## HDFS JAVA API

## 获取文件系统

|  |
| --- |
| /\*\*  \* 获取文件系统  \*  \* @return FileSystem  \*/  public static FileSystem getFileSystem() {  //读取配置文件  Configuration conf = new Configuration();  // 文件系统  FileSystem fs = null;    String hdfsUri = HDFSUri;  if(StringUtils.isBlank(hdfsUri)){  // 返回默认文件系统 如果在 Hadoop集群下运行，使用此种方法可直接获取默认文件系统  try {  fs = FileSystem.get(conf);  } catch (IOException e) {  logger.error("", e);  }  }else{  // 返回指定的文件系统,如果在本地测试，需要使用此种方法获取文件系统  try {  URI uri = new URI(hdfsUri.trim());  fs = FileSystem.get(uri,conf);  } catch (URISyntaxException | IOException e) {  logger.error("", e);  }  }    return fs;  } |

## **创建文件目录**

|  |
| --- |
| /\*\*  \* 创建文件目录  \*  \* @param path  \*/  public static void mkdir(String path) {  try {  // 获取文件系统  FileSystem fs = getFileSystem();    String hdfsUri = HDFSUri;  if(StringUtils.isNotBlank(hdfsUri)){  path = hdfsUri + path;  }    // 创建目录  fs.mkdirs(new Path(path));    //释放资源  fs.close();  } catch (IllegalArgumentException | IOException e) {  logger.error("", e);  }  } |

## **删除文件或者文件目录**

复制代码

|  |
| --- |
| /\*\*  \* 删除文件或者文件目录  \*  \* @param path  \*/  public static void rmdir(String path) {  try {  // 返回FileSystem对象  FileSystem fs = getFileSystem();    String hdfsUri = HDFSUri;  if(StringUtils.isNotBlank(hdfsUri)){  path = hdfsUri + path;  }    // 删除文件或者文件目录 delete(Path f) 此方法已经弃用  fs.delete(new Path(path),true);    // 释放资源  fs.close();  } catch (IllegalArgumentException | IOException e) {  logger.error("", e);  }  } |

## **根据filter获取目录下的文件**

复制代码

|  |
| --- |
| /\*\*  \* 根据filter获取目录下的文件  \*  \* @param path  \* @param pathFilter  \* @return String[]  \*/  public static String[] ListFile(String path,PathFilter pathFilter) {  String[] files = new String[0];    try {  // 返回FileSystem对象  FileSystem fs = getFileSystem();    String hdfsUri = HDFSUri;  if(StringUtils.isNotBlank(hdfsUri)){  path = hdfsUri + path;  }    FileStatus[] status;  if(pathFilter != null){  // 根据filter列出目录内容  status = fs.listStatus(new Path(path),pathFilter);  }else{  // 列出目录内容  status = fs.listStatus(new Path(path));  }    // 获取目录下的所有文件路径  Path[] listedPaths = FileUtil.stat2Paths(status);  // 转换String[]  if (listedPaths != null && listedPaths.length > 0){  files = new String[listedPaths.length];  for (int i = 0; i < files.length; i++){  files[i] = listedPaths[i].toString();  }  }  // 释放资源  fs.close();  } catch (IllegalArgumentException | IOException e) {  logger.error("", e);  }    return files;  } |

## **文件上传至 HDFS**

|  |
| --- |
| /\*\*  \* 文件上传至 HDFS  \*  \* @param delSrc  \* @param overwrite  \* @param srcFile  \* @param destPath  \*/  public static void copyFileToHDFS(boolean delSrc, boolean overwrite,String srcFile,String destPath) {  // 源文件路径是Linux下的路径，如果在 windows 下测试，需要改写为Windows下的路径，比如D://hadoop/djt/weibo.txt  Path srcPath = new Path(srcFile);    // 目的路径  String hdfsUri = HDFSUri;  if(StringUtils.isNotBlank(hdfsUri)){  destPath = hdfsUri + destPath;  }  Path dstPath = new Path(destPath);    // 实现文件上传  try {  // 获取FileSystem对象  FileSystem fs = getFileSystem();  fs.copyFromLocalFile(srcPath, dstPath);  fs.copyFromLocalFile(delSrc,overwrite,srcPath, dstPath);  //释放资源  fs.close();  } catch (IOException e) {  logger.error("", e);  }  } |

## **从 HDFS 下载文件**

|  |
| --- |
| /\*\*  \* 从 HDFS 下载文件  \*  \* @param srcFile  \* @param destPath  \*/  public static void getFile(String srcFile,String destPath) {  // 源文件路径  String hdfsUri = HDFSUri;  if(StringUtils.isNotBlank(hdfsUri)){  srcFile = hdfsUri + srcFile;  }  Path srcPath = new Path(srcFile);    // 目的路径是Linux下的路径，如果在 windows 下测试，需要改写为Windows下的路径，比如D://hadoop/djt/  Path dstPath = new Path(destPath);    try {  // 获取FileSystem对象  FileSystem fs = getFileSystem();  // 下载hdfs上的文件  fs.copyToLocalFile(srcPath, dstPath);  // 释放资源  fs.close();  } catch (IOException e) {  logger.error("", e);  }  } |

## **获取 HDFS 集群节点信息**

|  |
| --- |
| /\*\*  \* 获取 HDFS 集群节点信息  \*  \* @return DatanodeInfo[]  \*/  public static DatanodeInfo[] getHDFSNodes() {  // 获取所有节点  DatanodeInfo[] dataNodeStats = new DatanodeInfo[0];    try {  // 返回FileSystem对象  FileSystem fs = getFileSystem();    // 获取分布式文件系统  DistributedFileSystem hdfs = (DistributedFileSystem)fs;    dataNodeStats = hdfs.getDataNodeStats();  } catch (IOException e) {  logger.error("", e);  }  return dataNodeStats;  } |

## **查找某个文件在 HDFS集群的位置**

|  |
| --- |
| /\*\*  \* 查找某个文件在 HDFS集群的位置  \*  \* @param filePath  \* @return BlockLocation[]  \*/  public static BlockLocation[] getFileBlockLocations(String filePath) {  // 文件路径  String hdfsUri = HDFSUri;  if(StringUtils.isNotBlank(hdfsUri)){  filePath = hdfsUri + filePath;  }  Path path = new Path(filePath);    // 文件块位置列表  BlockLocation[] blkLocations = new BlockLocation[0];  try {  // 返回FileSystem对象  FileSystem fs = getFileSystem();  // 获取文件目录  FileStatus filestatus = fs.getFileStatus(path);  //获取文件块位置列表  blkLocations = fs.getFileBlockLocations(filestatus, 0, filestatus.getLen());  } catch (IOException e) {  logger.error("", e);  }  return blkLocations;  } |

## **文件重命名**

|  |
| --- |
| /\*\*  \* 文件重命名  \*  \* @param srcPath  \* @param dstPath  \*/  public boolean rename(String srcPath, String dstPath){  boolean flag = false;  try {  // 返回FileSystem对象  FileSystem fs = getFileSystem();    String hdfsUri = HDFSUri;  if(StringUtils.isNotBlank(hdfsUri)){  srcPath = hdfsUri + srcPath;  dstPath = hdfsUri + dstPath;  }    flag = fs.rename(new Path(srcPath), new Path(dstPath));  } catch (IOException e) {  logger.error("{} rename to {} error.", srcPath, dstPath);  }    return flag;  } |

## **判断目录是否存在**

|  |
| --- |
| /\*\*  \* 判断目录是否存在  \*  \* @param srcPath  \* @param dstPath  \*/  public boolean existDir(String filePath, boolean create){  boolean flag = false;    if (StringUtils.isEmpty(filePath)){  return flag;  }    try{  Path path = new Path(filePath);  // FileSystem对象  FileSystem fs = getFileSystem();    if (create){  if (!fs.exists(path)){  fs.mkdirs(path);  }  }    if (fs.isDirectory(path)){  flag = true;  }  }catch (Exception e){  logger.error("", e);  }    return flag;  } |

## **读取HDFS数据**

|  |
| --- |
| **public** **class** FSDSample {    **public** **static** **void** main(String [] args) **throws** Exception{    Configuration conf = **new** Configuration();  FileSystem fs = FileSystem.*get*(conf);  FSDataInputStream fsin = fs.open(**new** Path("/hdfs\_test/11/11.txt"));  **byte**[] buff = **new** **byte**[128];  //long ln = 0;  **int** length = 0;  **while**((length = fsin.read(buff,0,128)) !=-1){  System.***out***.println(**new** String(buff,0,length-1));  System.***out***.println(fsin.getPos());      } |

## **向HDFS写入数据**

|  |
| --- |
| **public** **class** FSWriteSample {    **public** **static** **void** main(String[] args) **throws** Exception{    Configuration conf = **new** Configuration();  FileSystem fs = FileSystem.*get*(conf);  Path path = **new** Path("/hdfs\_test/11/22.txt");  FSDataOutputStream fsout = fs.append(path);  **byte**[] buff = "I am love you\n".getBytes();  fsout.write(buff);  IOUtils.*closeQuietly*(fsout);    }  } |