write will succeed, and the block will be asynchronously replicated across the cluster until its target replication factor is reached (dfs.replication, which defaults to three).

2. How to ensure the integrity of data stored on HDFS?

2.1 Checksums

HDFS transparently checksums all data written to it and by default verifies checksums when reading data. The default is 512 bytes, and since a CRC-32 checksum is 4 bytes long, the storage overhead is less than 1%.

2.2 DataBlockScanner

Each datanode runs a DataBlockScanner in a background thread that periodically verifies all the blocks stored on the datanode. This is to guard against corruption due to “bit rot” in the physical storage media.

2.3 ‘Heal’ Corrupted Blocks

Since HDFS stores replicas of blocks, it can “heal” corrupted blocks by copying one of the good replicas to produce a new, uncorrupt replica. The way this works is that if a client detects an error when reading a block, it reports the bad block and the datanode it was trying to read from to the namenode before throwing a ChecksumException. The namenode marks the block replica as corrupt, so it doesn’t direct clients to it, or try to copy this replica to another datanode. It then schedules a copy of the block to be replicated on another datanode, so its replication factor is back at the expected level. Once this has happened, the corrupt replica is deleted.

3 . HDFS is optimized for large data files, and it is inefficient to process small files. Read the introduction of Hadoop‘s Sequence File in the chapter “File-Based Data Structures”. Write code to pack small files into a sequence file. Requirements are as follows :

1. Development Enviroment：Intellj IDEA + MAVEN，JDK 1.8, Hadoop 2.9.2
2. Pack File: Input a directory containing 100 small images (<100KB), Output a sequenceFile containing all pictures
3. Unpack File: Input a sequenceFile containing all pictures, Output a directory containing all small images.

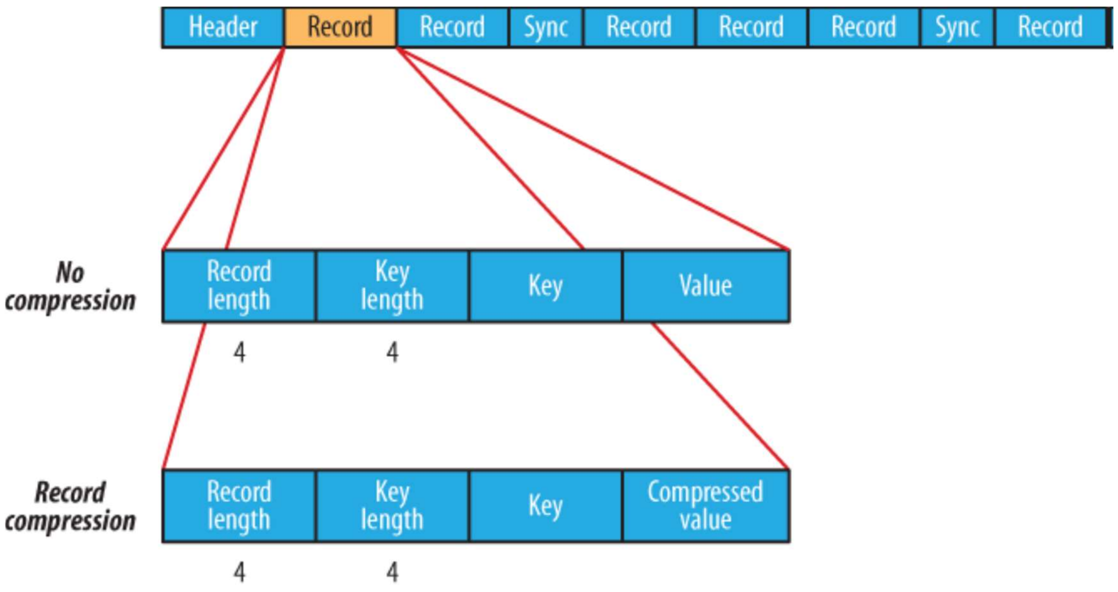


Figure 3-1 The internal structure of a sequence file with no compression and record compression

SequenceFile is a binary file provided by the Hadoop API that serializes data into a file as. This binary internally implements serialization and deserialization using Hadoop's standard Writable interface. It is compatible with MapFile in the Hadoop API. The SequenceFile in Hive is inherited from the SequenceFile in the Hadoop API, but it has an empty key and USES value to store the actual value, in order to avoid the sorting process of MR during the run of map. If you write Sequencefiles in Java apis and let Hive read them, make sure you use the Value field to hold the data, otherwise you'll need to customize the InputFormat class and OutputFormat class of the SequenceFile.

3.1 Configure Maven Enviroment

```
<dependencies>
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-client</artifactId>
    <version>2.9.2</version>
  </dependency>
</dependencies>
<build>
  <pluginManagement>    <plugins>
          <plugin>
        <artifactId>maven-clean-plugin</artifactId>
        <version>3.1.0</version>
      </plugin>
           <plugin>
        <artifactId>maven-resources-plugin</artifactId>
        <version>3.0.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
      </plugin>
      <plugin>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>2.22.1</version>
      </plugin>
      <plugin>
        <artifactId>maven-jar-plugin</artifactId>
        <version>3.0.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-install-plugin</artifactId>
        <version>2.9.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-deploy-plugin</artifactId>
        <version>2.8.2</version>
      </plugin>
           <plugin>
        <artifactId>maven-site-plugin</artifactId>
        <version>3.7.1</version>
      </plugin>
      <plugin>
```

```
         <artifactId>maven-project-info-reports-plugin</artifactId>
         <version>3.0.0</version>
       </plugin>
     </plugins>
   </pluginManagement>
</build>
```

3.2 Code--Input and Output File Path

```
public class SequenceFileTest {

//output
 static String PATH = "/home/hadoop01/output/test.seq";
 static SequenceFile.Writer writer = null;
 public static void main(String[] args) throws Exception{
  Configuration conf = new Configuration();
  conf.set("fs.default.name", "file:///");
  conf.set("mapred.job.tracker", "local");



//input
  String path = "/home/hadoop01/test/";
  URI uri = new URI(path);
  FileSystem fileSystem = FileSystem.get(uri, conf);
  writer = SequenceFile.createWriter(fileSystem, conf, new Path(PATH), Text.class,
BytesWritable.class);
  listFileAndWriteToSequenceFile(fileSystem,path);
  System.out.println("Seq 文件路径"+PATH+"----文件名："+"test.seq");
  org.apache.hadoop.io.IOUtils.closeStream(writer);
}
```

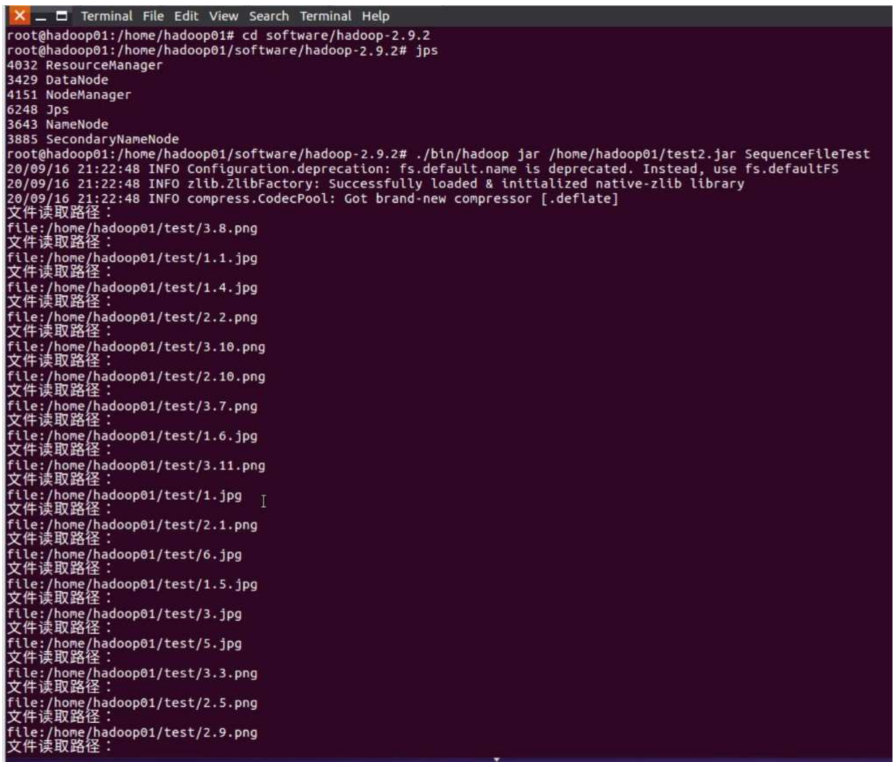
3.3 Code-- Pack File

```
//Iterate through the write to seq file
 public static void listFileAndWriteToSequenceFile(FileSystem fileSystem,String path)
throws Exception{
  final FileStatus[] listStatuses = fileSystem.listStatus(new Path(path));
  for (FileStatus fileStatus : listStatuses) {
   if(fileStatus.isFile()){
    Text fileText = new Text(fileStatus.getPath().toString());
    System.out.println("文件读取路径：");
    System.out.println(fileText.toString());
    FSDataInputStream in = fileSystem.open(new Path(fileText.toString()));
    byte[] buffer = IOUtils.toByteArray(in);
    in.read(buffer);

    BytesWritable value = new BytesWritable(buffer);
    writer.append(fileText, value);
   }

   if(fileStatus.isDirectory()){
    listFileAndWriteToSequenceFile(fileSystem,fileStatus.getPath().toString());
   }

  }

 }
```

3.4 Code-- Unpack File

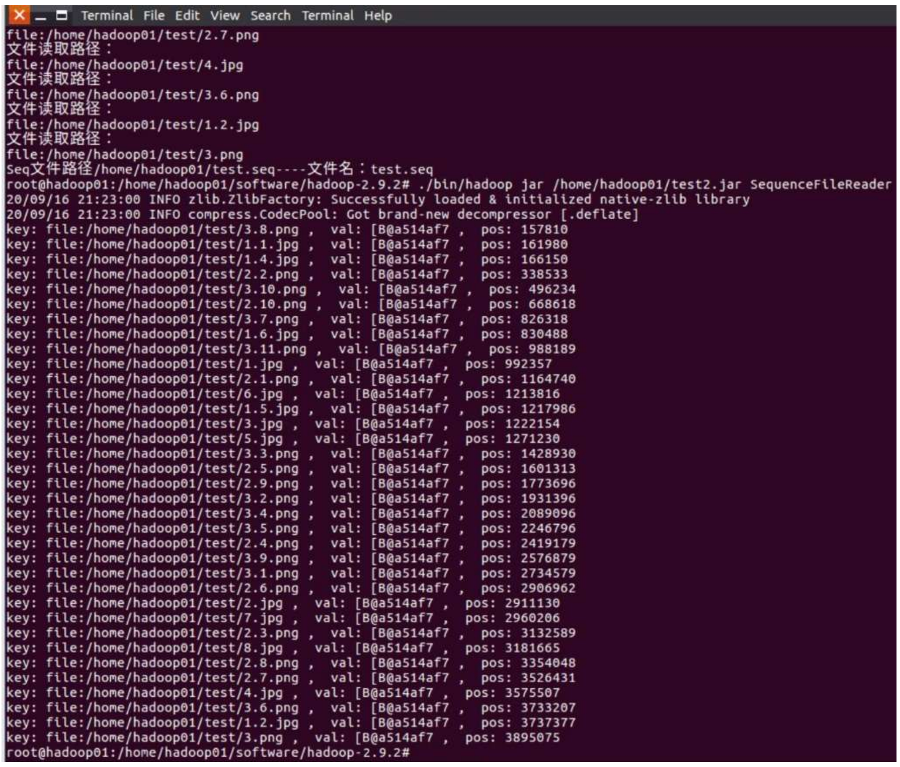
```
public class SequenceFileReader {
  public static void main(String[] args) throws IOException {
   Configuration conf = new Configuration();
      conf.set("fs.defaultFS", "file:///");

      FileSystem fs = FileSystem.get(conf);

      Path path = new Path("/home/hadoop01/test.seq");

      SequenceFile.Reader reader = new SequenceFile.Reader(fs, path, conf);

      Text key = new Text();
      BytesWritable value = new BytesWritable();

      while ((reader.next(key, value))) {
          long position = reader.getPosition();
          System.out.println("key: " + key.toString() + " , " + " val: " + value.get() +
" , " + " pos: " + position);
      }
  }
}
```

3.5 The Result in Console



```
root@hadoop01:/home/hadoop01# cd software/hadoop-2.9.2
root@hadoop01:/home/hadoop01/software/hadoop-2.9.2# jps
4032 ResourceManager
3429 DataNode
4151 NodeManager
6248 Jps
3643 NameNode
3885 SecondaryNameNode
root@hadoop01:/home/hadoop01/software/hadoop-2.9.2# ./bin/hadoop jar /home/hadoop01/test2.jar SequenceFileTest
20/09/16 21:22:48 INFO Configuration.deprecation: fs.default.name is deprecated. Instead, use fs.defaultFS
20/09/16 21:22:48 INFO zlib.ZlibFactory: Successfully loaded & initialized native-zlib library
20/09/16 21:22:48 INFO compress.CodecPool: Got brand-new compressor [.deflate]
文件读取路径：
file:/home/hadoop01/test/3.8.png
文件读取路径：
file:/home/hadoop01/test/1.1.jpg
文件读取路径：
file:/home/hadoop01/test/1.4.jpg
文件读取路径：
file:/home/hadoop01/test/2.2.png
文件读取路径：
file:/home/hadoop01/test/3.10.png
文件读取路径：
file:/home/hadoop01/test/2.10.png
文件读取路径：
file:/home/hadoop01/test/3.7.png
文件读取路径：
file:/home/hadoop01/test/1.6.jpg
文件读取路径：
file:/home/hadoop01/test/3.11.png
文件读取路径：
file:/home/hadoop01/test/1.jpg
文件读取路径：
file:/home/hadoop01/test/2.1.png
文件读取路径：
file:/home/hadoop01/test/6.jpg
文件读取路径：
file:/home/hadoop01/test/1.5.jpg
文件读取路径：
file:/home/hadoop01/test/3.jpg
文件读取路径：
file:/home/hadoop01/test/5.jpg
文件读取路径：
file:/home/hadoop01/test/3.3.png
文件读取路径：
file:/home/hadoop01/test/2.5.png
文件读取路径：
file:/home/hadoop01/test/2.9.png
文件读取路径：
```

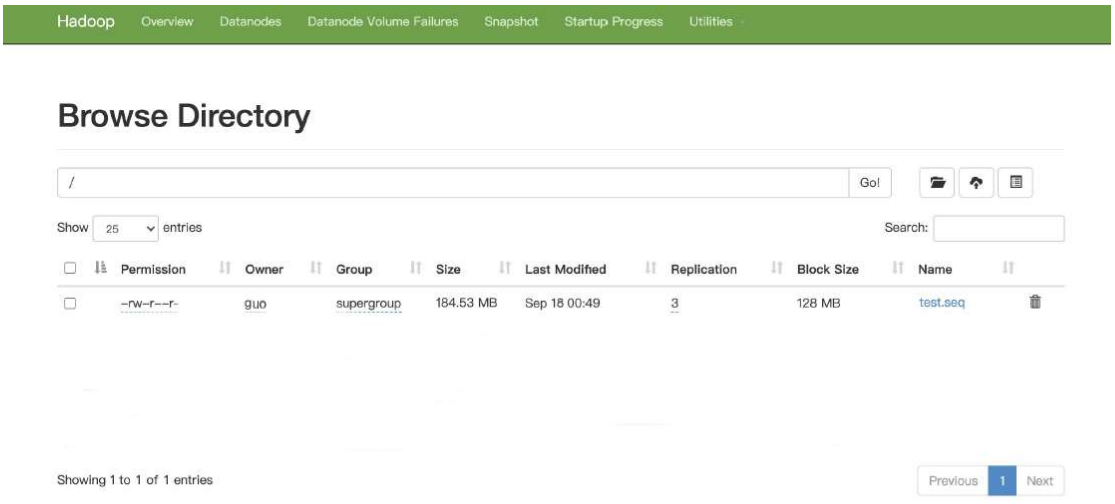
Figure 3-2 Output a sequenceFile containing all pictures



```
file:/home/hadoop01/test/2.7.png
文件读取路径：
file:/home/hadoop01/test/4.jpg
文件读取路径：
file:/home/hadoop01/test/3.6.png
文件读取路径：
file:/home/hadoop01/test/1.2.jpg
文件读取路径：
file:/home/hadoop01/test/3.png
Seq文件路径/home/hadoop01/test.seq----文件名：test.seq
root@hadoop01:/home/hadoop01/software/hadoop-2.9.2# ./bin/hadoop jar /home/hadoop01/test2.jar SequenceFileReader
20/09/16 21:23:00 INFO zlib.ZlibFactory: Successfully loaded & initialized native-zlib library
20/09/16 21:23:00 INFO compress.CodecPool: Got brand-new decompressor [.deflate]
key: file:/home/hadoop01/test/3.8.png ,  val: [B@a514af7 ,  pos: 157810
key: file:/home/hadoop01/test/1.1.jpg ,  val: [B@a514af7 ,  pos: 161980
key: file:/home/hadoop01/test/1.4.jpg ,  val: [B@a514af7 ,  pos: 166150
key: file:/home/hadoop01/test/2.2.png ,  val: [B@a514af7 ,  pos: 338533
key: file:/home/hadoop01/test/3.10.png ,  val: [B@a514af7 ,  pos: 496234
key: file:/home/hadoop01/test/2.10.png ,  val: [B@a514af7 ,  pos: 668618
key: file:/home/hadoop01/test/3.7.png ,  val: [B@a514af7 ,  pos: 826318
key: file:/home/hadoop01/test/1.6.jpg ,  val: [B@a514af7 ,  pos: 830488
key: file:/home/hadoop01/test/3.11.png ,  val: [B@a514af7 ,  pos: 988189
key: file:/home/hadoop01/test/1.jpg ,  val: [B@a514af7 ,  pos: 992357
key: file:/home/hadoop01/test/2.1.png ,  val: [B@a514af7 ,  pos: 1164740
key: file:/home/hadoop01/test/6.jpg ,  val: [B@a514af7 ,  pos: 1213816
key: file:/home/hadoop01/test/1.5.jpg ,  val: [B@a514af7 ,  pos: 1217986
key: file:/home/hadoop01/test/3.jpg ,  val: [B@a514af7 ,  pos: 1222154
key: file:/home/hadoop01/test/5.jpg ,  val: [B@a514af7 ,  pos: 1271230
key: file:/home/hadoop01/test/3.3.png ,  val: [B@a514af7 ,  pos: 1428930
key: file:/home/hadoop01/test/2.5.png ,  val: [B@a514af7 ,  pos: 1601313
key: file:/home/hadoop01/test/2.9.png ,  val: [B@a514af7 ,  pos: 1773696
key: file:/home/hadoop01/test/3.2.png ,  val: [B@a514af7 ,  pos: 1931396
key: file:/home/hadoop01/test/3.4.png ,  val: [B@a514af7 ,  pos: 2089096
key: file:/home/hadoop01/test/3.5.png ,  val: [B@a514af7 ,  pos: 2246796
key: file:/home/hadoop01/test/2.4.png ,  val: [B@a514af7 ,  pos: 2419179
key: file:/home/hadoop01/test/3.9.png ,  val: [B@a514af7 ,  pos: 2576879
key: file:/home/hadoop01/test/3.1.png ,  val: [B@a514af7 ,  pos: 2734579
key: file:/home/hadoop01/test/2.6.png ,  val: [B@a514af7 ,  pos: 2906962
key: file:/home/hadoop01/test/2.jpg ,  val: [B@a514af7 ,  pos: 2911130
key: file:/home/hadoop01/test/7.jpg ,  val: [B@a514af7 ,  pos: 2960206
key: file:/home/hadoop01/test/2.3.png ,  val: [B@a514af7 ,  pos: 3132589
key: file:/home/hadoop01/test/8.jpg ,  val: [B@a514af7 ,  pos: 3181665
key: file:/home/hadoop01/test/2.8.png ,  val: [B@a514af7 ,  pos: 3354048
key: file:/home/hadoop01/test/2.7.png ,  val: [B@a514af7 ,  pos: 3526431
key: file:/home/hadoop01/test/4.jpg ,  val: [B@a514af7 ,  pos: 3575507
key: file:/home/hadoop01/test/3.6.png ,  val: [B@a514af7 ,  pos: 3733207
key: file:/home/hadoop01/test/1.2.jpg ,  val: [B@a514af7 ,  pos: 3737377
key: file:/home/hadoop01/test/3.png ,  val: [B@a514af7 ,  pos: 3895075
root@hadoop01:/home/hadoop01/software/hadoop-2.9.2#
```

Figure 3-3 Output a directory containing all small images

3.6 The Result in Browse Directory



Browse Directory

/

Show 25 entries

Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r-	guo	supergroup	184.53 MB	Sep 18 00:49	3	128 MB	test.seq

Showing 1 to 1 of 1 entries

Figure 3-4 The result in browse directory