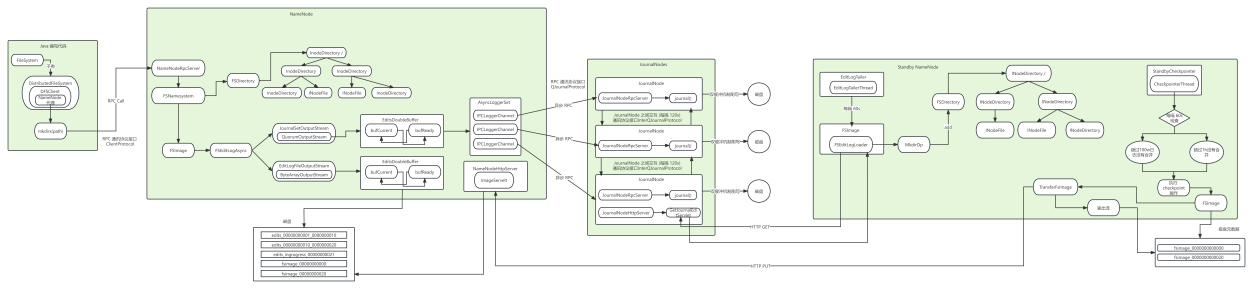
# hadoop-hdfs元数据管理源码分析

## 一 场景驱动案例

|  |
| --- |
| <dependency>  <groupId>org.apache.hadoop</groupId>  <artifactId>hadoop-client</artifactId>  <version>3.1.3</version>  </dependency> |

|  |
| --- |
| public class MkdirMain {  public static void main(String[] args) throws Exception {  System.setProperty("HADOOP\_USER\_NAME", "tanbs");  Configuration conf = new Configuration();  FileSystem fileSystem = FileSystem.get(conf);  fileSystem.mkdirs(new Path("/mkdir"));  }  } |

## 二 元数据管理源码分析



### 2.1 创建 FileSystem

|  |
| --- |
| FileSystem fileSystem = FileSystem.get(conf); |

|  |
| --- |
| /\*\*  \* Returns the configured FileSystem implementation.  \*  \* @param conf the configuration to use  \*/  public static FileSystem get(Configuration conf) throws IOException {  // 往下追  return get(  // 解析 key = fs.defaultFS 对应的值 比如 hdfs://mycluster  getDefaultUri(conf), conf  );  } |

|  |
| --- |
| /\*\*  \* Get a FileSystem for this URI's scheme and authority.  \* <ol>  \* <li>  \* If the configuration has the property  \* {@code "fs.$SCHEME.impl.disable.cache"} set to true,  \* a new instance will be created, initialized with the supplied URI and  \* configuration, then returned without being cached.  \* </li>  \* <li>  \* If the there is a cached FS instance matching the same URI, it will  \* be returned.  \* </li>  \* <li>  \* Otherwise: a new FS instance will be created, initialized with the  \* configuration and URI, cached and returned to the caller.  \* </li>  \* </ol>  \*  \* @throws IOException if the FileSystem cannot be instantiated.  \*/  public static FileSystem get(URI uri, Configuration conf) throws IOException {  // hdfs  String scheme = uri.getScheme();  // mycluster  String authority = uri.getAuthority();  if (scheme == null && authority == null) { // use default FS  return get(conf);  }  if (scheme != null && authority == null) { // no authority  URI defaultUri = getDefaultUri(conf);  if (scheme.equals(defaultUri.getScheme()) // if scheme matches default  && defaultUri.getAuthority() != null) { // & default has authority  return get(defaultUri, conf); // return default  }  }  // fs.hdfs.impl.disable.cache  String disableCacheName = String.format("fs.%s.impl.disable.cache", scheme);  if (conf.getBoolean(disableCacheName, false)) {  LOGGER.debug("Bypassing cache to create filesystem {}", uri);  return createFileSystem(uri, conf);  }  // 往下追 (最终返回 DistributedFileSystem 对象)  return CACHE.get(uri, conf);  } |

|  |
| --- |
| FileSystem get(URI uri, Configuration conf) throws IOException {  Key key = new Key(uri, conf);  // 往下追  return getInternal(uri, conf, key);  } |

|  |
| --- |
| /\*\*  \* Get the FS instance if the key maps to an instance, creating and  \* initializing the FS if it is not found.  \* If this is the first entry in the map and the JVM is not shutting down,  \* this registers a shutdown hook to close filesystems, and adds this  \* FS to the {@code toAutoClose} set if {@code "fs.automatic.close"}  \* is set in the configuration (default: true).  \*  \* @param uri filesystem URI  \* @param conf configuration  \* @param key key to store/retrieve this FileSystem in the cache  \* @return a cached or newly instantiated FileSystem.  \* @throws IOException  \*/  private FileSystem getInternal(URI uri, Configuration conf, Key key)  throws IOException {  FileSystem fs;  synchronized (this) {  fs = map.get(key);  }  if (fs != null) {  return fs;  }  // 创建 DistributedFileSystem  fs = createFileSystem(uri, conf);  synchronized (this) { // refetch the lock again  FileSystem oldfs = map.get(key);  if (oldfs != null) { // a file system is created while lock is releasing  fs.close(); // close the new file system  return oldfs; // return the old file system  }  // now insert the new file system into the map  if (map.isEmpty()  && !ShutdownHookManager.get().isShutdownInProgress()) {  ShutdownHookManager.get().addShutdownHook(clientFinalizer, SHUTDOWN\_HOOK\_PRIORITY);  }  fs.key = key;  map.put(key, fs);  if (conf.getBoolean(  FS\_AUTOMATIC\_CLOSE\_KEY, FS\_AUTOMATIC\_CLOSE\_DEFAULT)) {  toAutoClose.add(key);  }  return fs;  }  } |

|  |
| --- |
| /\*\*  \* Create and initialize a new instance of a FileSystem.  \*  \* @param uri URI containing the FS schema and FS details  \* @param conf configuration to use to look for the FS instance declaration  \* and to pass to the {@link FileSystem#initialize(URI, Configuration)}.  \* @return the initialized filesystem.  \* @throws IOException problems loading or initializing the FileSystem  \*/  private static FileSystem createFileSystem(URI uri, Configuration conf)  throws IOException {  Tracer tracer = FsTracer.get(conf);  try (TraceScope scope = tracer.newScope("FileSystem#createFileSystem")) {  scope.addKVAnnotation("scheme", uri.getScheme());  // 返回 DistributedFileSystem.class  Class<?> clazz = getFileSystemClass(uri.getScheme(), conf);  // 创建 DistributedFileSystem  FileSystem fs = (FileSystem) ReflectionUtils.newInstance(clazz, conf);  // 调用 DistributedFileSystem.initialize()  fs.initialize(uri, conf);  return fs;  }  } |

#### 2.1.1 创建 DistributedFileSystem

|  |
| --- |
| /\*\* Create an object for the given class and initialize it from conf  \*  \* @param theClass class of which an object is created  \* @param conf Configuration  \* @return a new object  \*/  @SuppressWarnings("unchecked")  public static <T> T newInstance(Class<T> theClass, Configuration conf) {  T result;  try {  Constructor<T> meth = (Constructor<T>) CONSTRUCTOR\_CACHE.get(theClass);  if (meth == null) {  meth = theClass.getDeclaredConstructor(EMPTY\_ARRAY);  meth.setAccessible(true);  CONSTRUCTOR\_CACHE.put(theClass, meth);  }  // 调用 DistributedFileSystem 无参构造  result = meth.newInstance();  } catch (Exception e) {  throw new RuntimeException(e);  }  // null  setConf(result, conf);  // 返回 DistributedFileSystem 对象  return result;  } |

|  |
| --- |
| public DistributedFileSystem() {  } |

#### 2.1.2 初始化 DistributedFileSystem

|  |
| --- |
| @Override  public void initialize(URI uri, Configuration conf) throws IOException {  // 调用父类  super.initialize(uri, conf);  // 设置 配置对象  setConf(conf);  // mycluster  String host = uri.getHost();  if (host == null) {  throw new IOException("Incomplete HDFS URI, no host: "+ uri);  }  // 创建 DFSClient (底层创建 NameNode 代理对象 通讯协议接口为 ClientProtocol)  this.dfs = new DFSClient(uri, conf, statistics);  // hdfs://mycluster  this.uri = URI.create(uri.getScheme()+"://"+uri.getAuthority());  this.workingDir = getHomeDirectory();  storageStatistics = (DFSOpsCountStatistics) GlobalStorageStatistics.INSTANCE  .put(DFSOpsCountStatistics.NAME,  new StorageStatisticsProvider() {  @Override  public StorageStatistics provide() {  return new DFSOpsCountStatistics();  }  });  } |

##### 2.1.2.1 创建 DFSClient

|  |
| --- |
| /\*\*  \* Same as this(nameNodeUri, null, conf, stats);  \* @see #DFSClient(URI, ClientProtocol, Configuration, FileSystem.Statistics)  \*/  public DFSClient(URI nameNodeUri, Configuration conf,  FileSystem.Statistics stats) throws IOException {  // 往下追  this(nameNodeUri, null, conf, stats);  } |

|  |
| --- |
| /\*\*  \* Create a new DFSClient connected to the given nameNodeUri or rpcNamenode.  \* If HA is enabled and a positive value is set for  \* {@link HdfsClientConfigKeys#DFS\_CLIENT\_TEST\_DROP\_NAMENODE\_RESPONSE\_NUM\_KEY}  \* in the configuration, the DFSClient will use  \* {@link LossyRetryInvocationHandler} as its RetryInvocationHandler.  \* Otherwise one of nameNodeUri or rpcNamenode must be null.  \*/  @VisibleForTesting  public DFSClient(URI nameNodeUri, ClientProtocol rpcNamenode,  Configuration conf, FileSystem.Statistics stats) throws IOException {  // Copy only the required DFSClient configuration  this.tracer = FsTracer.get(conf);  // 创建 DfsClientConf (里面封装了很多参数)  this.dfsClientConf = new DfsClientConf(conf);  this.conf = conf;  this.stats = stats;  // 返回 StandardSocketFactory 对象  this.socketFactory = NetUtils.getSocketFactory(conf, ClientProtocol.class);  this.dtpReplaceDatanodeOnFailure = ReplaceDatanodeOnFailure.get(conf);  // 512  this.smallBufferSize = DFSUtilClient.getSmallBufferSize(conf);  // 0  this.dtpReplaceDatanodeOnFailureReplication = (short) conf  .getInt(HdfsClientConfigKeys.BlockWrite.ReplaceDatanodeOnFailure.  MIN\_REPLICATION,  HdfsClientConfigKeys.BlockWrite.ReplaceDatanodeOnFailure.  MIN\_REPLICATION\_DEFAULT);  if (LOG.isDebugEnabled()) {  LOG.debug(  "Sets " + HdfsClientConfigKeys.BlockWrite.ReplaceDatanodeOnFailure.  MIN\_REPLICATION + " to "  + dtpReplaceDatanodeOnFailureReplication);  }  this.ugi = UserGroupInformation.getCurrentUser();  // hdfs://mycluster  this.namenodeUri = nameNodeUri;  // 客户端 ID  this.clientName = "DFSClient\_" + dfsClientConf.getTaskId() + "\_" +  ThreadLocalRandom.current().nextInt() + "\_" +  Thread.currentThread().getId();  // 0  int numResponseToDrop = conf.getInt(  DFS\_CLIENT\_TEST\_DROP\_NAMENODE\_RESPONSE\_NUM\_KEY,  DFS\_CLIENT\_TEST\_DROP\_NAMENODE\_RESPONSE\_NUM\_DEFAULT);  ProxyAndInfo<ClientProtocol> proxyInfo = null;  AtomicBoolean nnFallbackToSimpleAuth = new AtomicBoolean(false);  if (numResponseToDrop > 0) {  // This case is used for testing.  LOG.warn(DFS\_CLIENT\_TEST\_DROP\_NAMENODE\_RESPONSE\_NUM\_KEY  + " is set to " + numResponseToDrop  + ", this hacked client will proactively drop responses");  proxyInfo = NameNodeProxiesClient.createProxyWithLossyRetryHandler(conf,  nameNodeUri, ClientProtocol.class, numResponseToDrop,  nnFallbackToSimpleAuth);  }  if (proxyInfo != null) {  this.dtService = proxyInfo.getDelegationTokenService();  this.namenode = proxyInfo.getProxy();  } else if (rpcNamenode != null) {  // This case is used for testing.  Preconditions.checkArgument(nameNodeUri == null);  this.namenode = rpcNamenode;  dtService = null;  } else {  // 默认来到这  Preconditions.checkArgument(nameNodeUri != null,  "null URI");  // 获取 NameNode 代理 (通讯协议接口为 ClientProtocol) 返回 ProxyAndInfo  proxyInfo = NameNodeProxiesClient.createProxyWithClientProtocol(  conf,  nameNodeUri,  nnFallbackToSimpleAuth);  this.dtService = proxyInfo.getDelegationTokenService();  // 真正 NameNode 代理对象  this.namenode = proxyInfo.getProxy();  }  String localInterfaces[] =  conf.getTrimmedStrings(DFS\_CLIENT\_LOCAL\_INTERFACES);  localInterfaceAddrs = getLocalInterfaceAddrs(localInterfaces);  if (LOG.isDebugEnabled() && 0 != localInterfaces.length) {  LOG.debug("Using local interfaces [" +  Joiner.on(',').join(localInterfaces) + "] with addresses [" +  Joiner.on(',').join(localInterfaceAddrs) + "]");  }  Boolean readDropBehind =  (conf.get(DFS\_CLIENT\_CACHE\_DROP\_BEHIND\_READS) == null) ?  null : conf.getBoolean(DFS\_CLIENT\_CACHE\_DROP\_BEHIND\_READS, false);  Long readahead = (conf.get(DFS\_CLIENT\_CACHE\_READAHEAD) == null) ?  null : conf.getLong(DFS\_CLIENT\_CACHE\_READAHEAD, 0);  this.serverDefaultsValidityPeriod =  conf.getLong(DFS\_CLIENT\_SERVER\_DEFAULTS\_VALIDITY\_PERIOD\_MS\_KEY,  DFS\_CLIENT\_SERVER\_DEFAULTS\_VALIDITY\_PERIOD\_MS\_DEFAULT);  Boolean writeDropBehind =  (conf.get(DFS\_CLIENT\_CACHE\_DROP\_BEHIND\_WRITES) == null) ?  null : conf.getBoolean(DFS\_CLIENT\_CACHE\_DROP\_BEHIND\_WRITES, false);  this.defaultReadCachingStrategy =  new CachingStrategy(readDropBehind, readahead);  this.defaultWriteCachingStrategy =  new CachingStrategy(writeDropBehind, readahead);  this.clientContext = ClientContext.get(  conf.get(DFS\_CLIENT\_CONTEXT, DFS\_CLIENT\_CONTEXT\_DEFAULT),  dfsClientConf, conf);  if (dfsClientConf.getHedgedReadThreadpoolSize() > 0) {  this.initThreadsNumForHedgedReads(dfsClientConf.  getHedgedReadThreadpoolSize());  }  this.initThreadsNumForStripedReads(dfsClientConf.  getStripedReadThreadpoolSize());  this.saslClient = new SaslDataTransferClient(  conf, DataTransferSaslUtil.getSaslPropertiesResolver(conf),  TrustedChannelResolver.getInstance(conf), nnFallbackToSimpleAuth);  } |

###### 2.1.2.1.1 创建 NameNode 代理 (通讯协议接口 ClientProtocol)

|  |
| --- |
| /\*\*  \* Creates the namenode proxy with the ClientProtocol. This will handle  \* creation of either HA- or non-HA-enabled proxy objects, depending upon  \* if the provided URI is a configured logical URI.  \*  \* @param conf the configuration containing the required IPC  \* properties, client failover configurations, etc.  \* @param nameNodeUri the URI pointing either to a specific NameNode  \* or to a logical nameservice.  \* @param fallbackToSimpleAuth set to true or false during calls to indicate  \* if a secure client falls back to simple auth  \* @return an object containing both the proxy and the associated  \* delegation token service it corresponds to  \* @throws IOException if there is an error creating the proxy  \* @see {@link NameNodeProxies#createProxy(Configuration, URI, Class)}.  \*/  public static ProxyAndInfo<ClientProtocol> createProxyWithClientProtocol(  Configuration conf, URI nameNodeUri, AtomicBoolean fallbackToSimpleAuth)  throws IOException {  // 默认返回 null  AbstractNNFailoverProxyProvider<ClientProtocol> failoverProxyProvider =  createFailoverProxyProvider(conf,  nameNodeUri,  ClientProtocol.class,  true,  fallbackToSimpleAuth);  if (failoverProxyProvider == null) {  // 获取 NameNode 地址 比如 hj101:8020  InetSocketAddress nnAddr = DFSUtilClient.getNNAddress(nameNodeUri);  Text dtService = SecurityUtil.buildTokenService(nnAddr);  // 创建 ClientProtocol 代理对象 (代理对象底层连接 NameNode Rpc 代理)  ClientProtocol proxy = createNonHAProxyWithClientProtocol(  nnAddr,  conf,  UserGroupInformation.getCurrentUser(),  true, fallbackToSimpleAuth);  // 封装 ClientProtocol 代理对象 为 ProxyAndInfo  return new ProxyAndInfo<>(proxy, dtService, nnAddr);  } else {  return createHAProxy(conf, nameNodeUri, ClientProtocol.class,  failoverProxyProvider);  }  } |

|  |
| --- |
| public static ClientProtocol createNonHAProxyWithClientProtocol(  InetSocketAddress address, Configuration conf, UserGroupInformation ugi,  boolean withRetries, AtomicBoolean fallbackToSimpleAuth)  throws IOException {  // 往下追  return createProxyWithAlignmentContext(address,  conf, ugi,  withRetries,  fallbackToSimpleAuth,  null);  } |

|  |
| --- |
| public static ClientProtocol createProxyWithAlignmentContext(  InetSocketAddress address, Configuration conf, UserGroupInformation ugi,  boolean withRetries, AtomicBoolean fallbackToSimpleAuth,  AlignmentContext alignmentContext)  throws IOException {  // 设置通信协议接口 (ClientProtocol)  RPC.setProtocolEngine(conf, ClientNamenodeProtocolPB.class,  ProtobufRpcEngine.class);  final RetryPolicy defaultPolicy =  RetryUtils.getDefaultRetryPolicy(  conf,  HdfsClientConfigKeys.Retry.POLICY\_ENABLED\_KEY,  HdfsClientConfigKeys.Retry.POLICY\_ENABLED\_DEFAULT,  HdfsClientConfigKeys.Retry.POLICY\_SPEC\_KEY,  HdfsClientConfigKeys.Retry.POLICY\_SPEC\_DEFAULT,  SafeModeException.class.getName());  // 1  final long version = RPC.getProtocolVersion(ClientNamenodeProtocolPB.class);  // ClientNamenodeProtocolPB 代理  ClientNamenodeProtocolPB proxy = RPC.getProtocolProxy(  ClientNamenodeProtocolPB.class,  version, address, ugi, conf,  NetUtils.getDefaultSocketFactory(conf),  org.apache.hadoop.ipc.Client.getTimeout(conf), defaultPolicy,  fallbackToSimpleAuth, alignmentContext).getProxy();  // true  if (withRetries) { // create the proxy with retries  Map<String, RetryPolicy> methodNameToPolicyMap = new HashMap<>();  ClientProtocol translatorProxy =  new ClientNamenodeProtocolTranslatorPB(proxy);  // hadoop-rpc 标配操作  return (ClientProtocol) RetryProxy.create(  ClientProtocol.class,  new DefaultFailoverProxyProvider<>(ClientProtocol.class,  translatorProxy),  methodNameToPolicyMap,  defaultPolicy);  } else {  return new ClientNamenodeProtocolTranslatorPB(proxy);  }  } |

### 2.2 执行创建目录

|  |
| --- |
| /\*\*  \* Call {@link #mkdirs(Path, FsPermission)} with default permission.  \*  \* @param f path  \* @return true if the directory was created  \* @throws IOException IO failure  \*/  public boolean mkdirs(Path f) throws IOException {  // 往下追  return mkdirs(f, FsPermission.getDirDefault());  } |

|  |
| --- |
| /\*\*  \* Make the given file and all non-existent parents into  \* directories. Has roughly the semantics of Unix @{code mkdir -p}.  \* Existence of the directory hierarchy is not an error.  \*  \* @param f path to create  \* @param permission to apply to f  \* @throws IOException IO failure  \*/  // 调用 DistributedFileSystem  public abstract boolean mkdirs(Path f, FsPermission permission  ) throws IOException; |

|  |
| --- |
| /\*\*  \* Create a directory and its parent directories.  \*  \* See {@link FsPermission#applyUMask(FsPermission)} for details of how  \* the permission is applied.  \*  \* @param f The path to create  \* @param permission The permission. See FsPermission#applyUMask for  \* details about how this is used to calculate the  \* effective permission.  \*/  @Override  public boolean mkdirs(Path f, FsPermission permission) throws IOException {  // 往下追  return mkdirsInternal(f, permission, true);  } |

|  |
| --- |
| private boolean mkdirsInternal(Path f, final FsPermission permission,  final boolean createParent) throws IOException {  statistics.incrementWriteOps(1);  storageStatistics.incrementOpCounter(OpType.MKDIRS);  Path absF = fixRelativePart(f);  return new FileSystemLinkResolver<Boolean>() {  @Override  public Boolean doCall(final Path p) throws IOException {  // 往下追  return dfs.mkdirs(getPathName(p), permission, createParent);  }  @Override  public Boolean next(final FileSystem fs, final Path p)  throws IOException {  // FileSystem doesn't have a non-recursive mkdir() method  // Best we can do is error out  if (!createParent) {  throw new IOException("FileSystem does not support non-recursive"  + "mkdir");  }  return fs.mkdirs(p, permission);  }  // 闭包结构 最终调用 doCall()  }.resolve(this, absF);  } |

|  |
| --- |
| /\*\*  \* Create a directory (or hierarchy of directories) with the given  \* name and permission.  \*  \* @param src The path of the directory being created  \* @param permission The permission of the directory being created.  \* If permission == null, use {@link FsPermission#getDirDefault()}.  \* @param createParent create missing parent directory if true  \* @return True if the operation success.  \* @see ClientProtocol#mkdirs(String, FsPermission, boolean)  \*/  public boolean mkdirs(String src, FsPermission permission,  boolean createParent) throws IOException {  // 权限相关  final FsPermission masked = applyUMaskDir(permission);  // 往下追  return primitiveMkdir(src, masked, createParent);  } |

|  |
| --- |
| /\*\*  \* Same {{@link #mkdirs(String, FsPermission, boolean)} except  \* that the permissions has already been masked against umask.  \*/  public boolean primitiveMkdir(String src, FsPermission absPermission,  boolean createParent) throws IOException {  checkOpen();  if (absPermission == null) {  absPermission = applyUMaskDir(null);  }  LOG.debug("{}: masked={}", src, absPermission);  try (TraceScope ignored = tracer.newScope("mkdir")) {  // 调用 NameNodeRpcServer.mkdirs()  return namenode.mkdirs(src, absPermission, createParent);  } catch (RemoteException re) {  throw re.unwrapRemoteException(AccessControlException.class,  InvalidPathException.class,  FileAlreadyExistsException.class,  FileNotFoundException.class,  ParentNotDirectoryException.class,  SafeModeException.class,  NSQuotaExceededException.class,  DSQuotaExceededException.class,  QuotaByStorageTypeExceededException.class,  UnresolvedPathException.class,  SnapshotAccessControlException.class);  }  } |

### 2.3 调用 NameNodeRpcServer.mkdirs() 执行服务端创建目录

|  |
| --- |
| @Override // ClientProtocol  public boolean mkdirs(String src, FsPermission masked, boolean createParent)  throws IOException {  checkNNStartup();  if (stateChangeLog.isDebugEnabled()) {  stateChangeLog.debug("\*DIR\* NameNode.mkdirs: " + src);  }  if (!checkPathLength(src)) {  throw new IOException("mkdirs: Pathname too long. Limit "  + MAX\_PATH\_LENGTH + " characters, " + MAX\_PATH\_DEPTH + " levels.");  }  // 往下追  return namesystem.mkdirs(src,  new PermissionStatus(getRemoteUser().getShortUserName(),  null, masked), createParent);  } |

|  |
| --- |
| /\*\*  \* Create all the necessary directories  \*/  boolean mkdirs(String src, PermissionStatus permissions,  boolean createParent) throws IOException {  final String operationName = "mkdirs";  FileStatus auditStat = null;  checkOperation(OperationCategory.WRITE);  final FSPermissionChecker pc = getPermissionChecker();  writeLock();  try {  checkOperation(OperationCategory.WRITE);  checkNameNodeSafeMode("Cannot create directory " + src);  // 创建目录  auditStat = FSDirMkdirOp.mkdirs(this, pc, src, permissions,  createParent);  } catch (AccessControlException e) {  logAuditEvent(false, operationName, src);  throw e;  } finally {  writeUnlock(operationName);  }  // 同步 EditLog (当前 hadoop-3.1.3 为异步 故忽略)  getEditLog().logSync();  // 审计日志相关  logAuditEvent(true, operationName, src, null, auditStat);  return true;  } |

#### 2.3.1 创建目录

|  |
| --- |
| static FileStatus mkdirs(FSNamesystem fsn, FSPermissionChecker pc, String src,  PermissionStatus permissions, boolean createParent) throws IOException {  // 获取 FSNamesystem  FSDirectory fsd = fsn.getFSDirectory();  if (NameNode.stateChangeLog.isDebugEnabled()) {  NameNode.stateChangeLog.debug("DIR\* NameSystem.mkdirs: " + src);  }  fsd.writeLock();  try {  // 检查创建目录不包含非法字符并解析其路径  INodesInPath iip = fsd.resolvePath(pc, src, DirOp.CREATE);  // 获取目标目录的上一 INode  final INode lastINode = iip.getLastINode();  if (lastINode != null && lastINode.isFile()) {  throw new FileAlreadyExistsException("Path is not a directory: " + src);  }  if (lastINode == null) {  if (fsd.isPermissionEnabled()) {  fsd.checkAncestorAccess(pc, iip, FsAction.WRITE);  }  if (!createParent) {  fsd.verifyParentDir(iip);  }  // validate that we have enough inodes. This is, at best, a  // heuristic because the mkdirs() operation might need to  // create multiple inodes.  fsn.checkFsObjectLimit();  // Ensure that the user can traversal the path by adding implicit  // u+wx permission to all ancestor directories.  // 返回父目录  INodesInPath existing =  createParentDirectories(fsd, iip, permissions, false);  if (existing != null) {  // 创建目录 返回该目录信息  existing = createSingleDirectory(  fsd, existing, iip.getLastLocalName(), permissions);  }  if (existing == null) {  throw new IOException("Failed to create directory: " + src);  }  iip = existing;  }  return fsd.getAuditFileInfo(iip);  } finally {  fsd.writeUnlock();  }  } |

|  |
| --- |
| private static INodesInPath createSingleDirectory(FSDirectory fsd,  INodesInPath existing, byte[] localName, PermissionStatus perm)  throws IOException {  assert fsd.hasWriteLock();  // 通过父目录创建子目录  existing = unprotectedMkdir(fsd, fsd.allocateNewInodeId(), existing,  localName, perm, null, now());  if (existing == null) {  return null;  }  final INode newNode = existing.getLastINode();  // Directory creation also count towards FilesCreated  // to match count of FilesDeleted metric.  NameNode.getNameNodeMetrics().incrFilesCreated();  // 获取已经创建目录 path  String cur = existing.getPath();  // 往 EditLog 添加一条记录  fsd.getEditLog().logMkDir(cur, newNode);  if (NameNode.stateChangeLog.isDebugEnabled()) {  NameNode.stateChangeLog.debug("mkdirs: created directory " + cur);  }  // 返回创建目录信息  return existing;  } |

##### 2.3.1.1 通过父目录创建子目录

|  |
| --- |
| /\*\*  \* create a directory at path specified by parent  \*/  private static INodesInPath unprotectedMkdir(FSDirectory fsd, long inodeId,  INodesInPath parent, byte[] name, PermissionStatus permission,  List<AclEntry> aclEntries, long timestamp)  throws QuotaExceededException, AclException, FileAlreadyExistsException {  assert fsd.hasWriteLock();  assert parent.getLastINode() != null;  if (!parent.getLastINode().isDirectory()) {  throw new FileAlreadyExistsException("Parent path is not a directory: " +  parent.getPath() + " " + DFSUtil.bytes2String(name));  }  // 创建 INodeDirectory  final INodeDirectory dir = new INodeDirectory(inodeId, name, permission,  timestamp);  // 往父目录添加子目录 返回子目录信息  INodesInPath iip =  fsd.addLastINode(parent, dir, permission.getPermission(), true);  if (iip != null && aclEntries != null) {  AclStorage.updateINodeAcl(dir, aclEntries, Snapshot.CURRENT\_STATE\_ID);  }  return iip;  } |

|  |
| --- |
| /\*\*  \* Add a child to the end of the path specified by INodesInPath.  \* @param existing the INodesInPath containing all the ancestral INodes  \* @param inode the new INode to add  \* @param modes create modes  \* @param checkQuota whether to check quota  \* @return an INodesInPath instance containing the new INode  \*/  @VisibleForTesting  public INodesInPath addLastINode(INodesInPath existing, INode inode,  FsPermission modes, boolean checkQuota) throws QuotaExceededException {  assert existing.getLastINode() != null &&  existing.getLastINode().isDirectory();  final int pos = existing.length();  // Disallow creation of /.reserved. This may be created when loading  // editlog/fsimage during upgrade since /.reserved was a valid name in older  // release. This may also be called when a user tries to create a file  // or directory /.reserved.  if (pos == 1 && existing.getINode(0) == rootDir && isReservedName(inode)) {  throw new HadoopIllegalArgumentException(  "File name \"" + inode.getLocalName() + "\" is reserved and cannot "  + "be created. If this is during upgrade change the name of the "  + "existing file or directory to another name before upgrading "  + "to the new release.");  }  // 父目录  final INodeDirectory parent = existing.getINode(pos - 1).asDirectory();  // The filesystem limits are not really quotas, so this check may appear  // odd. It's because a rename operation deletes the src, tries to add  // to the dest, if that fails, re-adds the src from whence it came.  // The rename code disables the quota when it's restoring to the  // original location because a quota violation would cause the the item  // to go "poof". The fs limits must be bypassed for the same reason.  if (checkQuota) {  final String parentPath = existing.getPath();  verifyMaxComponentLength(inode.getLocalNameBytes(), parentPath);  verifyMaxDirItems(parent, parentPath);  }  // always verify inode name  verifyINodeName(inode.getLocalNameBytes());  final QuotaCounts counts = inode  .computeQuotaUsage(getBlockStoragePolicySuite(),  parent.getStoragePolicyID(), false, Snapshot.CURRENT\_STATE\_ID);  updateCount(existing, pos, counts, checkQuota);  boolean isRename = (inode.getParent() != null);  // 往父目录添加子目录  final boolean added = parent.addChild(inode, true,  existing.getLatestSnapshotId());  if (!added) {  updateCountNoQuotaCheck(existing, pos, counts.negation());  return null;  } else {  if (!isRename) {  copyINodeDefaultAcl(inode, modes);  }  // 缓存 Inode  addToInodeMap(inode);  }  // 返回 Inode 信息  return INodesInPath.append(existing, inode, inode.getLocalNameBytes());  } |

|  |
| --- |
| /\*\*  \* Add a child inode to the directory.  \*  \* @param node INode to insert  \* @param setModTime set modification time for the parent node  \* not needed when replaying the addition and  \* the parent already has the proper mod time  \* @return false if the child with this name already exists;  \* otherwise, return true;  \*/  public boolean addChild(INode node, final boolean setModTime,  final int latestSnapshotId) {  final int low = searchChildren(node.getLocalNameBytes());  if (low >= 0) {  return false;  }  if (isInLatestSnapshot(latestSnapshotId)) {  // create snapshot feature if necessary  DirectoryWithSnapshotFeature sf = this.getDirectoryWithSnapshotFeature();  if (sf == null) {  sf = this.addSnapshotFeature(null);  }  return sf.addChild(this, node, setModTime, latestSnapshotId);  }  // 添加  addChild(node, low);  if (setModTime) {  // update modification time of the parent directory  updateModificationTime(node.getModificationTime(), latestSnapshotId);  }  return true;  } |

|  |
| --- |
| /\*\*  \* Add the node to the children list at the given insertion point.  \* The basic add method which actually calls children.add(..).  \*/  private void addChild(final INode node, final int insertionPoint) {  if (children == null) {  children = new ArrayList<>(DEFAULT\_FILES\_PER\_DIRECTORY);  }  node.setParent(this);  // 添加  children.add(-insertionPoint - 1, node);  if (node.getGroupName() == null) {  node.setGroup(getGroupName());  }  } |

##### 2.3.1.2 往 EditLog 添加一条记录

|  |
| --- |
| /\*\*  \* Add create directory record to edit log  \*/  public void logMkDir(String path, INode newNode) {  PermissionStatus permissions = newNode.getPermissionStatus();  // Mkdir 操作符记录封装  MkdirOp op = MkdirOp.getInstance(cache.get())  .setInodeId(newNode.getId())  .setPath(path)  .setTimestamp(newNode.getModificationTime())  .setPermissionStatus(permissions);  AclFeature f = newNode.getAclFeature();  if (f != null) {  op.setAclEntries(AclStorage.readINodeLogicalAcl(newNode));  }  XAttrFeature x = newNode.getXAttrFeature();  if (x != null) {  op.setXAttrs(x.getXAttrs());  }  // 执行添加 (调用 FSEditLogAsync.logEdit())  logEdit(op);  } |

|  |
| --- |
| @Override  void logEdit(final FSEditLogOp op) {  // 获取 Edit 对象 返回 RpcEdit  Edit edit = getEditInstance(op);  // 当前线程内部变量设置 RpcEdit  THREAD\_EDIT.set(edit);  // 将 RpcEdit 添加到队列  enqueueEdit(edit);  } |

###### 2.3.1.2.1 创建 RpcEdit

|  |
| --- |
| private Edit getEditInstance(FSEditLogOp op) {  final Edit edit;  final Server.Call rpcCall = Server.getCurCall().get();  // only rpc calls not explicitly sync'ed on the log will be async.  if (rpcCall != null && !Thread.holdsLock(this)) {  // 创建 RpcEdit  edit = new RpcEdit(this, op, rpcCall);  } else {  edit = new SyncEdit(this, op);  }  return edit;  } |

###### 2.3.1.2.2 将 RpcEdit 添加到队列

|  |
| --- |
| private void enqueueEdit(Edit edit) {  if (LOG.isDebugEnabled()) {  LOG.debug("logEdit " + edit);  }  try {  // not checking for overflow yet to avoid penalizing performance of  // the common case. if there is persistent overflow, a mutex will be  // use to throttle contention on the queue.  // 将 RpcEdit 添加到队列  if (!editPendingQ.offer(edit)) {  Preconditions.checkState(  isSyncThreadAlive(), "sync thread is not alive");  if (Thread.holdsLock(this)) {  // if queue is full, synchronized caller must immediately relinquish  // the monitor before re-offering to avoid deadlock with sync thread  // which needs the monitor to write transactions.  int permits = overflowMutex.drainPermits();  try {  do {  this.wait(1000); // will be notified by next logSync.  } while (!editPendingQ.offer(edit));  } finally {  overflowMutex.release(permits);  }  } else {  // mutex will throttle contention during persistent overflow.  overflowMutex.acquire();  try {  editPendingQ.put(edit);  } finally {  overflowMutex.release();  }  }  }  } catch (Throwable t) {  // should never happen! failure to enqueue an edit is fatal  terminate(t);  }  } |

#### 2.3.2 异步同步 EditLog 到磁盘和 JournalNode

入口类：FSEditLogAsync.run()

|  |
| --- |
| @Override  public void run() {  try {  while (true) {  boolean doSync;  // 取出一个 RpcEdit  Edit edit = dequeueEdit();  if (edit != null) {  // sync if requested by edit log.  // 往下追 返回是否强制刷写 EditLog  doSync = edit.logEdit();  syncWaitQ.add(edit);  } else {  // sync when editq runs dry, but have edits pending a sync.  doSync = !syncWaitQ.isEmpty();  }  if (doSync) {  // normally edit log exceptions cause the NN to terminate, but tests  // relying on ExitUtil.terminate need to see the exception.  RuntimeException syncEx = null;  try {  // 刷写 EditLog  logSync(getLastWrittenTxId());  } catch (RuntimeException ex) {  syncEx = ex;  }  while ((edit = syncWaitQ.poll()) != null) {  edit.logSyncNotify(syncEx);  }  }  }  } catch (InterruptedException ie) {  LOG.info(Thread.currentThread().getName() + " was interrupted, exiting");  } catch (Throwable t) {  terminate(t);  }  } |

##### 2.3.2.1 返回是否强制刷写 EditLog

|  |
| --- |
| // return whether edit log wants to sync.  boolean logEdit() {  // 往下追  return log.doEditTransaction(op);  } |

|  |
| --- |
| synchronized boolean doEditTransaction(final FSEditLogOp op) {  // 开启事务  long start = beginTransaction();  op.setTransactionId(txid);  try {  // 写 EditLog  // 往本地写 EditLog 调用 EditLogFileOutputStream  // 往 JournalNode 写 EditLog 调用 JournalSetOutputStream (JournalSet)  editLogStream.write(op);  } catch (IOException ex) {  // All journals failed, it is handled in logSync.  } finally {  op.reset();  }  // 结束事务  endTransaction(start);  // 判断是否强制刷写 EditLog  return shouldForceSync();  } |

###### 2.3.2.1.1 开启事务

|  |
| --- |
| private long beginTransaction() {  assert Thread.holdsLock(this);  // get a new transactionId  // 事务 ID 累计  txid++;  //  // record the transactionId when new data was written to the edits log  //  TransactionId id = myTransactionId.get();  id.txid = txid;  return monotonicNow();  } |

###### 2.3.2.1.2 写 EditLog

2.3.2.1.2.1 往本地写 EditLog 调用 EditLogFileOutputStream

|  |
| --- |
| @Override  public void write(FSEditLogOp op) throws IOException {  // 往双缓冲写 EditLog  doubleBuf.writeOp(op, getCurrentLogVersion());  } |

|  |
| --- |
| public void writeOp(FSEditLogOp op, int logVersion) throws IOException {  // 往 bufCurrent 写 EditLog  bufCurrent.writeOp(op, logVersion);  } |

2.3.2.1.2.2 往 JournalNode 写 EditLog 调用 JournalSetOutputStream (JournalSet)

|  |
| --- |
| @Override  public void write(final FSEditLogOp op)  throws IOException {  mapJournalsAndReportErrors(new JournalClosure() {  @Override  public void apply(JournalAndStream jas) throws IOException {  if (jas.isActive()) {  // 调用 QuorumOutputStream.write  jas.getCurrentStream().write(op);  }  }  }, "write op");  } |

|  |
| --- |
| @Override  public void write(FSEditLogOp op) throws IOException {  // 往双缓冲写 EditLog  buf.writeOp(op, getCurrentLogVersion());  } |

|  |
| --- |
| public void writeOp(FSEditLogOp op, int logVersion) throws IOException {  // 往 bufCurrent 写 EditLog  bufCurrent.writeOp(op, logVersion);  } |

###### 2.3.2.1.3 结束事务

|  |
| --- |
| private void endTransaction(long start) {  assert Thread.holdsLock(this);  // update statistics  long end = monotonicNow();  numTransactions++;  totalTimeTransactions += (end - start);  if (metrics != null) // Metrics is non-null only when used inside name node  metrics.addTransaction(end - start);  } |

###### 2.3.2.1.4 判断是否强制刷写 EditLog

|  |
| --- |
| /\*\*  \* Check if should automatically sync buffered edits to  \* persistent store  \*  \* @return true if any of the edit stream says that it should sync  \*/  private boolean shouldForceSync() {  // 往下追  return editLogStream.shouldForceSync();  } |

|  |
| --- |
| /\*\*  \* @return true if the number of buffered data exceeds the intial buffer size  \*/  @Override  public boolean shouldForceSync() {  // 往下追  return doubleBuf.shouldForceSync();  } |

|  |
| --- |
| public boolean shouldForceSync() {  // 判断 bufCurrent 大小是否大于 512 KB  return bufCurrent.size() >= initBufferSize;  } |

##### 2.3.2.2 刷写 EditLog

|  |
| --- |
| protected void logSync(long mytxid) {  long syncStart = 0;  boolean sync = false;  long editsBatchedInSync = 0;  try {  EditLogOutputStream logStream = null;  synchronized (this) {  try {  printStatistics(false);  // if somebody is already syncing, then wait  // 已经有其他线程正在刷写 EditLog  while (mytxid > synctxid && isSyncRunning) {  try {  wait(1000);  } catch (InterruptedException ie) {  }  }  //  // If this transaction was already flushed, then nothing to do  //  if (mytxid <= synctxid) {  return;  }  // now, this thread will do the sync. track if other edits were  // included in the sync - ie. batched. if this is the only edit  // synced then the batched count is 0  editsBatchedInSync = txid - synctxid - 1;  syncStart = txid;  isSyncRunning = true;  sync = true;  // swap buffers  try {  if (journalSet.isEmpty()) {  throw new IOException("No journals available to flush");  }  // 双缓冲内存交换  editLogStream.setReadyToFlush();  } catch (IOException e) {  final String msg =  "Could not sync enough journals to persistent storage " +  "due to " + e.getMessage() + ". " +  "Unsynced transactions: " + (txid - synctxid);  LOG.error(msg, new Exception());  synchronized (journalSetLock) {  IOUtils.cleanupWithLogger(LOG, journalSet);  }  terminate(1, msg);  }  } finally {  // Prevent RuntimeException from blocking other log edit write  doneWithAutoSyncScheduling();  }  //editLogStream may become null,  //so store a local variable for flush.  logStream = editLogStream;  }  // do the sync  long start = monotonicNow();  try {  if (logStream != null) {  // 刷写 EditLog  logStream.flush();  }  } catch (IOException ex) {  synchronized (this) {  final String msg =  "Could not sync enough journals to persistent storage. "  + "Unsynced transactions: " + (txid - synctxid);  LOG.error(msg, new Exception());  synchronized (journalSetLock) {  IOUtils.cleanupWithLogger(LOG, journalSet);  }  terminate(1, msg);  }  }  long elapsed = monotonicNow() - start;  if (metrics != null) { // Metrics non-null only when used inside name node  metrics.addSync(elapsed);  metrics.incrTransactionsBatchedInSync(editsBatchedInSync);  numTransactionsBatchedInSync.addAndGet(editsBatchedInSync);  }  } finally {  // Prevent RuntimeException from blocking other log edit sync  synchronized (this) {  if (sync) {  synctxid = syncStart;  for (JournalManager jm : journalSet.getJournalManagers()) {  /\*\*  \* {@link FileJournalManager#lastReadableTxId} is only meaningful  \* for file-based journals. Therefore the interface is not added to  \* other types of {@link JournalManager}.  \*/  if (jm instanceof FileJournalManager) {  ((FileJournalManager) jm).setLastReadableTxId(syncStart);  }  }  isSyncRunning = false;  }  this.notifyAll();  }  }  } |

|  |
| --- |
| /\*\*  \* Flush data to persistent store.  \* Collect sync metrics.  \*/  public void flush() throws IOException {  // 本地刷写 EditLog 往下追  // JournalNode 刷写 EditLog 调用子类  flush(true);  } |

###### 2.3.2.2.1 本地刷写 EditLog

|  |
| --- |
| public void flush(boolean durable) throws IOException {  numSync++;  long start = monotonicNow();  // 刷写  flushAndSync(durable);  long end = monotonicNow();  totalTimeSync += (end - start);  } |

|  |
| --- |
| /\*\*  \* Flush ready buffer to persistent store. currentBuffer is not flushed as it  \* accumulates new log records while readyBuffer will be flushed and synced.  \*/  @Override  public void flushAndSync(boolean durable) throws IOException {  if (fp == null) {  throw new IOException("Trying to use aborted output stream");  }  if (doubleBuf.isFlushed()) {  LOG.info("Nothing to flush");  return;  }  preallocate(); // preallocate file if necessary  // 刷写  doubleBuf.flushTo(fp);  if (durable && !shouldSkipFsyncForTests && !shouldSyncWritesAndSkipFsync) {  fc.force(false); // metadata updates not needed  }  } |

|  |
| --- |
| /\*\*  \* Writes the content of the "ready" buffer to the given output stream,  \* and resets it. Does not swap any buffers.  \*/  public void flushTo(OutputStream out) throws IOException {  // 往下追  bufReady.writeTo(out); // write data to file  bufReady.reset(); // erase all data in the buffer  } |

|  |
| --- |
| /\*\* Write to a file stream \*/  public void writeTo(OutputStream out) throws IOException {  // 刷写  buffer.writeTo(out);  } |

###### 2.3.2.2.2 往 JournalNode 刷写 EditLog

|  |
| --- |
| @Override  public void flush() throws IOException {  mapJournalsAndReportErrors(new JournalClosure() {  @Override  public void apply(JournalAndStream jas) throws IOException {  if (jas.isActive()) {  // 往 JournalNode 刷写 EditLog  jas.getCurrentStream().flush();  }  }  }, "flush");  } |

|  |
| --- |
| @Override  protected void flushAndSync(final boolean durable) throws IOException {  mapJournalsAndReportErrors(new JournalClosure() {  @Override  public void apply(JournalAndStream jas) throws IOException {  if (jas.isActive()) {  jas.getCurrentStream().flushAndSync(durable);  }  }  }, "flushAndSync");  } |

|  |
| --- |
| @Override  protected void flushAndSync(boolean durable) throws IOException {  int numReadyBytes = buf.countReadyBytes();  if (numReadyBytes > 0) {  int numReadyTxns = buf.countReadyTxns();  long firstTxToFlush = buf.getFirstReadyTxId();  assert numReadyTxns > 0;  // Copy from our double-buffer into a new byte array. This is for  // two reasons:  // 1) The IPC code has no way of specifying to send only a slice of  // a larger array.  // 2) because the calls to the underlying nodes are asynchronous, we  // need a defensive copy to avoid accidentally mutating the buffer  // before it is sent.  // 计算刷写数据  DataOutputBuffer bufToSend = new DataOutputBuffer(numReadyBytes);  buf.flushTo(bufToSend);  assert bufToSend.getLength() == numReadyBytes;  byte[] data = bufToSend.getData();  assert data.length == bufToSend.getLength();  // 往下追  QuorumCall<AsyncLogger, Void> qcall = loggers.sendEdits(  segmentTxId,  firstTxToFlush,  numReadyTxns,  data);  loggers.waitForWriteQuorum(qcall, writeTimeoutMs, "sendEdits");  // Since we successfully wrote this batch, let the loggers know. Any future  // RPCs will thus let the loggers know of the most recent transaction, even  // if a logger has fallen behind.  loggers.setCommittedTxId(firstTxToFlush + numReadyTxns - 1);  }  } |

|  |
| --- |
| public QuorumCall<AsyncLogger, Void> sendEdits(  long segmentTxId, long firstTxnId, int numTxns, byte[] data) {  Map<AsyncLogger, ListenableFuture<Void>> calls = Maps.newHashMap();  // 变量每个 JournalNode 连接对象  for (AsyncLogger logger : loggers) {  // 往下追  ListenableFuture<Void> future =  logger.sendEdits(segmentTxId, firstTxnId, numTxns, data);  calls.put(logger, future);  }  return QuorumCall.create(calls);  } |

2.3.2.2.2.1 最终调用IPCLoggerChannel.sendEdits() 发送异步 RPC

|  |
| --- |
| @Override  public ListenableFuture<Void> sendEdits(  final long segmentTxId, final long firstTxnId,  final int numTxns, final byte[] data) {  try {  reserveQueueSpace(data.length);  } catch (LoggerTooFarBehindException e) {  return Futures.immediateFailedFuture(e);  }  // When this batch is acked, we use its submission time in order  // to calculate how far we are lagging.  final long submitNanos = System.nanoTime();  ListenableFuture<Void> ret = null;  try {  ret = singleThreadExecutor.submit(new Callable<Void>() {  @Override  public Void call() throws IOException {  throwIfOutOfSync();  long rpcSendTimeNanos = System.nanoTime();  try {  // 调用 JournalNodeRpcServer.journal()  getProxy().journal(createReqInfo(),  segmentTxId, firstTxnId, numTxns, data);  } catch (IOException e) {  QuorumJournalManager.LOG.warn(  "Remote journal " + IPCLoggerChannel.this + " failed to " +  "write txns " + firstTxnId + "-" + (firstTxnId + numTxns - 1) +  ". Will try to write to this JN again after the next " +  "log roll.", e);  synchronized (IPCLoggerChannel.this) {  outOfSync = true;  }  throw e;  } finally {  long now = System.nanoTime();  long rpcTime = TimeUnit.MICROSECONDS.convert(  now - rpcSendTimeNanos, TimeUnit.NANOSECONDS);  long endToEndTime = TimeUnit.MICROSECONDS.convert(  now - submitNanos, TimeUnit.NANOSECONDS);  metrics.addWriteEndToEndLatency(endToEndTime);  metrics.addWriteRpcLatency(rpcTime);  if (rpcTime / 1000 > WARN\_JOURNAL\_MILLIS\_THRESHOLD) {  QuorumJournalManager.LOG.warn(  "Took " + (rpcTime / 1000) + "ms to send a batch of " +  numTxns + " edits (" + data.length + " bytes) to " +  "remote journal " + IPCLoggerChannel.this);  }  }  synchronized (IPCLoggerChannel.this) {  highestAckedTxId = firstTxnId + numTxns - 1;  lastAckNanos = submitNanos;  }  return null;  }  });  } finally {  if (ret == null) {  // it didn't successfully get submitted,  // so adjust the queue size back down.  unreserveQueueSpace(data.length);  } else {  // It was submitted to the queue, so adjust the length  // once the call completes, regardless of whether it  // succeeds or fails.  Futures.addCallback(ret, new FutureCallback<Void>() {  @Override  public void onFailure(Throwable t) {  unreserveQueueSpace(data.length);  }  @Override  public void onSuccess(Void t) {  unreserveQueueSpace(data.length);  }  }, MoreExecutors.directExecutor());  }  }  return ret;  } |

2.3.2.2.2.1.1 调用 JournalNodeRpcServer.journal()

|  |
| --- |
| @Override  public void journal(RequestInfo reqInfo,  long segmentTxId, long firstTxnId,  int numTxns, byte[] records) throws IOException {  jn.getOrCreateJournal(reqInfo.getJournalId(), reqInfo.getNameServiceId())  .journal(reqInfo, segmentTxId, firstTxnId, numTxns, records);  } |

### 2.4 NameNode 异步 RPC 发送 EditLog 给 JournalNode

入口类:JournalNodeRpcServer.journal()

|  |
| --- |
| @Override  public void journal(RequestInfo reqInfo,  long segmentTxId, long firstTxnId,  int numTxns, byte[] records) throws IOException {  // 接收 NameNode 异步 RPC 发送 EditLog 元数据    // 创建 Journal 对象  jn.getOrCreateJournal(  reqInfo.getJournalId(),  reqInfo.getNameServiceId()  )  // 执行接收 RPC 发送的 EditLog 元数据请求  .journal(reqInfo, segmentTxId, firstTxnId, numTxns, records);  } |

#### 2.4.1 创建 Journal 对象

|  |
| --- |
| public Journal getOrCreateJournal(String jid,  String nameServiceId)  throws IOException {  // 往下追  return getOrCreateJournal(jid, nameServiceId, StartupOption.REGULAR);  } |

|  |
| --- |
| synchronized Journal getOrCreateJournal(String jid,  String nameServiceId,  StartupOption startOpt)  throws IOException {  QuorumJournalManager.checkJournalId(jid);  Journal journal = journalsById.get(jid);  if (journal == null) {  // 获取 JournalNode 存储元数据目录  File logDir = getLogDir(jid, nameServiceId);  // Initializing journal in directory /opt/app/hadoop-3.1.3/data/jn/mycluster  LOG.info("Initializing journal in directory " + logDir);  // 创建 Journal  journal = new Journal(conf, logDir, jid, startOpt, new ErrorReporter());  // 缓存  journalsById.put(jid, journal);  // Start SyncJouranl thread, if JournalNode Sync is enabled  if (conf.getBoolean(  DFSConfigKeys.DFS\_JOURNALNODE\_ENABLE\_SYNC\_KEY,  DFSConfigKeys.DFS\_JOURNALNODE\_ENABLE\_SYNC\_DEFAULT)) {  // 开启同步线程 同步 JournalNode 之间的相关信息  startSyncer(journal, jid, nameServiceId);  }  } else if (journalSyncersById.get(jid) != null &&  !journalSyncersById.get(jid).isJournalSyncerStarted() &&  !journalsById.get(jid).getTriedJournalSyncerStartedwithnsId() &&  nameServiceId != null) {  startSyncer(journal, jid, nameServiceId);  }  return journal;  } |

##### 2.4.1.1 执行创建 Journal 对象

|  |
| --- |
| Journal(Configuration conf, File logDir, String journalId,  StartupOption startOpt, StorageErrorReporter errorReporter)  throws IOException {  this.conf = conf;  // 创建 JNStorage  storage = new JNStorage(conf, logDir, startOpt, errorReporter);  // mycluster  this.journalId = journalId;  refreshCachedData();  // FileJournalManager  this.fjm = storage.getJournalManager();  this.cache = createCache();  this.metrics = JournalMetrics.create(this);  // 扫描最新的 EditLog 文件  EditLogFile latest = scanStorageForLatestEdits();  if (latest != null) {  updateHighestWrittenTxId(latest.getLastTxId());  }  } |

##### 2.4.1.1 启动同步线程 (同步JournalNode 之间相关信息确保一致性)

|  |
| --- |
| private void startSyncer(Journal journal, String jid, String nameServiceId) {  JournalNodeSyncer jSyncer = journalSyncersById.get(jid);  if (jSyncer == null) {  // 创建 JournalNodeSyncer  jSyncer = new JournalNodeSyncer(this, journal, jid, conf, nameServiceId);  // 缓存  journalSyncersById.put(jid, jSyncer);  }  // 启动 JournalNodeSyncer 执行 JournalNode 之间同步相关信息  jSyncer.start(nameServiceId);  } |

###### 2.4.1.1.1 创建 JournalNodeSyncer

|  |
| --- |
| JournalNodeSyncer(JournalNode jouranlNode, Journal journal, String jid,  Configuration conf, String nameServiceId) {  // 赋值操作  this.jn = jouranlNode;  this.journal = journal;  this.jid = jid;  this.nameServiceId = nameServiceId;  this.jnStorage = journal.getStorage();  this.conf = conf;  // 120s  journalSyncInterval = conf.getLong(  DFSConfigKeys.DFS\_JOURNALNODE\_SYNC\_INTERVAL\_KEY,  DFSConfigKeys.DFS\_JOURNALNODE\_SYNC\_INTERVAL\_DEFAULT);  // 30  logSegmentTransferTimeout = conf.getInt(  DFSConfigKeys.DFS\_EDIT\_LOG\_TRANSFER\_TIMEOUT\_KEY,  DFSConfigKeys.DFS\_EDIT\_LOG\_TRANSFER\_TIMEOUT\_DEFAULT);  throttler = getThrottler(conf);  metrics = journal.getMetrics();  journalSyncerStarted = false;  } |

###### 2.4.1.1.2 启动线程

|  |
| --- |
| public void start(String nsId) {  if (nsId != null) {  // mycluster  this.nameServiceId = nsId;  journal.setTriedJournalSyncerStartedwithnsId(true);  }  if (!journalSyncerStarted && // 获取其他 JournalNode 代理  getOtherJournalNodeProxies()  ) {  // Starting SyncJournal daemon for journal mycluster  LOG.info("Starting SyncJournal daemon for journal " + jid);  // 启动同步 JournalNode EditLog 线程  startSyncJournalsDaemon();  journalSyncerStarted = true;  }  } |

2.4.1.1.2.1 获取其他 JournalNode 代理

|  |
| --- |
| private boolean getOtherJournalNodeProxies() {  // 获取其他 JournalNode 地址  List<InetSocketAddress> otherJournalNodes = getOtherJournalNodeAddrs();  if (otherJournalNodes == null || otherJournalNodes.isEmpty()) {  LOG.warn("Other JournalNode addresses not available. Journal Syncing " +  "cannot be done");  return false;  }  for (InetSocketAddress addr : otherJournalNodes) {  try {  // 添加其他 JournalNode 代理  otherJNProxies.add(new JournalNodeProxy(addr));  } catch (IOException e) {  LOG.warn("Could not add proxy for Journal at addresss " + addr, e);  }  }  if (otherJNProxies.isEmpty()) {  LOG.error("Cannot sync as there is no other JN available for sync.");  return false;  }  numOtherJNs = otherJNProxies.size();  return true;  } |

2.4.1.1.2.1.1 创建 JournalNodeProxy

|  |
| --- |
| JournalNodeProxy(InetSocketAddress jnAddr) throws IOException {  final Configuration confCopy = new Configuration(conf);  this.jnAddr = jnAddr;  // 获取其他 JournalNode 的代理  this.jnProxy = SecurityUtil.doAsLoginUser(  new PrivilegedExceptionAction<InterQJournalProtocol>() {  @Override  public InterQJournalProtocol run() throws IOException {  // 设置 JournalNode 之间内部通讯协议接口 InterQJournalProtocol  RPC.setProtocolEngine(confCopy, InterQJournalProtocolPB.class,  ProtobufRpcEngine.class);  // 获取其他 JournalNode 的代理  InterQJournalProtocolPB interQJournalProtocolPB = RPC.getProxy(  InterQJournalProtocolPB.class,  RPC.getProtocolVersion(InterQJournalProtocolPB.class),  jnAddr, confCopy);  return new InterQJournalProtocolTranslatorPB(  interQJournalProtocolPB);  }  });  } |

2.4.1.1.2.2 执行启动线程 (创建 EditLog 存储目录)

|  |
| --- |
| private void startSyncJournalsDaemon() {  // 创建线程并启动  syncJournalDaemon = new Daemon(() -> {  // Wait for journal to be formatted to create edits.sync directory  while(!journal.isFormatted()) {  try {  Thread.sleep(journalSyncInterval);  } catch (InterruptedException e) {  LOG.error("JournalNodeSyncer daemon received Runtime exception.", e);  Thread.currentThread().interrupt();  return;  }  }  // 创建 JournalNode 同步 EditLog 目录  if (!createEditsSyncDir()) {  LOG.error("Failed to create directory for downloading log " +  "segments: %s. Stopping Journal Node Sync.",  journal.getStorage().getEditsSyncDir());  return;  }  while(shouldSync) {  try {  if (!journal.isFormatted()) {  LOG.warn("Journal cannot sync. Not formatted.");  } else {  // 执行 同步 JournalNode 相关的信息  syncJournals();  }  } catch (Throwable t) {  if (!shouldSync) {  if (t instanceof InterruptedException) {  LOG.info("Stopping JournalNode Sync.");  Thread.currentThread().interrupt();  return;  } else {  LOG.warn("JournalNodeSyncer received an exception while " +  "shutting down.", t);  }  break;  } else {  if (t instanceof InterruptedException) {  LOG.warn("JournalNodeSyncer interrupted", t);  Thread.currentThread().interrupt();  return;  }  }  LOG.error(  "JournalNodeSyncer daemon received Runtime exception. ", t);  }  try {  // 120s  Thread.sleep(journalSyncInterval);  } catch (InterruptedException e) {  if (!shouldSync) {  LOG.info("Stopping JournalNode Sync.");  } else {  LOG.warn("JournalNodeSyncer interrupted", e);  }  Thread.currentThread().interrupt();  return;  }  }  });  syncJournalDaemon.start();  } |

#### 2.4.2 执行接收 RPC 发送的 EditLog 元数据请求 (写磁盘)

|  |
| --- |
| /\*\*  \* Write a batch of edits to the journal.  \* {@see QJournalProtocol#journal(RequestInfo, long, long, int, byte[])}  \*/  synchronized void journal(RequestInfo reqInfo,  long segmentTxId, long firstTxnId,  int numTxns, byte[] records) throws IOException {  checkFormatted();  checkWriteRequest(reqInfo);  // If numTxns is 0, it's actually a fake send which aims at updating  // committedTxId only. So we can return early.  if (numTxns == 0) {  return;  }  checkSync(curSegment != null,  "Can't write, no segment open" + " ; journal id: " + journalId);  if (curSegmentTxId != segmentTxId) {  // Sanity check: it is possible that the writer will fail IPCs  // on both the finalize() and then the start() of the next segment.  // This could cause us to continue writing to an old segment  // instead of rolling to a new one, which breaks one of the  // invariants in the design. If it happens, abort the segment  // and throw an exception.  JournalOutOfSyncException e = new JournalOutOfSyncException(  "Writer out of sync: it thinks it is writing segment " + segmentTxId  + " but current segment is " + curSegmentTxId  + " ; journal id: " + journalId);  abortCurSegment();  throw e;  }    checkSync(nextTxId == firstTxnId,  "Can't write txid " + firstTxnId + " expecting nextTxId=" + nextTxId  + " ; journal id: " + journalId);    long lastTxnId = firstTxnId + numTxns - 1;  if (LOG.isTraceEnabled()) {  LOG.trace("Writing txid " + firstTxnId + "-" + lastTxnId +  " ; journal id: " + journalId);  }  if (cache != null) {  // 缓存 EditLog  cache.storeEdits(records, firstTxnId, lastTxnId, curSegmentLayoutVersion);  }  // If the edit has already been marked as committed, we know  // it has been fsynced on a quorum of other nodes, and we are  // "catching up" with the rest. Hence we do not need to fsync.  boolean isLagging = lastTxnId <= committedTxnId.get();  boolean shouldFsync = !isLagging;  // 往双缓存写 EditLog  curSegment.writeRaw(records, 0, records.length);  // 交换双缓存  curSegment.setReadyToFlush();  StopWatch sw = new StopWatch();  sw.start();  // 刷写缓存 EditLog 到磁盘  curSegment.flush(shouldFsync);  sw.stop();  long nanoSeconds = sw.now();  metrics.addSync(  TimeUnit.MICROSECONDS.convert(nanoSeconds, TimeUnit.NANOSECONDS));  long milliSeconds = TimeUnit.MILLISECONDS.convert(  nanoSeconds, TimeUnit.NANOSECONDS);  if (milliSeconds > WARN\_SYNC\_MILLIS\_THRESHOLD) {  LOG.warn("Sync of transaction range " + firstTxnId + "-" + lastTxnId +  " took " + milliSeconds + "ms" + " ; journal id: " + journalId);  }  if (isLagging) {  // This batch of edits has already been committed on a quorum of other  // nodes. So, we are in "catch up" mode. This gets its own metric.  metrics.batchesWrittenWhileLagging.incr(1);  }    metrics.batchesWritten.incr(1);  metrics.bytesWritten.incr(records.length);  metrics.txnsWritten.incr(numTxns);    updateHighestWrittenTxId(lastTxnId);  nextTxId = lastTxnId + 1;  lastJournalTimestamp = Time.now();  } |

### 2.5 Standby NameNode 拉取 JournalNode EditLog 元数据信息

入口类：EditLogTailer.EditLogTailerThread.run()

|  |
| --- |
| @Override  public void run() {  SecurityUtil.doAsLoginUserOrFatal(  new PrivilegedAction<Object>() {  @Override  public Object run() {  // 执行  doWork();  return null;  }  });  } |

|  |
| --- |
| private void doWork() {  long currentSleepTimeMs = sleepTimeMs;  while (shouldRun) {  long editsTailed = 0;  try {  // There's no point in triggering a log roll if the Standby hasn't  // read any more transactions since the last time a roll was  // triggered.  // 触发 EditLog 文件滚动判断  boolean triggeredLogRoll = false;  if (// 超过 120s 没有更新  tooLongSinceLastLoad() &&  lastRollTriggerTxId < lastLoadedTxnId) {  // 触发 Active EditLog 滚动  triggerActiveLogRoll();  triggeredLogRoll = true;  }  /\*\*  \* Check again in case someone calls {@link EditLogTailer#stop} while  \* we're triggering an edit log roll, since ipc.Client catches and  \* ignores {@link InterruptedException} in a few places. This fixes  \* the bug described in HDFS-2823.  \*/  if (!shouldRun) {  break;  }  // Prevent reading of name system while being modified. The full  // name system lock will be acquired to further block even the block  // state updates.  namesystem.cpLockInterruptibly();  long startTime = Time.monotonicNow();  try {  NameNode.getNameNodeMetrics().addEditLogTailInterval(  startTime - lastLoadTimeMs);  // 往下追  editsTailed = doTailEdits();  } finally {  namesystem.cpUnlock();  NameNode.getNameNodeMetrics().addEditLogTailTime(  Time.monotonicNow() - startTime);  }  // Update NameDirSize Metric  if (triggeredLogRoll) {  namesystem.getFSImage().getStorage().updateNameDirSize();  }  } catch (EditLogInputException elie) {  LOG.warn("Error while reading edits from disk. Will try again.", elie);  } catch (InterruptedException ie) {  // interrupter should have already set shouldRun to false  continue;  } catch (Throwable t) {  LOG.fatal("Unknown error encountered while tailing edits. " +  "Shutting down standby NN.", t);  terminate(1, t);  }  try {  if (editsTailed == 0 && maxSleepTimeMs > 0) {  // If no edits were tailed, apply exponential backoff  // before tailing again. Double the current sleep time on each  // empty response, but don't exceed the max. If the sleep time  // was configured as 0, start the backoff at 1 ms.  currentSleepTimeMs = Math.min(maxSleepTimeMs,  (currentSleepTimeMs == 0 ? 1 : currentSleepTimeMs) \* 2);  } else {  currentSleepTimeMs = sleepTimeMs; // reset to initial sleep time  }  // 睡眠  EditLogTailer.this.sleep(currentSleepTimeMs);  } catch (InterruptedException e) {  LOG.warn("Edit log tailer interrupted", e);  }  }  }  } |

#### 2.5.1 触发 EditLog 文件滚动判断

|  |
| --- |
| /\*\*  \* @return true if the configured log roll period has elapsed.  \*/  private boolean tooLongSinceLastLoad() {  // 120s >= 0  return logRollPeriodMs >= 0 &&  (monotonicNow() - lastRollTimeMs) > logRollPeriodMs;  } |

##### 2.5.1.1 触发 Active NameNode EditLog 滚动

|  |
| --- |
| /\*\*  \* Trigger the active node to roll its logs.  \*/  @VisibleForTesting  void triggerActiveLogRoll() {  LOG.info("Triggering log roll on remote NameNode");  Future<Void> future = null;  try {  // 发送 RPC 请求给 Active NameNode 滚动 EditLog  future = rollEditsRpcExecutor.submit(  // 发送 RPC 请求  getNameNodeProxy()  );  future.get(rollEditsTimeoutMs, TimeUnit.MILLISECONDS);  lastRollTimeMs = monotonicNow();  lastRollTriggerTxId = lastLoadedTxnId;  } catch (ExecutionException e) {  LOG.warn("Unable to trigger a roll of the active NN", e);  } catch (TimeoutException e) {  if (future != null) {  future.cancel(true);  }  LOG.warn(String.format(  "Unable to finish rolling edits in %d ms", rollEditsTimeoutMs));  } catch (InterruptedException e) {  LOG.warn("Unable to trigger a roll of the active NN", e);  }  } |

|  |
| --- |
| /\*\*  \* NameNodeProxy factory method.  \*  \* @return a Callable to roll logs on remote NameNode.  \*/  @VisibleForTesting  Callable<Void> getNameNodeProxy() {  return new MultipleNameNodeProxy<Void>() {  @Override  protected Void doWork() throws IOException {  // 调用 NameNodeRpcServer.rollEditLog() (通讯协议接口 NamenodeProtocol)  cachedActiveProxy.rollEditLog();  return null;  }  };  } |

#### 2.5.2 执行拉取 JournalNode EditLog 数据 (HTTP GET 方式)

|  |
| --- |
| @VisibleForTesting  public long doTailEdits() throws IOException, InterruptedException {  // Write lock needs to be interruptible here because the  // transitionToActive RPC takes the write lock before calling  // tailer.stop() -- so if we're not interruptible, it will  // deadlock.  namesystem.writeLockInterruptibly();  try {  // 获取 FSImage  FSImage image = namesystem.getFSImage();  long lastTxnId = image.getLastAppliedTxId();  if (LOG.isDebugEnabled()) {  LOG.debug("lastTxnId: " + lastTxnId);  }  Collection<EditLogInputStream> streams;  long startTime = Time.monotonicNow();  try {  // 选择 JournalNode 流  streams = editLog.selectInputStreams(lastTxnId + 1, 0,  null, inProgressOk, true);  } catch (IOException ioe) {  // This is acceptable. If we try to tail edits in the middle of an edits  // log roll, i.e. the last one has been finalized but the new inprogress  // edits file hasn't been started yet.  LOG.warn("Edits tailer failed to find any streams. Will try again " +  "later.", ioe);  return 0;  } finally {  NameNode.getNameNodeMetrics().addEditLogFetchTime(  Time.monotonicNow() - startTime);  }  if (LOG.isDebugEnabled()) {  LOG.debug("edit streams to load from: " + streams.size());  }  // Once we have streams to load, errors encountered are legitimate cause  // for concern, so we don't catch them here. Simple errors reading from  // disk are ignored.  long editsLoaded = 0;  try {  // 从 streams 拉取 EditLog  editsLoaded = image.loadEdits(  streams, namesystem, maxTxnsPerLock, null, null);  } catch (EditLogInputException elie) {  editsLoaded = elie.getNumEditsLoaded();  throw elie;  } finally {  if (editsLoaded > 0 || LOG.isDebugEnabled()) {  LOG.debug(String.format("Loaded %d edits starting from txid %d ",  editsLoaded, lastTxnId));  }  NameNode.getNameNodeMetrics().addNumEditLogLoaded(editsLoaded);  }  if (editsLoaded > 0) {  lastLoadTimeMs = monotonicNow();  }  lastLoadedTxnId = image.getLastAppliedTxId();  return editsLoaded;  } finally {  namesystem.writeUnlock();  }  } |

##### 2.5.2.1 选择 JournalNode 流

|  |
| --- |
| /\*\*  \* Select a list of input streams.  \*  \* @param fromTxId first transaction in the selected streams  \* @param toAtLeastTxId the selected streams must contain this transaction  \* @param recovery recovery context  \* @param inProgressOk set to true if in-progress streams are OK  \* @param onlyDurableTxns set to true if streams are bounded  \* by the durable TxId  \*/  public Collection<EditLogInputStream> selectInputStreams(long fromTxId,  long toAtLeastTxId, MetaRecoveryContext recovery, boolean inProgressOk,  boolean onlyDurableTxns) throws IOException {  List<EditLogInputStream> streams = new ArrayList<EditLogInputStream>();  synchronized (journalSetLock) {  Preconditions.checkState(journalSet.isOpen(), "Cannot call " +  "selectInputStreams() on closed FSEditLog");  // 往下追  selectInputStreams(streams, fromTxId, inProgressOk, onlyDurableTxns);  }  try {  checkForGaps(streams, fromTxId, toAtLeastTxId, inProgressOk);  } catch (IOException e) {  if (recovery != null) {  // If recovery mode is enabled, continue loading even if we know we  // can't load up to toAtLeastTxId.  LOG.error("Exception while selecting input streams", e);  } else {  closeAllStreams(streams);  throw e;  }  }  return streams;  } |

|  |
| --- |
| @Override  public void selectInputStreams(Collection<EditLogInputStream> streams,  long fromTxId, boolean inProgressOk, boolean onlyDurableTxns)  throws IOException {  // 往下追  journalSet.selectInputStreams(  streams, fromTxId,  inProgressOk, onlyDurableTxns);  } |

|  |
| --- |
| /\*\*  \* In this function, we get a bunch of streams from all of our JournalManager  \* objects. Then we add these to the collection one by one.  \*  \* @param streams The collection to add the streams to. It may or  \* may not be sorted-- this is up to the caller.  \* @param fromTxId The transaction ID to start looking for streams at  \* @param inProgressOk Should we consider unfinalized streams?  \* @param onlyDurableTxns Set to true if streams are bounded by the durable  \* TxId. A durable TxId is the committed txid in QJM  \* or the largest txid written into file in FJM  \*/  @Override  public void selectInputStreams(Collection<EditLogInputStream> streams,  long fromTxId, boolean inProgressOk, boolean onlyDurableTxns) {  final PriorityQueue<EditLogInputStream> allStreams =  new PriorityQueue<EditLogInputStream>(64,  EDIT\_LOG\_INPUT\_STREAM\_COMPARATOR);  // 遍历每个 JournalNode  for (JournalAndStream jas : journals) {  if (jas.isDisabled()) {  LOG.info("Skipping jas " + jas + " since it's disabled");  continue;  }  try {  // 往下追  jas.getManager()  // 调用 QuorumJournalManager  .selectInputStreams(allStreams, fromTxId,  inProgressOk, onlyDurableTxns);  } catch (IOException ioe) {  LOG.warn("Unable to determine input streams from " + jas.getManager() +  ". Skipping.", ioe);  }  }  chainAndMakeRedundantStreams(streams, allStreams, fromTxId);  } |

###### 2.5.2.1.1 调用 QuorumJournalManager.selectInputStreams() 进行选择

|  |
| --- |
| @Override  public void selectInputStreams(Collection<EditLogInputStream> streams,  long fromTxnId, boolean inProgressOk,  boolean onlyDurableTxns) throws IOException {  // Some calls will use inProgressOK to get in-progress edits even if  // the cache used for RPC calls is not enabled; fall back to using the  // streaming mechanism to serve such requests  if (inProgressOk && inProgressTailingEnabled) {  if (LOG.isDebugEnabled()) {  LOG.debug("Tailing edits starting from txn ID " + fromTxnId +  " via RPC mechanism");  }  try {  Collection<EditLogInputStream> rpcStreams = new ArrayList<>();  selectRpcInputStreams(rpcStreams, fromTxnId, onlyDurableTxns);  streams.addAll(rpcStreams);  return;  } catch (IOException ioe) {  LOG.warn("Encountered exception while tailing edits >= " + fromTxnId +  " via RPC; falling back to streaming.", ioe);  }  }  // 选择流式输入流  selectStreamingInputStreams(streams, fromTxnId, inProgressOk,  onlyDurableTxns);  } |

|  |
| --- |
| /\*\*  \* Select input streams from the journals, specifically using the streaming  \* mechanism optimized for resiliency / bulk load.  \*/  private void selectStreamingInputStreams(  Collection<EditLogInputStream> streams, long fromTxnId,  boolean inProgressOk, boolean onlyDurableTxns) throws IOException {  // 通过 RPC Request 请求获取 JournalNode EditLog 清单  QuorumCall<AsyncLogger, RemoteEditLogManifest> q =  loggers.getEditLogManifest(fromTxnId, inProgressOk);  Map<AsyncLogger, RemoteEditLogManifest> resps =  loggers.waitForWriteQuorum(q, selectInputStreamsTimeoutMs,  "selectStreamingInputStreams");  LOG.debug("selectStreamingInputStream manifests:\n" +  Joiner.on("\n").withKeyValueSeparator(": ").join(resps));  final PriorityQueue<EditLogInputStream> allStreams =  new PriorityQueue<EditLogInputStream>(64,  JournalSet.EDIT\_LOG\_INPUT\_STREAM\_COMPARATOR);  for (Map.Entry<AsyncLogger, RemoteEditLogManifest> e : resps.entrySet()) {  AsyncLogger logger = e.getKey();  RemoteEditLogManifest manifest = e.getValue();  long committedTxnId = manifest.getCommittedTxnId();  for (RemoteEditLog remoteLog : manifest.getLogs()) {  // 构建拉取 JournalNode EditLog URL (HTTP 方式)  URL url = logger.buildURLToFetchLogs(remoteLog.getStartTxId());  long endTxId = remoteLog.getEndTxId();  // If it's bounded by durable Txns, endTxId could not be larger  // than committedTxnId. This ensures the consistency.  // We don't do the following for finalized log segments, since all  // edits in those are guaranteed to be committed.  if (onlyDurableTxns && inProgressOk && remoteLog.isInProgress()) {  endTxId = Math.min(endTxId, committedTxnId);  if (endTxId < remoteLog.getStartTxId()) {  LOG.warn("Found endTxId (" + endTxId + ") that is less than " +  "the startTxId (" + remoteLog.getStartTxId() +  ") - setting it to startTxId.");  endTxId = remoteLog.getStartTxId();  }  }  // 创建 HTTP 拉取 JournalNode EditLog 输入流 返回 EditLogFileInputStream  EditLogInputStream elis = EditLogFileInputStream.fromUrl(  connectionFactory, url, remoteLog.getStartTxId(),  endTxId, remoteLog.isInProgress());  // 添加  allStreams.add(elis);  }  }  // 往下追  JournalSet.chainAndMakeRedundantStreams(streams, allStreams, fromTxnId);  } |

2.5.2.1.1.1 构建拉取 JournalNode EditLog URL (HTTP 方式)

|  |
| --- |
| @Override  public URL buildURLToFetchLogs(long segmentTxId) {  Preconditions.checkArgument(segmentTxId > 0,  "Invalid segment: %s", segmentTxId);  Preconditions.checkState(hasHttpServerEndPoint(), "No HTTP/HTTPS endpoint");  try {  // 构建 HTTP 方式拉取 JournalNode EditLog  String path = GetJournalEditServlet.buildPath(  journalId, segmentTxId, nsInfo, true);  return new URL(httpServerURL, path);  } catch (MalformedURLException e) {  // should never get here.  throw new RuntimeException(e);  }  } |

|  |
| --- |
| public static String buildPath(String journalId, long segmentTxId,  NamespaceInfo nsInfo, boolean inProgressOk) {  // 构建 PATH  StringBuilder path = new StringBuilder("/getJournal?");  try {  path.append(JOURNAL\_ID\_PARAM).append("=")  .append(URLEncoder.encode(journalId, "UTF-8"));  path.append("&" + SEGMENT\_TXID\_PARAM).append("=")  .append(segmentTxId);  path.append("&" + STORAGEINFO\_PARAM).append("=")  .append(URLEncoder.encode(nsInfo.toColonSeparatedString(), "UTF-8"));  path.append("&" + IN\_PROGRESS\_OK).append("=")  .append(inProgressOk);  } catch (UnsupportedEncodingException e) {  // Never get here -- everyone supports UTF-8  throw new RuntimeException(e);  }  return path.toString();  } |

###### 2.5.2.1.2 创建 HTTP 拉取 JournalNode EditLog 输入流 返回 EditLogFileInputStream

|  |
| --- |
| /\*\*  \* Open an EditLogInputStream for the given URL.  \*  \* @param connectionFactory  \* the URLConnectionFactory used to create the connection.  \* @param url  \* the url hosting the log  \* @param startTxId  \* the expected starting txid  \* @param endTxId  \* the expected ending txid  \* @param inProgress  \* whether the log is in-progress  \* @return a stream from which edits may be read  \*/  public static EditLogInputStream fromUrl(  URLConnectionFactory connectionFactory, URL url, long startTxId,  long endTxId, boolean inProgress) {  // 创建 EditLogFileInputStream  return new EditLogFileInputStream(new URLLog(connectionFactory, url),  startTxId, endTxId, inProgress);  } |

|  |
| --- |
| private EditLogFileInputStream(LogSource log,  long firstTxId, long lastTxId,  boolean isInProgress) {  //  this.log = log;  this.firstTxId = firstTxId;  this.lastTxId = lastTxId;  this.isInProgress = isInProgress;  this.maxOpSize = DFSConfigKeys.DFS\_NAMENODE\_MAX\_OP\_SIZE\_DEFAULT;  } |

##### 2.5.2.2 从 streams 拉取 EditLog

|  |
| --- |
| public long loadEdits(Iterable<EditLogInputStream> editStreams,  FSNamesystem target, long maxTxnsToRead,  StartupOption startOpt, MetaRecoveryContext recovery)  throws IOException {  LOG.debug("About to load edits:\n " + Joiner.on("\n ").join(editStreams));  StartupProgress prog = NameNode.getStartupProgress();  prog.beginPhase(Phase.LOADING\_EDITS);    long prevLastAppliedTxId = lastAppliedTxId;  long remainingReadTxns = maxTxnsToRead;  try {  // 创建 FSEditLogLoader  FSEditLogLoader loader = new FSEditLogLoader(target, lastAppliedTxId);    // Load latest edits  for (EditLogInputStream editIn : editStreams) {  LogAction logAction = loadEditLogHelper.record();  if (logAction.shouldLog()) {  String logSuppressed = "";  if (logAction.getCount() > 1) {  logSuppressed = "; suppressed logging for " +  (logAction.getCount() - 1) + " edit reads";  }  LOG.info("Reading " + editIn + " expecting start txid #" +  (lastAppliedTxId + 1) + logSuppressed);  }  try {  // 执行  remainingReadTxns -= loader.loadFSEdits(editIn, lastAppliedTxId + 1,  remainingReadTxns, startOpt, recovery);  } finally {  // Update lastAppliedTxId even in case of error, since some ops may  // have been successfully applied before the error.  lastAppliedTxId = loader.getLastAppliedTxId();  }  // If we are in recovery mode, we may have skipped over some txids.  if (editIn.getLastTxId() != HdfsServerConstants.INVALID\_TXID  && recovery != null) {  lastAppliedTxId = editIn.getLastTxId();  }  if (remainingReadTxns <= 0) {  break;  }  }  } finally {  FSEditLog.closeAllStreams(editStreams);  }  prog.endPhase(Phase.LOADING\_EDITS);  return lastAppliedTxId - prevLastAppliedTxId;  } |

###### 2.5.2.2.1 创建 FSEditLogLoader

|  |
| --- |
| public FSEditLogLoader(FSNamesystem fsNamesys, long lastAppliedTxId) {  // 往下追  this(fsNamesys, lastAppliedTxId, new Timer());  } |

|  |
| --- |
| @VisibleForTesting  FSEditLogLoader(FSNamesystem fsNamesys, long lastAppliedTxId, Timer timer) {  this.fsNamesys = fsNamesys;  this.blockManager = fsNamesys.getBlockManager();  this.lastAppliedTxId = lastAppliedTxId;  this.timer = timer;  } |

###### 2.5.2.2.2 执行拉取

|  |
| --- |
| /\*\*  \* Load an edit log, and apply the changes to the in-memory structure  \* This is where we apply edits that we've been writing to disk all  \* along.  \*/  long loadFSEdits(EditLogInputStream edits, long expectedStartingTxId,  long maxTxnsToRead,  StartupOption startOpt, MetaRecoveryContext recovery) throws IOException {  StartupProgress prog = NameNode.getStartupProgress();  Step step = createStartupProgressStep(edits);  prog.beginStep(Phase.LOADING\_EDITS, step);  fsNamesys.writeLock();  try {  long startTime = timer.monotonicNow();  LogAction preLogAction = loadEditsLogHelper.record("pre", startTime);  if (preLogAction.shouldLog()) {  FSImage.LOG.info("Start loading edits file " + edits.getName()  + " maxTxnsToRead = " + maxTxnsToRead +  LogThrottlingHelper.getLogSupressionMessage(preLogAction));  }  // 加载 (返回 EditLog 个数)  long numEdits = loadEditRecords(edits, false, expectedStartingTxId,  maxTxnsToRead, startOpt, recovery);  long endTime = timer.monotonicNow();  LogAction postLogAction = loadEditsLogHelper.record("post", endTime,  numEdits, edits.length(), endTime - startTime);  if (postLogAction.shouldLog()) {  FSImage.LOG.info("Loaded " + postLogAction.getCount()  + " edits file(s) (the last named " + edits.getName()  + ") of total size " + postLogAction.getStats(1).getSum()  + ", total edits " + postLogAction.getStats(0).getSum()  + ", total load time " + postLogAction.getStats(2).getSum()  + " ms");  }  return numEdits;  } finally {  edits.close();  fsNamesys.writeUnlock("loadFSEdits");  prog.endStep(Phase.LOADING\_EDITS, step);  }  } |

|  |
| --- |
| long loadEditRecords(EditLogInputStream in, boolean closeOnExit,  long expectedStartingTxId, long maxTxnsToRead, StartupOption startOpt,  MetaRecoveryContext recovery) throws IOException {  EnumMap<FSEditLogOpCodes, Holder<Integer>> opCounts =  new EnumMap<FSEditLogOpCodes, Holder<Integer>>(FSEditLogOpCodes.class);  if (LOG.isTraceEnabled()) {  LOG.trace("Acquiring write lock to replay edit log");  }  fsNamesys.writeLock();  FSDirectory fsDir = fsNamesys.dir;  fsDir.writeLock();  long recentOpcodeOffsets[] = new long[4];  Arrays.fill(recentOpcodeOffsets, -1);    long expectedTxId = expectedStartingTxId;  long numEdits = 0;  long lastTxId = in.getLastTxId();  long numTxns = (lastTxId - expectedStartingTxId) + 1;  StartupProgress prog = NameNode.getStartupProgress();  Step step = createStartupProgressStep(in);  prog.setTotal(Phase.LOADING\_EDITS, step, numTxns);  Counter counter = prog.getCounter(Phase.LOADING\_EDITS, step);  long lastLogTime = timer.monotonicNow();  long lastInodeId = fsNamesys.dir.getLastInodeId();    try {  while (true) {  try {  FSEditLogOp op;  try {  // 调用 GetJournalEditServlet.doGet() 的输出流 (这里是输入流)  // 真正读取 JournalNode 的 EditLogFile  op = in.readOp();  if (op == null) {  break;  }  } catch (Throwable e) {  // Handle a problem with our input  check203UpgradeFailure(in.getVersion(true), e);  String errorMessage =  formatEditLogReplayError(in, recentOpcodeOffsets, expectedTxId);  FSImage.LOG.error(errorMessage, e);  if (recovery == null) {  // We will only try to skip over problematic opcodes when in  // recovery mode.  throw new EditLogInputException(errorMessage, e, numEdits);  }  MetaRecoveryContext.editLogLoaderPrompt(  "We failed to read txId " + expectedTxId,  recovery, "skipping the bad section in the log");  in.resync();  continue;  }  recentOpcodeOffsets[(int)(numEdits % recentOpcodeOffsets.length)] =  in.getPosition();  if (op.hasTransactionId()) {  if (op.getTransactionId() > expectedTxId) {  MetaRecoveryContext.editLogLoaderPrompt("There appears " +  "to be a gap in the edit log. We expected txid " +  expectedTxId + ", but got txid " +  op.getTransactionId() + ".", recovery, "ignoring missing " +  " transaction IDs");  } else if (op.getTransactionId() < expectedTxId) {  MetaRecoveryContext.editLogLoaderPrompt("There appears " +  "to be an out-of-order edit in the edit log. We " +  "expected txid " + expectedTxId + ", but got txid " +  op.getTransactionId() + ".", recovery,  "skipping the out-of-order edit");  continue;  }  }  try {  if (LOG.isTraceEnabled()) {  LOG.trace("op=" + op + ", startOpt=" + startOpt  + ", numEdits=" + numEdits + ", totalEdits=" + totalEdits);  }  // 将读取的 EditLog 更新到内存  long inodeId = applyEditLogOp(op, fsDir, startOpt,  in.getVersion(true), lastInodeId);  if (lastInodeId < inodeId) {  lastInodeId = inodeId;  }  } catch (RollingUpgradeOp.RollbackException e) {  throw e;  } catch (Throwable e) {  LOG.error("Encountered exception on operation " + op, e);  if (recovery == null) {  throw e instanceof IOException? (IOException)e: new IOException(e);  }  MetaRecoveryContext.editLogLoaderPrompt("Failed to " +  "apply edit log operation " + op + ": error " +  e.getMessage(), recovery, "applying edits");  }  // Now that the operation has been successfully decoded and  // applied, update our bookkeeping.  incrOpCount(op.opCode, opCounts, step, counter);  if (op.hasTransactionId()) {  lastAppliedTxId = op.getTransactionId();  expectedTxId = lastAppliedTxId + 1;  } else {  expectedTxId = lastAppliedTxId = expectedStartingTxId;  }  // log progress  if (op.hasTransactionId()) {  long now = timer.monotonicNow();  if (now - lastLogTime > REPLAY\_TRANSACTION\_LOG\_INTERVAL) {  long deltaTxId = lastAppliedTxId - expectedStartingTxId + 1;  int percent = Math.round((float) deltaTxId / numTxns \* 100);  LOG.info("replaying edit log: " + deltaTxId + "/" + numTxns  + " transactions completed. (" + percent + "%)");  lastLogTime = now;  }  }  numEdits++;  totalEdits++;  if(numEdits >= maxTxnsToRead) {  break;  }  } catch (RollingUpgradeOp.RollbackException e) {  LOG.info("Stopped at OP\_START\_ROLLING\_UPGRADE for rollback.");  break;  } catch (MetaRecoveryContext.RequestStopException e) {  MetaRecoveryContext.LOG.warn("Stopped reading edit log at " +  in.getPosition() + "/" + in.length());  break;  }  }  } finally {  fsNamesys.dir.resetLastInodeId(lastInodeId);  if(closeOnExit) {  in.close();  }  fsDir.writeUnlock();  fsNamesys.writeUnlock("loadEditRecords");  if (LOG.isTraceEnabled()) {  LOG.trace("replaying edit log finished");  }  if (FSImage.LOG.isDebugEnabled()) {  dumpOpCounts(opCounts);  FSImage.LOG.debug("maxTxnsToRead = " + maxTxnsToRead  + " actual edits read = " + numEdits);  }  assert numEdits <= maxTxnsToRead || numEdits == 1 :  "should read at least one txn, but not more than the configured max";  }  return numEdits;  } |

2.5.2.2.2.1 调用 GetJournalEditServlet.doGet()

|  |
| --- |
| @Override  public void doGet(final HttpServletRequest request,  final HttpServletResponse response) throws ServletException, IOException {  FileInputStream editFileIn = null;  try {  final ServletContext context = getServletContext();  final Configuration conf = (Configuration) getServletContext()  .getAttribute(JspHelper.CURRENT\_CONF);  final String journalId = request.getParameter(JOURNAL\_ID\_PARAM);  final String inProgressOkStr = request.getParameter(IN\_PROGRESS\_OK);  final boolean inProgressOk;  if (inProgressOkStr != null &&  inProgressOkStr.equalsIgnoreCase("false")) {  inProgressOk = false;  } else {  inProgressOk = true;  }  QuorumJournalManager.checkJournalId(journalId);  // 获取 JNStorage  final JNStorage storage = JournalNodeHttpServer  .getJournalFromContext(context, journalId).getStorage();  // Check security  if (!checkRequestorOrSendError(conf, request, response)) {  return;  }  // Check that the namespace info is correct  if (!checkStorageInfoOrSendError(storage, request, response)) {  return;  }    long segmentTxId = ServletUtil.parseLongParam(request,  SEGMENT\_TXID\_PARAM);  // 获取 FileJournalManager  FileJournalManager fjm = storage.getJournalManager();  File editFile;  synchronized (fjm) {  // Synchronize on the FJM so that the file doesn't get finalized  // out from underneath us while we're in the process of opening  // it up.  // 获取 EditLogFile  EditLogFile elf = fjm.getLogFile(segmentTxId, inProgressOk);  if (elf == null) {  response.sendError(HttpServletResponse.SC\_NOT\_FOUND,  "No edit log found starting at txid " + segmentTxId);  return;  }  editFile = elf.getFile();  ImageServlet.setVerificationHeadersForGet(response, editFile);  ImageServlet.setFileNameHeaders(response, editFile);  // 构建输入流读取 EditLog  editFileIn = new FileInputStream(editFile);  }  // null  DataTransferThrottler throttler = ImageServlet.getThrottler(conf);  // send edits  // 发送 EditLog (流拷贝)  TransferFsImage.copyFileToStream(response.getOutputStream(), editFile,  editFileIn, throttler);  } catch (Throwable t) {  String errMsg = "getedit failed. " + StringUtils.stringifyException(t);  response.sendError(HttpServletResponse.SC\_INTERNAL\_SERVER\_ERROR, errMsg);  throw new IOException(errMsg);  } finally {  IOUtils.closeStream(editFileIn);  }  } |

2.5.2.2.2.2 将读取的 EditLog 更新到内存

|  |
| --- |
| @SuppressWarnings("deprecation")  private long applyEditLogOp(FSEditLogOp op, FSDirectory fsDir,  StartupOption startOpt, int logVersion, long lastInodeId) throws IOException {  long inodeId = HdfsConstants.GRANDFATHER\_INODE\_ID;  if (LOG.isTraceEnabled()) {  LOG.trace("replaying edit log: " + op);  }  final boolean toAddRetryCache = fsNamesys.hasRetryCache() && op.hasRpcIds();  switch (op.opCode) {  case OP\_ADD: {  AddCloseOp addCloseOp = (AddCloseOp)op;  final String path =  renameReservedPathsOnUpgrade(addCloseOp.path, logVersion);  if (FSNamesystem.LOG.isDebugEnabled()) {  FSNamesystem.LOG.debug(op.opCode + ": " + path +  " numblocks : " + addCloseOp.blocks.length +  " clientHolder " + addCloseOp.clientName +  " clientMachine " + addCloseOp.clientMachine);  }  // There are 3 cases here:  // 1. OP\_ADD to create a new file  // 2. OP\_ADD to update file blocks  // 3. OP\_ADD to open file for append (old append)  // See if the file already exists (persistBlocks call)  INodesInPath iip = fsDir.getINodesInPath(path, DirOp.WRITE);  INodeFile oldFile = INodeFile.valueOf(iip.getLastINode(), path, true);  if (oldFile != null && addCloseOp.overwrite) {  // This is OP\_ADD with overwrite  FSDirDeleteOp.deleteForEditLog(fsDir, iip, addCloseOp.mtime);  iip = INodesInPath.replace(iip, iip.length() - 1, null);  oldFile = null;  }  INodeFile newFile = oldFile;  if (oldFile == null) { // this is OP\_ADD on a new file (case 1)  // versions > 0 support per file replication  // get name and replication  final short replication = fsNamesys.getBlockManager()  .adjustReplication(addCloseOp.replication);  assert addCloseOp.blocks.length == 0;  // add to the file tree  inodeId = getAndUpdateLastInodeId(addCloseOp.inodeId, logVersion, lastInodeId);  newFile = FSDirWriteFileOp.addFileForEditLog(fsDir, inodeId,  iip.getExistingINodes(), iip.getLastLocalName(),  addCloseOp.permissions, addCloseOp.aclEntries,  addCloseOp.xAttrs, replication, addCloseOp.mtime,  addCloseOp.atime, addCloseOp.blockSize, true,  addCloseOp.clientName, addCloseOp.clientMachine,  addCloseOp.storagePolicyId, addCloseOp.erasureCodingPolicyId);  assert newFile != null;  iip = INodesInPath.replace(iip, iip.length() - 1, newFile);  fsNamesys.leaseManager.addLease(addCloseOp.clientName, newFile.getId());  // add the op into retry cache if necessary  if (toAddRetryCache) {  HdfsFileStatus stat =  FSDirStatAndListingOp.createFileStatusForEditLog(fsDir, iip);  fsNamesys.addCacheEntryWithPayload(addCloseOp.rpcClientId,  addCloseOp.rpcCallId, stat);  }  } else { // This is OP\_ADD on an existing file (old append)  if (!oldFile.isUnderConstruction()) {  // This is case 3: a call to append() on an already-closed file.  if (FSNamesystem.LOG.isDebugEnabled()) {  FSNamesystem.LOG.debug("Reopening an already-closed file " +  "for append");  }  LocatedBlock lb = FSDirAppendOp.prepareFileForAppend(fsNamesys, iip,  addCloseOp.clientName, addCloseOp.clientMachine, false, false,  false);  // add the op into retry cache if necessary  if (toAddRetryCache) {  HdfsFileStatus stat =  FSDirStatAndListingOp.createFileStatusForEditLog(fsDir, iip);  fsNamesys.addCacheEntryWithPayload(addCloseOp.rpcClientId,  addCloseOp.rpcCallId, new LastBlockWithStatus(lb, stat));  }  }  }  // Fall-through for case 2.  // Regardless of whether it's a new file or an updated file,  // update the block list.    // Update the salient file attributes.  newFile.setAccessTime(addCloseOp.atime, Snapshot.CURRENT\_STATE\_ID, false);  newFile.setModificationTime(addCloseOp.mtime, Snapshot.CURRENT\_STATE\_ID);  ErasureCodingPolicy ecPolicy =  FSDirErasureCodingOp.unprotectedGetErasureCodingPolicy(  fsDir.getFSNamesystem(), iip);  updateBlocks(fsDir, addCloseOp, iip, newFile, ecPolicy);  break;  }  ......  case OP\_MKDIR: {  // 强制转换  MkdirOp mkdirOp = (MkdirOp)op;  inodeId = getAndUpdateLastInodeId(mkdirOp.inodeId, logVersion,  lastInodeId);  // 往下追  FSDirMkdirOp.mkdirForEditLog(fsDir, inodeId,  renameReservedPathsOnUpgrade(mkdirOp.path, logVersion),  mkdirOp.permissions, mkdirOp.aclEntries, mkdirOp.timestamp);  break;  }  case OP\_SET\_GENSTAMP\_V1: {  SetGenstampV1Op setGenstampV1Op = (SetGenstampV1Op)op;  blockManager.getBlockIdManager().setLegacyGenerationStamp(  setGenstampV1Op.genStampV1);  break;  }  ......  default:  throw new IOException("Invalid operation read " + op.opCode);  }  return inodeId;  } |

|  |
| --- |
| static void mkdirForEditLog(FSDirectory fsd, long inodeId, String src,  PermissionStatus permissions, List<AclEntry> aclEntries, long timestamp)  throws QuotaExceededException, UnresolvedLinkException, AclException,  FileAlreadyExistsException, ParentNotDirectoryException,  AccessControlException {  assert fsd.hasWriteLock();  //  INodesInPath iip = fsd.getINodesInPath(src, DirOp.WRITE\_LINK);  final byte[] localName = iip.getLastLocalName();  final INodesInPath existing = iip.getParentINodesInPath();  Preconditions.checkState(existing.getLastINode() != null);  // 往下追 (Standby NameNode 添加目录信息跟 Active NameNode 相同)  unprotectedMkdir(fsd, inodeId, existing, localName, permissions, aclEntries,  timestamp);  } |

|  |
| --- |
| /\*\*  \* create a directory at path specified by parent  \*/  private static INodesInPath unprotectedMkdir(FSDirectory fsd, long inodeId,  INodesInPath parent, byte[] name, PermissionStatus permission,  List<AclEntry> aclEntries, long timestamp)  throws QuotaExceededException, AclException, FileAlreadyExistsException {  assert fsd.hasWriteLock();  assert parent.getLastINode() != null;  if (!parent.getLastINode().isDirectory()) {  throw new FileAlreadyExistsException("Parent path is not a directory: " +  parent.getPath() + " " + DFSUtil.bytes2String(name));  }  // 创建 INodeDirectory  final INodeDirectory dir = new INodeDirectory(inodeId, name, permission,  timestamp);  // 往父目录添加子目录 返回子目录信息  INodesInPath iip =  fsd.addLastINode(parent, dir, permission.getPermission(), true);  if (iip != null && aclEntries != null) {  AclStorage.updateINodeAcl(dir, aclEntries, Snapshot.CURRENT\_STATE\_ID);  }  return iip;  } |

### 2.6 Standby NameNode 执行 checkpoint 并上传 Active NameNode

入口类：StandbyCheckpointer.CheckpointerThread.run()

|  |
| --- |
| @Override  public void run() {  // We have to make sure we're logged in as far as JAAS  // is concerned, in order to use kerberized SSL properly.  SecurityUtil.doAsLoginUserOrFatal(  new PrivilegedAction<Object>() {  @Override  public Object run() {  // 执行  doWork();  return null;  }  });  } |

|  |
| --- |
| private void doWork() {  // 检查 checkpoint 间隔 默认 60 \* 1000L  final long checkPeriod = 1000 \* checkpointConf.getCheckPeriod();  // Reset checkpoint time so that we don't always checkpoint  // on startup.  lastCheckpointTime = monotonicNow();  lastUploadTime = monotonicNow();  while (shouldRun) {  // 判断是否 checkpoint fsimage  boolean needRollbackCheckpoint = namesystem.isNeedRollbackFsImage();  if (!needRollbackCheckpoint) {  try {  // 睡眠 60s  Thread.sleep(checkPeriod);  } catch (InterruptedException ie) {  }  if (!shouldRun) {  break;  }  }  try {  // We may have lost our ticket since last checkpoint, log in again, just in case  if (UserGroupInformation.isSecurityEnabled()) {  UserGroupInformation.getCurrentUser().checkTGTAndReloginFromKeytab();  }  final long now = monotonicNow();  // 获取没有 checkpoint tx 总数  final long uncheckpointed = countUncheckpointedTxns();  final long secsSinceLast = (now - lastCheckpointTime) / 1000;  // if we need a rollback checkpoint, always attempt to checkpoint  boolean needCheckpoint = needRollbackCheckpoint;  if (needCheckpoint) {  LOG.info("Triggering a rollback fsimage for rolling upgrade.");  } else if (  // uncheckpointed >= 100w  uncheckpointed >= checkpointConf.getTxnCount()  ) {  LOG.info("Triggering checkpoint because there have been {} txns " +  "since the last checkpoint, " +  "which exceeds the configured threshold {}",  uncheckpointed, checkpointConf.getTxnCount());  needCheckpoint = true;  } else if (  // secsSinceLast >= 3600s = 1h  secsSinceLast >= checkpointConf.getPeriod()  ) {  LOG.info("Triggering checkpoint because it has been {} seconds " +  "since the last checkpoint, which exceeds the configured " +  "interval {}", secsSinceLast, checkpointConf.getPeriod());  needCheckpoint = true;  }  if (needCheckpoint) {  synchronized (cancelLock) {  if (now < preventCheckpointsUntil) {  LOG.info("But skipping this checkpoint since we are about to failover!");  canceledCount++;  continue;  }  assert canceler == null;  canceler = new Canceler();  }  // on all nodes, we build the checkpoint. However, we only ship the checkpoint if have a  // rollback request, are the checkpointer, are outside the quiet period.  final long secsSinceLastUpload = (now - lastUploadTime) / 1000;  boolean sendRequest =  // true  isPrimaryCheckPointer  ||  // secsSinceLastUpload >= 3600 \* 15  secsSinceLastUpload >= checkpointConf.getQuietPeriod();  // 执行 checkpoint  doCheckpoint(sendRequest);  // reset needRollbackCheckpoint to false only when we finish a ckpt  // for rollback image  if (needRollbackCheckpoint  && namesystem.getFSImage().hasRollbackFSImage()) {  namesystem.setCreatedRollbackImages(true);  namesystem.setNeedRollbackFsImage(false);  }  lastCheckpointTime = now;  LOG.info("Checkpoint finished successfully.");  }  } catch (SaveNamespaceCancelledException ce) {  LOG.info("Checkpoint was cancelled: {}", ce.getMessage());  canceledCount++;  } catch (InterruptedException ie) {  LOG.info("Interrupted during checkpointing", ie);  // Probably requested shutdown.  continue;  } catch (Throwable t) {  LOG.error("Exception in doCheckpoint", t);  } finally {  synchronized (cancelLock) {  canceler = null;  }  }  }  } |

#### 2.6.1 执行 checkpoint

|  |
| --- |
| private void doCheckpoint(boolean sendCheckpoint) throws InterruptedException, IOException {  assert canceler != null;  final long txid;  final NameNodeFile imageType;  // Acquire cpLock to make sure no one is modifying the name system.  // It does not need the full namesystem write lock, since the only thing  // that modifies namesystem on standby node is edit log replaying.  namesystem.cpLockInterruptibly();  try {  assert namesystem.getEditLog().isOpenForRead() :  "Standby Checkpointer should only attempt a checkpoint when " +  "NN is in standby mode, but the edit logs are in an unexpected state";  // 获取 FSImage  FSImage img = namesystem.getFSImage();  long prevCheckpointTxId = img.getStorage().getMostRecentCheckpointTxId();  long thisCheckpointTxId = img.getCorrectLastAppliedOrWrittenTxId();  assert thisCheckpointTxId >= prevCheckpointTxId;  if (thisCheckpointTxId == prevCheckpointTxId) {  LOG.info("A checkpoint was triggered but the Standby Node has not " +  "received any transactions since the last checkpoint at txid {}. " +  "Skipping...", thisCheckpointTxId);  return;  }  if (namesystem.isRollingUpgrade()  && !namesystem.getFSImage().hasRollbackFSImage()) {  // if we will do rolling upgrade but have not created the rollback image  // yet, name this checkpoint as fsimage\_rollback  imageType = NameNodeFile.IMAGE\_ROLLBACK;  } else {  imageType = NameNodeFile.IMAGE;  }  // 将 fsimage+editLog元数据写一份到磁盘 fsimage[合并结果] (这个 fsimage 将来上传给 Active NameNode)  img.saveNamespace(namesystem, imageType, canceler);  txid = img.getStorage().getMostRecentCheckpointTxId();  assert txid == thisCheckpointTxId : "expected to save checkpoint at txid=" +  thisCheckpointTxId + " but instead saved at txid=" + txid;  // Save the legacy OIV image, if the output dir is defined.  String outputDir = checkpointConf.getLegacyOivImageDir();  if (outputDir != null && !outputDir.isEmpty()) {  try {  img.saveLegacyOIVImage(namesystem, outputDir, canceler);  } catch (IOException ioe) {  LOG.warn("Exception encountered while saving legacy OIV image; "  + "continuing with other checkpointing steps", ioe);  }  }  } finally {  namesystem.cpUnlock();  }  //early exit if we shouldn't actually send the checkpoint to the ANN  if (!sendCheckpoint) {  return;  }  // Upload the saved checkpoint back to the active  // Do this in a separate thread to avoid blocking transition to active, but don't allow more  // than the expected number of tasks to run or queue up  // See HDFS-4816  // 创建一个线程池  ExecutorService executor = new ThreadPoolExecutor(  0,  activeNNAddresses.size(),  100,  TimeUnit.MILLISECONDS,  new LinkedBlockingQueue<Runnable>(activeNNAddresses.size()),  uploadThreadFactory);  // for right now, just match the upload to the nn address by convention. There is no need to  // directly tie them together by adding a pair class.  List<Future<TransferFsImage.TransferResult>> uploads =  new ArrayList<Future<TransferFsImage.TransferResult>>();  // 遍历 Active NameNode  for (final URL activeNNAddress : activeNNAddresses) {  Future<TransferFsImage.TransferResult> upload =  executor.submit(new Callable<TransferFsImage.TransferResult>() {  @Override  public TransferFsImage.TransferResult call()  throws IOException, InterruptedException {  CheckpointFaultInjector.getInstance().duringUploadInProgess();  // 请求 NameNode 上传 fsimage (http)  return TransferFsImage.uploadImageFromStorage(  activeNNAddress, conf, namesystem  .getFSImage().getStorage(),  imageType, txid, canceler);  }  });  uploads.add(upload);  }  InterruptedException ie = null;  IOException ioe = null;  int i = 0;  boolean success = false;  for (; i < uploads.size(); i++) {  // 获取请求结果  Future<TransferFsImage.TransferResult> upload = uploads.get(i);  try {  // TODO should there be some smarts here about retries nodes that are not the active NN?  if (upload.get() == TransferFsImage.TransferResult.SUCCESS) {  success = true;  //avoid getting the rest of the results - we don't care since we had a successful upload  break;  }  } catch (ExecutionException e) {  ioe = new IOException("Exception during image upload", e);  break;  } catch (InterruptedException e) {  ie = e;  break;  }  }  if (ie == null && ioe == null) {  //Update only when response from remote about success or  lastUploadTime = monotonicNow();  // we are primary if we successfully updated the ANN  this.isPrimaryCheckPointer = success;  }  // cleaner than copying code for multiple catch statements and better than catching all  // exceptions, so we just handle the ones we expect.  if (ie != null || ioe != null) {  // cancel the rest of the tasks, and close the pool  for (; i < uploads.size(); i++) {  Future<TransferFsImage.TransferResult> upload = uploads.get(i);  // The background thread may be blocked waiting in the throttler, so  // interrupt it.  upload.cancel(true);  }  // shutdown so we interrupt anything running and don't start anything new  executor.shutdownNow();  // this is a good bit longer than the thread timeout, just to make sure all the threads  // that are not doing any work also stop  executor.awaitTermination(500, TimeUnit.MILLISECONDS);  // re-throw the exception we got, since one of these two must be non-null  if (ie != null) {  throw ie;  } else if (ioe != null) {  throw ioe;  }  }  } |

##### 2.6.1.1 请求 NameNode 上传 fsimage (http)

|  |
| --- |
| /\*\*  \* Requests that the NameNode download an image from this node. Allows for  \* optional external cancelation.  \*  \* @param fsName the http address for the remote NN  \* @param conf Configuration  \* @param storage the storage directory to transfer the image from  \* @param nnf the NameNodeFile type of the image  \* @param txid the transaction ID of the image to be uploaded  \* @param canceler optional canceler to check for abort of upload  \* @throws IOException if there is an I/O error or cancellation  \*/  public static TransferResult uploadImageFromStorage(URL fsName, Configuration conf,  NNStorage storage, NameNodeFile nnf, long txid, Canceler canceler)  throws IOException {  // 构建 URL (xxx/imagetransfer)  URL url = new URL(fsName, ImageServlet.PATH\_SPEC);  long startTime = Time.monotonicNow();  try {  // 请求上传 fsimage  uploadImage(url, conf, storage, nnf, txid, canceler);  } catch (HttpPutFailedException e) {  // translate the error code to a result, which is a bit more obvious in usage  TransferResult result = TransferResult.getResultForCode(e.getResponseCode());  if (result.shouldReThrowException) {  throw e;  }  return result;  }  double xferSec = Math.max(  ((float) (Time.monotonicNow() - startTime)) / 1000.0, 0.001);  LOG.info("Uploaded image with txid " + txid + " to namenode at " + fsName  + " in " + xferSec + " seconds");  return TransferResult.SUCCESS;  } |

###### 2.6.1.1.1 请求上传 fsimage

|  |
| --- |
| /\*  \* Uploads the imagefile using HTTP PUT method  \*/  private static void uploadImage(URL url, Configuration conf,  NNStorage storage, NameNodeFile nnf, long txId, Canceler canceler)  throws IOException {  // 获取本地 Standby NameNode fsimage[合并结果] 路径文件 (文件已经打开)  File imageFile = storage.findImageFile(nnf, txId);  if (imageFile == null) {  throw new IOException("Could not find image with txid " + txId);  }  HttpURLConnection connection = null;  try {  // 构建 HTTP PUT 请求参数  URIBuilder uriBuilder = new URIBuilder(url.toURI());  // write all params for image upload request as query itself.  // Request body contains the image to be uploaded.  Map<String, String> params = ImageServlet.getParamsForPutImage(storage,  txId, imageFile.length(), nnf);  for (Entry<String, String> entry : params.entrySet()) {  uriBuilder.addParameter(entry.getKey(), entry.getValue());  }  URL urlWithParams = uriBuilder.build().toURL();  // 连接 NameNodeHttpServer 的 ImageServlet 容器  connection = (HttpURLConnection) connectionFactory.openConnection(  urlWithParams, UserGroupInformation.isSecurityEnabled());  // Set the request to PUT  // 请求方式为 PUT  connection.setRequestMethod("PUT");  connection.setDoOutput(true);  // 64 KB  int chunkSize = conf.getInt(  DFSConfigKeys.DFS\_IMAGE\_TRANSFER\_CHUNKSIZE\_KEY,  DFSConfigKeys.DFS\_IMAGE\_TRANSFER\_CHUNKSIZE\_DEFAULT);  if (imageFile.length() > chunkSize) {  // using chunked streaming mode to support upload of 2GB+ files and to  // avoid internal buffering.  // this mode should be used only if more than chunkSize data is present  // to upload. otherwise upload may not happen sometimes.  connection.setChunkedStreamingMode(chunkSize);  }  setTimeout(connection);  // set headers for verification  ImageServlet.setVerificationHeadersForPut(connection, imageFile);  // Write the file to output stream.  // 执行请求 (流拷贝 调用 ImageServlet.doPut())  writeFileToPutRequest(conf, connection, imageFile, canceler);  int responseCode = connection.getResponseCode();  if (responseCode != HttpURLConnection.HTTP\_OK) {  throw new HttpPutFailedException(String.format(  "Image uploading failed, status: %d, url: %s, message: %s",  responseCode, urlWithParams, connection.getResponseMessage()),  responseCode);  }  } catch (AuthenticationException | URISyntaxException e) {  throw new IOException(e);  } finally {  if (connection != null) {  connection.disconnect();  }  }  } |

|  |
| --- |
| private static void writeFileToPutRequest(Configuration conf,  HttpURLConnection connection, File imageFile, Canceler canceler)  throws IOException {  connection.setRequestProperty(Util.CONTENT\_TYPE, "application/octet-stream");  connection.setRequestProperty(Util.CONTENT\_TRANSFER\_ENCODING, "binary");  // 获取 HTTP 输出流  OutputStream output = connection.getOutputStream();  FileInputStream input = new FileInputStream(imageFile);  try {  // 将上传的 fsimage 写入输出流  copyFileToStream(output, imageFile, input,  ImageServlet.getThrottler(conf), canceler);  } finally {  IOUtils.closeStream(input);  IOUtils.closeStream(output);  }  } |

2.6.1.1.1.1 请求调用 ImageServlet.doPut()

|  |
| --- |
| @Override  protected void doPut(final HttpServletRequest request,  final HttpServletResponse response) throws ServletException, IOException {  try {  ServletContext context = getServletContext();  // 获取 Active NameNode FSImage  final FSImage nnImage = NameNodeHttpServer.getFsImageFromContext(context);  final Configuration conf = (Configuration) getServletContext()  .getAttribute(JspHelper.CURRENT\_CONF);  // 解析请求参数  final PutImageParams parsedParams = new PutImageParams(request, response,  conf);  final NameNodeMetrics metrics = NameNode.getNameNodeMetrics();  validateRequest(context, conf, request, response, nnImage,  parsedParams.getStorageInfoString());  UserGroupInformation.getCurrentUser().doAs(  new PrivilegedExceptionAction<Void>() {  @Override  public Void run() throws Exception {  // if its not the active NN, then we need to notify the caller it was was the wrong  // target (regardless of the fact that we got the image)  HAServiceProtocol.HAServiceState state = NameNodeHttpServer  .getNameNodeStateFromContext(getServletContext());  if (state != HAServiceProtocol.HAServiceState.ACTIVE) {  // we need a different response type here so the client can differentiate this  // from the failure to upload due to (1) security, or (2) other checkpoints already  // present  response.sendError(HttpServletResponse.SC\_EXPECTATION\_FAILED,  "Nameode "+request.getLocalAddr()+" is currently not in a state which can "  + "accept uploads of new fsimages. State: "+state);  return null;  }  final long txid = parsedParams.getTxId();  String remoteAddr = request.getRemoteAddr();  ImageUploadRequest imageRequest = new ImageUploadRequest(txid, remoteAddr);  final NameNodeFile nnf = parsedParams.getNameNodeFile();  // if the node is attempting to upload an older transaction, we ignore it  SortedSet<ImageUploadRequest> larger = currentlyDownloadingCheckpoints.tailSet(imageRequest);  if (larger.size() > 0) {  response.sendError(HttpServletResponse.SC\_CONFLICT,  "Another checkpointer is already in the process of uploading a" +  " checkpoint made up to transaction ID " + larger.last());  return null;  }  //make sure no one else has started uploading one  if (!currentlyDownloadingCheckpoints.add(imageRequest)) {  response.sendError(HttpServletResponse.SC\_CONFLICT,  "Either current namenode is checkpointing or another"  + " checkpointer is already in the process of "  + "uploading a checkpoint made at transaction ID "  + txid);  return null;  }  try {  if (nnImage.getStorage().findImageFile(nnf, txid) != null) {  response.sendError(HttpServletResponse.SC\_CONFLICT,  "Either current namenode has checkpointed or "  + "another checkpointer already uploaded an "  + "checkpoint for txid " + txid);  return null;  }  InputStream stream = request.getInputStream();  try {  long start = monotonicNow();  MD5Hash downloadImageDigest = TransferFsImage  // 处理 Standby NameNode 上传 fsimage 请求  .handleUploadImageRequest(  request,  txid,  nnImage.getStorage(),  stream,  parsedParams.getFileSize(), getThrottler(conf));  // fsimage 文件重命名  nnImage.saveDigestAndRenameCheckpointImage(nnf, txid,  downloadImageDigest);  // Metrics non-null only when used inside name node  if (metrics != null) {  long elapsed = monotonicNow() - start;  metrics.addPutImage(elapsed);  }  // Now that we have a new checkpoint, we might be able to  // remove some old ones.  nnImage.purgeOldStorage(nnf);  } finally {  // remove the request once we've processed it, or it threw an error, so we  // aren't using it either  currentlyDownloadingCheckpoints.remove(imageRequest);  stream.close();  }  } finally {  nnImage.removeFromCheckpointing(txid);  }  return null;  }  });  } catch (Throwable t) {  String errMsg = "PutImage failed. " + StringUtils.stringifyException(t);  response.sendError(HttpServletResponse.SC\_GONE, errMsg);  throw new IOException(errMsg);  }  } |

2.6.1.1.1.1.1 请求上传 fsimage 落盘到 Active NameNode 磁盘

|  |
| --- |
| static MD5Hash handleUploadImageRequest(HttpServletRequest request,  long imageTxId, Storage dstStorage, InputStream stream,  long advertisedSize, DataTransferThrottler throttler) throws IOException {  // 获取 image file  String fileName = NNStorage.getCheckpointImageFileName(imageTxId);  // 找到要存储 image files  List<File> dstFiles = dstStorage.getFiles(NameNodeDirType.IMAGE, fileName);  if (dstFiles.isEmpty()) {  throw new IOException("No targets in destination storage!");  }  MD5Hash advertisedDigest = parseMD5Header(request);  // 往下追  MD5Hash hash = Util.receiveFile(fileName,  dstFiles, dstStorage,  true,  advertisedSize, advertisedDigest,  fileName, stream, throttler);  LOG.info("Downloaded file " + dstFiles.get(0).getName() + " size "  + dstFiles.get(0).length() + " bytes.");  return hash;  } |

|  |
| --- |
| /\*\*  \* Receives file at the url location from the input stream and puts them in  \* the specified destination storage location.  \*/  public static MD5Hash receiveFile(String url, List<File> localPaths,  Storage dstStorage, boolean getChecksum, long advertisedSize,  MD5Hash advertisedDigest, String fsImageName, InputStream stream,  DataTransferThrottler throttler) throws  IOException {  long startTime = Time.monotonicNow();  Map<FileOutputStream, File> streamPathMap = new HashMap<>();  StringBuilder xferStats = new StringBuilder();  double xferCombined = 0;  if (localPaths != null) {  // If the local paths refer to directories, use the server-provided header  // as the filename within that directory  List<File> newLocalPaths = new ArrayList<>();  for (File localPath : localPaths) {  if (localPath.isDirectory()) {  if (fsImageName == null) {  throw new IOException("No filename header provided by server");  }  newLocalPaths.add(new File(localPath, fsImageName));  } else {  newLocalPaths.add(localPath);  }  }  localPaths = newLocalPaths;  }  long received = 0;  MessageDigest digester = null;  if (getChecksum) {  digester = MD5Hash.getDigester();  stream = new DigestInputStream(stream, digester);  }  boolean finishedReceiving = false;  int num = 1;  List<FileOutputStream> outputStreams = Lists.newArrayList();  try {  if (localPaths != null) {  for (File f : localPaths) {  try {  if (f.exists()) {  LOG.warn("Overwriting existing file " + f  + " with file downloaded from " + url);  }  FileOutputStream fos = new FileOutputStream(f);  outputStreams.add(fos);  streamPathMap.put(fos, f);  } catch (IOException ioe) {  LOG.warn("Unable to download file " + f, ioe);  // This will be null if we're downloading the fsimage to a file  // outside of an NNStorage directory.  if (dstStorage != null &&  (dstStorage instanceof StorageErrorReporter)) {  ((StorageErrorReporter) dstStorage).reportErrorOnFile(f);  }  }  }  if (outputStreams.isEmpty()) {  throw new IOException(  "Unable to download to any storage directory");  }  }  // 4096  byte[] buf = new byte[IO\_FILE\_BUFFER\_SIZE];  while (num > 0) {  // 读取 image  num = stream.read(buf);  if (num > 0) {  received += num;  for (FileOutputStream fos : outputStreams) {  // 写入 outputStreams  fos.write(buf, 0, num);  }  if (throttler != null) {  throttler.throttle(num);  }  }  }  finishedReceiving = true;  double xferSec = Math.max(  ((float) (Time.monotonicNow() - startTime)) / 1000.0, 0.001);  long xferKb = received / 1024;  xferCombined += xferSec;  xferStats.append(  String.format(" The file download took %.2fs at %.2f KB/s.",  xferSec, xferKb / xferSec));  } finally {  stream.close();  for (FileOutputStream fos : outputStreams) {  long flushStartTime = Time.monotonicNow();  fos.getChannel().force(true);  fos.close();  double writeSec = Math.max(((float)  (flushStartTime - Time.monotonicNow())) / 1000.0, 0.001);  xferCombined += writeSec;  xferStats.append(String  .format(" Synchronous (fsync) write to disk of " +  streamPathMap.get(fos).getAbsolutePath() +  " took %.2fs.", writeSec));  }  // Something went wrong and did not finish reading.  // Remove the temporary files.  if (!finishedReceiving) {  deleteTmpFiles(localPaths);  }  if (finishedReceiving && received != advertisedSize) {  // only throw this exception if we think we read all of it on our end  // -- otherwise a client-side IOException would be masked by this  // exception that makes it look like a server-side problem!  deleteTmpFiles(localPaths);  throw new IOException("File " + url + " received length " + received +  " is not of the advertised size " + advertisedSize +  ". Fsimage name: " + fsImageName + " lastReceived: " + num);  }  }  xferStats.insert(0, String.format("Combined time for file download and" +  " fsync to all disks took %.2fs.", xferCombined));  LOG.info(xferStats.toString());  if (digester != null) {  MD5Hash computedDigest = new MD5Hash(digester.digest());  if (advertisedDigest != null &&  !computedDigest.equals(advertisedDigest)) {  deleteTmpFiles(localPaths);  throw new IOException("File " + url + " computed digest " +  computedDigest + " does not match advertised digest " +  advertisedDigest);  }  return computedDigest;  } else {  return null;  }  } |

2.6.1.1.1.1.2 Active NameNode fsimage 文件重命名

|  |
| --- |
| /\*\*  \* This is called by the 2NN after having downloaded an image, and by  \* the NN after having received a new image from the 2NN. It  \* renames the image from fsimage\_N.ckpt to fsimage\_N and also  \* saves the related .md5 file into place.  \*/  public synchronized void saveDigestAndRenameCheckpointImage(NameNodeFile nnf,  long txid, MD5Hash digest) throws IOException {  // Write and rename MD5 file  List<StorageDirectory> badSds = Lists.newArrayList();    for (StorageDirectory sd : storage.dirIterable(NameNodeDirType.IMAGE)) {  File imageFile = NNStorage.getImageFile(sd, nnf, txid);  try {  MD5FileUtils.saveMD5File(imageFile, digest);  } catch (IOException ioe) {  badSds.add(sd);  }  }  storage.reportErrorsOnDirectories(badSds);    CheckpointFaultInjector.getInstance().afterMD5Rename();    // Rename image from tmp file  // 重命名  renameCheckpoint(txid, NameNodeFile.IMAGE\_NEW, nnf, false);  // So long as this is the newest image available,  // advertise it as such to other checkpointers  // from now on  if (txid > storage.getMostRecentCheckpointTxId()) {  storage.setMostRecentCheckpointInfo(txid, Time.now());  }  // Create a version file in any new storage directory.  initNewDirs();  } |