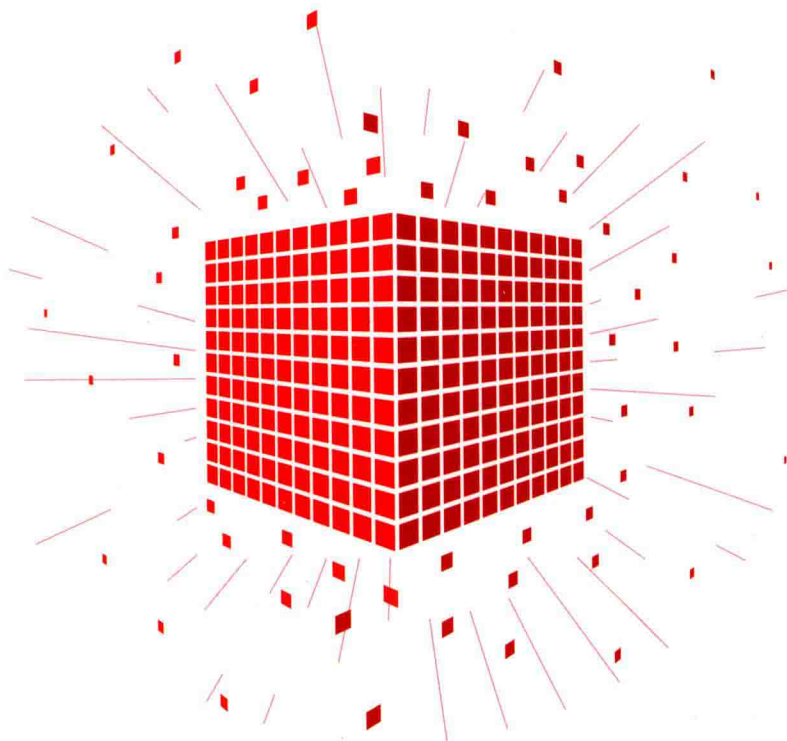




An In-Depth Guide to Hadoop HDFS

深度剖析 Hadoop HDFS

林意群◎编著



机械工业出版社
China Machine Press



技术丛书

深度剖析 Hadoop HDFS

林意群◎编著



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

深度剖析 Hadoop HDFS / 林意群编著. —北京: 机械工业出版社, 2017.3
(大数据技术丛书)

ISBN 978-7-111-56207-8

I. 深… II. 林… III. 数据处理软件 IV. TP274

中国版本图书馆 CIP 数据核字 (2017) 第 040479 号

深度剖析 Hadoop HDFS

出版发行: 机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码: 100037)

责任编辑: 吴 怡

责任校对: 李秋荣

印 刷: 北京诚信伟业印刷有限公司

版 次: 2017 年 4 月第 1 版第 1 次印刷

开 本: 186mm×240mm 1/16

印 张: 21

书 号: ISBN 978-7-111-56207-8

定 价: 79.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88379426 88361066

投稿热线: (010) 88379604

购书热线: (010) 68326294 88379649 68995259

读者信箱: hzit@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问: 北京大成律师事务所 韩光 / 邹晓东

我上大学时，就开始在 CSDN 上写技术博客，目的在于记录平时遇到的一些问题以及研究的技术细节，好在将来可以进行查阅。随着时间的增长，我开始专注于某个技术模块，因为这样可以让我对具体某项技术有更深入的研究，写出的内容也会更加系统化，而 HDFS 就是其中一个我持续研究的技术模块。同时作为一名 Hadoop 社区的活跃贡献者，我也会将社区上一些比较有意思的东西分享到博客上，许多博友给了不少反馈，描述他们在工作中碰到的一些实际问题。在这样不断的写作、交流过程中，我得到了快速成长。目前大数据领域相关的书籍并不是很多，而专门讲解其中一个模块的书则更少，所以我将我过去一年多时间内关于 HDFS 的博客文章进行了整理、改进，同时也加入了一些新的内容。可以说，本书的内容源自博客，但是超越博客。

本书不会是纯源码分析的书籍。首先，我把工作实践中遇到的许多经验写入了书中，第 7 章便属于纯实践型的经验总结。其次，本书会是一个比较“新”的书，这里的“新”并不是指所分析的代码版本新，而是包含了 HDFS 未来的一些比较棒的功能特性，以及 Hadoop 社区目前在做的一些事情。在这本书中，你会看到许多与社区相关的 JIRA，了解如何从社区上找到问题的解决办法。期待本书能给你带来更多的启发。

本书适合具有一定 Java 语言基础的同学，尤其适合以下读者朋友：

- 大数据架构师、开发者、运维工程师。
- 高年级本科生或研究生。
- 热衷于分布式存储技术的爱好者。

本书分为三大部分，“核心设计篇”介绍 HDFS 的基本原理、数据管理与策略等，“细节实现篇”介绍 HDFS 的块处理、流量处理、结构分析等，“解决方案篇”介绍数据管理技术与方案、数据读写技术、异常处理等。

第一部分“核心设计篇”包括内容如下：

第 1 章介绍 HDFS 现有的数据存储方式，主要介绍其中的内存存储和异构存储两个方面。

第2章介绍 HDFS 目前内部几种主要的功能机制，包括缓存管理、快照管理等。

第3章介绍 HDFS 比较新颖的一些功能，以及目前较少被人用到的功能特性。

第二部分“细节实现篇”包括内容如下：

第4章介绍 HDFS 的块处理相关操作，主要处理场景包括块如何组织、上报处理的过程以及多余块的清除。

第5章介绍 HDFS 的流量处理过程，包括 HDFS 目前流量处理的场景以及 Balancer 工具的数据平衡原理和优化。

第6章介绍 HDFS 一些特殊的结构对象类，包括这些类的作用、原理以及运用场景。

第三部分“解决方案篇”包括内容如下：

第7章介绍与 HDFS 相关的多套运维管理的操作方案，包括数据迁移、数据监控等方面。

第8章介绍 HDFS 写磁盘时的一些优化策略和改造方案。

第9章介绍 HDFS 的一些异常场景，并给出了相应的解决方案。

由于笔者水平有限，本书难免会有出错或者介绍不明确的地方，恳请读者批评指正，可以发送关于本书的意见和建议到我的个人邮箱：yqtin@apache.org。本书所涉及的源码，大家可以从 Hadoop 的 Git 地址上进行下载：<https://github.com/apache/hadoop>，其中，不同的分支对应不同版本的代码。相关 Git 地址和 CSDN 博客地址如下：

□ Git 地址：<https://github.com/linyiqun>

□ CSDN 地址：<http://blog.csdn.net/androidlushangderen>

感谢机械工业出版社的吴怡编辑，在我写作的过程中，不断指出其中的不足之处，督促和引导我完成本书的编写。

感谢蘑菇街数据平台部的同事们，在工作中不断地给予我帮助和支持，协助我解决各种各样的问题，于是才有了本书中所展现的精彩内容。

林意群

2017年2月

前言

第一部分 核心设计篇

第 1 章 HDFS 的数据存储	2
1.1 HDFS 内存存储	2
1.1.1 HDFS 内存存储原理	2
1.1.2 Linux 虚拟内存盘	4
1.1.3 HDFS 的内存存储流程分析	4
1.1.4 LAZY_PERSIST 内存存储的使用	14
1.2 HDFS 异构存储	15
1.2.1 异构存储类型	16
1.2.2 异构存储原理	17
1.2.3 块存储类型选择策略	22
1.2.4 块存储策略集合	24
1.2.5 块存储策略的调用	27
1.2.6 HDFS 异构存储策略的不足之处	28
1.2.7 HDFS 存储策略的使用	30
1.3 小结	31

第 2 章 HDFS 的数据管理与策略选择	32
2.1 HDFS 缓存与缓存块	32
2.1.1 HDFS 物理层面缓存块	33
2.1.2 缓存块的生命周期状态	34
2.1.3 CacheBlock、UnCacheBlock 场景触发	36
2.1.4 CacheBlock、UnCacheBlock 缓存块的确定	38
2.1.5 系统持有的缓存块列表如何更新	39
2.1.6 缓存块的使用	40
2.1.7 HDFS 缓存相关配置	40
2.2 HDFS 中心缓存管理	42
2.2.1 HDFS 缓存适用场景	43
2.2.2 HDFS 缓存的结构设计	43
2.2.3 HDFS 缓存管理机制分析	45
2.2.4 HDFS 中心缓存疑问点	55
2.2.5 HDFS CacheAdmin 命令使用	56
2.3 HDFS 快照管理	58
2.3.1 快照概念	59
2.3.2 HDFS 中的快照相关命令	59
2.3.3 HDFS 内部的快照管理机制	60
2.3.4 HDFS 的快照使用	71
2.4 HDFS 副本放置策略	72
2.4.1 副本放置策略概念与方法	72
2.4.2 副本放置策略的有效前提	73
2.4.3 默认副本放置策略的分析	73
2.4.4 目标存储好坏的判断	82
2.4.5 chooseTargets 的调用	83
2.4.6 BlockPlacementPolicyWithNodeGroup 继承类	84
2.4.7 副本放置策略的结果验证	85
2.5 HDFS 内部的认证机制	85
2.5.1 BlockToken 认证	85
2.5.2 HDFS 的 Sasl 认证	91
2.5.3 BlockToken 认证与 HDFS 的 Sasl 认证对比	97

2.6	HDFS 内部的磁盘目录服务	98
2.6.1	HDFS 的三大磁盘目录检测扫描服务	98
2.6.2	DiskChecker: 坏盘检测服务	99
2.6.3	DirectoryScanner: 目录扫描服务	104
2.6.4	VolumeScanner: 磁盘目录扫描服务	110
2.7	小结	116
第 3 章	HDFS 的新颖功能特性	117
3.1	HDFS 视图文件系统: ViewFileSystem	117
3.1.1	ViewFileSystem: 视图文件系统	118
3.1.2	ViewFileSystem 内部实现原理	119
3.1.3	ViewFileSystem 的使用	125
3.2	HDFS 的 Web 文件系统: WebHdfsFileSystem	126
3.2.1	WebHdfsFileSystem 的 REST API 操作	127
3.2.2	WebHdfsFileSystem 的流程调用	129
3.2.3	WebHdfsFileSystem 执行器调用	130
3.2.4	WebHDFS 的 OAuth2 认证	133
3.2.5	WebHDFS 的使用	135
3.3	HDFS 数据加密空间: Encryption zone	136
3.3.1	Encryption zone 原理介绍	136
3.3.2	Encryption zone 源码实现	136
3.3.3	Encryption zone 的使用	144
3.4	HDFS 纠删码技术	145
3.4.1	纠删码概念	145
3.4.2	纠删码技术的优劣势	146
3.4.3	Hadoop 纠删码概述	147
3.4.4	纠删码技术在 Hadoop 中的实现	148
3.5	HDFS 对象存储: Ozone	152
3.5.1	Ozone 介绍	153
3.5.2	Ozone 的高层级设计	154
3.5.3	Ozone 的实现细节	157
3.5.4	Ozone 的使用	157
3.6	小结	158

第二部分 细节实现篇

第 4 章 HDFS 的块处理	160
4.1 HDFS 块检查命令 fsck	160
4.1.1 fsck 参数使用	160
4.1.2 fsck 过程调用	161
4.1.3 fsck 原理分析	162
4.1.4 fsck 使用场景	171
4.2 HDFS 如何检测并删除多余副本块	171
4.2.1 多余副本块以及发生的场景	172
4.2.2 OverReplication 多余副本块处理	172
4.2.3 多余副本块清除的场景调用	177
4.3 HDFS 数据块的汇报与处理	179
4.3.1 块处理的五大类型	179
4.3.2 toAdd: 新添加的块	181
4.3.3 toRemove: 待移除的块	184
4.3.4 toInvalidate: 无效的块	186
4.3.5 toCorrupt: 损坏的块	189
4.3.6 toUC: 正在构建中的块	191
4.4 小结	193
第 5 章 HDFS 的流量处理	194
5.1 HDFS 的内部限流	194
5.1.1 数据的限流	194
5.1.2 DataTransferThrottler 限流原理	196
5.1.3 数据流限流在 Hadoop 中的使用	198
5.1.4 Hadoop 限流优化点	202
5.2 数据平衡	204
5.2.1 Balancer 和 Dispatcher	204
5.2.2 数据不平衡现象	207
5.2.3 Balancer 性能优化	207

5.3	HDFS 节点内数据平衡	210
5.3.1	磁盘间数据不平衡现象及问题	211
5.3.2	传统的磁盘间数据不平衡解决方案	211
5.3.3	社区解决方案: DiskBalancer	212
5.4	小结	216
第 6 章	HDFS 的部分结构分析	217
6.1	HDFS 镜像文件的解析与反解析	217
6.1.1	HDFS 的 FsImage 镜像文件	218
6.1.2	FsImage 的解析	218
6.1.3	FsImage 的反解析	221
6.1.4	HDFS 镜像文件的解析与反解析命令	226
6.2	DataNode 数据处理中心 DataXceiver	227
6.2.1	DataXceiver 的定义和结构	228
6.2.2	DataXceiver 下游处理方法	232
6.2.3	ShortCircuit	232
6.2.4	DataXceiver 的上游调用	233
6.2.5	DataXceiver 与 DataXceiverServer	234
6.3	HDFS 邻近信息块: BlockInfoContiguous	235
6.3.1	triplets 对象数组	236
6.3.2	BlockInfoContiguous 的链表操作	239
6.3.3	块迭代器 BlockIterator	244
6.4	小结	246

第三部分 解决方案篇

第 7 章	HDFS 的数据管理	248
7.1	HDFS 的读写限流方案	248
7.1.1	限流方案实现要点以及可能造成的影响	248
7.1.2	限流方案实现	249
7.1.3	限流测试结果	250

7.2	HDFS 数据资源使用量分析以及趋势预测	250
7.2.1	要获取哪些数据	251
7.2.2	如何获取这些数据	251
7.2.3	怎么用这些数据	254
7.3	HDFS 数据迁移解决方案	257
7.3.1	数据迁移使用场景	257
7.3.2	数据迁移要素考量	258
7.3.3	HDFS 数据迁移解决方案: DistCp	259
7.3.4	DistCp 优势特性	260
7.3.5	Hadoop DistCp 命令	264
7.3.6	DistCp 解决集群间数据迁移实例	265
7.4	DataNode 迁移方案	265
7.4.1	迁移方案的目标	266
7.4.2	DataNode 更换主机名、ip 地址时的迁移方案	267
7.5	HDFS 集群重命名方案	268
7.6	HDFS 的配置管理方案	271
7.6.1	HDFS 配置管理的问题	271
7.6.2	现有配置管理工具	272
7.6.3	运用 Git 来做配置管理	272
7.7	小结	273
第 8 章	HDFS 的数据读写	274
8.1	DataNode 引用计数磁盘选择策略	274
8.1.1	HDFS 现有磁盘选择策略	274
8.1.2	自定义磁盘选择策略	279
8.2	Hadoop 节点“慢磁盘”监控	282
8.2.1	慢磁盘的定义以及如何发现	282
8.2.2	慢磁盘监控	284
8.3	小结	287
第 9 章	HDFS 的异常场景	288
9.1	DataNode 慢启动问题	288

9.1.1	DataNode 慢启动现象	288
9.1.2	代码追踪分析	290
9.1.3	参数可配置化改造	293
9.2	Hadoop 中止下线操作后大量剩余复制块问题	295
9.2.1	节点下线操作的含义及问题	295
9.2.2	死节点“复活”	297
9.2.3	Decommission 下线操作如何运作	299
9.2.4	中止下线操作后移除残余副本块解决方案	303
9.3	DFSOutputStream 的 DataStream 线程泄漏问题	306
9.3.1	DFSOutputStream 写数据过程及周边相关类、变量	306
9.3.2	DataStream 数据流对象	307
9.3.3	ResponseProcessor 回复获取类	311
9.3.4	DataStream 与 DFSOutputStream 的关系	313
9.3.5	Streamer 线程泄漏问题	316
9.4	小结	319
附录	如何向开源社区提交自己的代码	320



第一部分 *Part 1*

核心设计篇

HDFS 的数据存储

本章将从 HDFS 的数据存储开始说起，因为正是先有了数据的存储，才有后续的写入和管理等操作。HDFS 的数据存储包括两块：一块是 HDFS 内存存储，另一块是 HDFS 异构存储。HDFS 内存存储是一种十分特殊的存储方式，将会对集群数据的读写带来不小的性能提升，而 HDFS 异构存储则能帮助我们更加合理地把数据存到应该存的地方。

1.1 HDFS 内存存储

HDFS 的内存存储是 HDFS 所有数据存储方式中比较特殊的一种，与之后将会提到的 HDFS 缓存有一些相同之处：都用机器的内存作为存储数据的载体。不同之处在于：HDFS 缓存需要用户主动设置目标待缓存的文件、目录，其间需要使用 HDFS 缓存管理命令。而 HDFS 内存存储策略：LAZY_PERSIST 则直接将内存作为数据存放的载体，可以这么理解，此时节点的内存也充当了一块“磁盘”。只要将文件设置为内存存储方式，最终会将其存储在节点的内存中。综合地看，HDFS 缓存更像是改进用户使用的一种功能，而 HDFS 内存存储则是从底层扩展了 HDFS 的数据存储方式。本节将对 HDFS 内存存储策略进行更细致的分析。

1.1.1 HDFS 内存存储原理

对于内存存储的存储策略，可能很多人会存有这么几种看法：

❑ 数据临时维持在内存中，服务一停止，数据全部丢失。

□ 数据存在于内存中，在服务停止时做持久化处理，最终将数据全部写入到磁盘。

仔细来看以上这 2 种观点，其实都有不小的瑕疵：

□ 第一个观点，服务一旦停止，内存数据全部丢失，这是无法接受的，我们只能容忍内存中少量的数据丢失。这个观点的另一个问题是，内存的存储空间是有限的，在服务运行过程中如果不及时处理一部分数据，内存空间迟早会被耗尽。

□ 第二个观点，在服务停止退出的时候做持久化操作，同样会面临上面提到的内存空间的限制问题。如果机器的内存足够大，数据可能会很多，那么最后写入磁盘的阶段速度会很慢。

所以一般情况下，通用的、比较好的做法是异步持久化，什么意思呢？在内存存储新数据的同时，持久化距离当前时刻最远（存储时间最早）的数据。换一个通俗的解释，好比有个内存数据块队列，在队列头部不断有新增的数据块插入，就是待存储的块，因为资源有限，需要把队列尾部的块，也就是更早些时间点的块持久化到磁盘中，这样才有空间存储新的块。然后形成这样的一个循环，新的块加入，老的块移除，保证了整体数据的更新。

HDFS 的 LAZY_PERSIST 内存存储策略用的就是这套方法，原理如图 1-1。

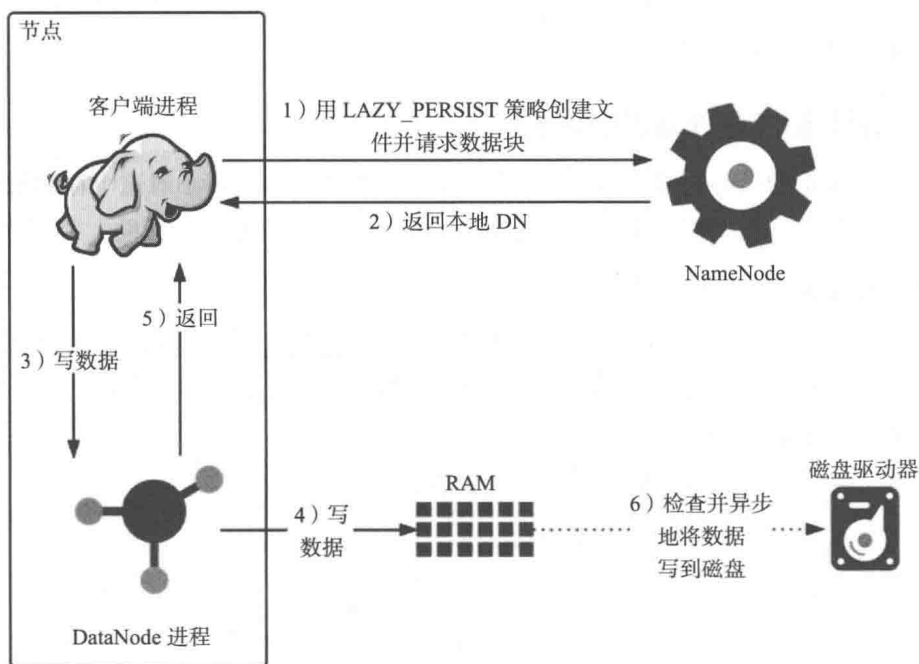


图 1-1 LAZY_PERSIST 策略原理图

上面描述的原理在图中的表示是第 4 个步骤和第 6 个步骤。第 4 步写数据到内存中，第 6 步异步地将数据写到磁盘。前面几个步骤是如何设置 StorageType 的操作，在下文中会

具体提到。所以异步存储的大体步骤可以归纳如下：

- 1) 对目标文件目录设置 `StoragePolicy` 为 `LAZY_PERSIST` 的内存存储策略。
- 2) 客户端进程向 `NameNode` 发起创建 / 写文件的请求。
- 3) 客户端请求到具体的 `DataNode` 后 `DataNode` 会把这些数据块写入 RAM 内存中，同时启动异步线程服务将内存数据持久化写到磁盘上。

内存的异步持久化存储是内存存储与其他介质存储不同的地方。这也是 `LAZY_PERSIST` 名称的源由，数据不是马上落盘，而是懒惰的、延时地进行处理。

1.1.2 Linux 虚拟内存盘

这里需要了解一个额外的知识点：Linux 虚拟内存盘。之前笔者也一直有个疑惑，内存也可以当作一个块盘使用？内存不就是临时存数据用的吗？于是在学习此模块知识之前，特意查了相关的资料。其实在 Linux 中，的确有将内存模拟为一个块盘的技术，叫**虚拟内存盘**（RAM disk）。这是一种模拟的盘，实际数据都是存放在内存中的。虚拟内存盘可以在某些特定的内存式存储文件系统下结合使用，比如 `tmpfs`、`ramfs`。关于 `tmpfs` 的具体内容，大家可以查阅维基百科等资料。通过此项技术，我们就可以将机器内存利用起来，作为一块独立的虚拟盘供 `DataNode` 使用了。

1.1.3 HDFS 的内存存储流程分析

下面讲述本章的核心内容：HDFS 内存存储的主要流程。不要小看这个存储策略，里面的过程可并不简单，在下面的内容中，笔者会给出比较多的过程图，帮助大家理解。

1. HDFS 文件内存存储策略设置

要想让文件数据存储到内存中，一开始要做的操作是设置此文件的存储策略，即上面提到的 `LAZY_PERSIST`，而不是使用默认的存储策略：`StoragePolicy.DEFAULT`，默认策略的存储介质是 `DISK` 类型的。设置存储策略的方法目前有以下 3 种：

第一种方法，通过命令行的方式，调用如下命令：

```
hdfs storagepolicies -setStoragePolicy -path <path> -policy LAZY_PERSIST
```

这种方式比较方便、快速。

第二种方法，调用对应的程序方法，比如调用暴露在外部的 `create` 文件方法，但是得带上参数 `CreateFlag.LAZY_PERSIST`。如下所示：

```
FSDataOutputStream fos =
    fs.create(
        path,
        FsPermission.getFileDefault(),
```

```
EnumSet.of(CreateFlag.CREATE, CreateFlag.LAZY_PERSIST),
bufferLength,
replicationFactor,
blockSize,
null);
```

上述方式最终调用的是 DFSCClient 的 create 同名方法，如下所示：

```
// DFSCClient 创建文件方法
public DFSOutputStream create(String src, FsPermission permission,
    EnumSet<CreateFlag> flag, short replication, long blockSize,
    Progressable progress, int buffersize, ChecksumOpt checksumOpt)
    throws IOException {
    return create(src, permission, flag, true,
        replication, blockSize, progress, buffersize, checksumOpt, null);
}
```

方法经过 RPC 层层调用，经过 FSNamesystem，最终会到 FSDirWriteFileOp 的 startFile 方法，在此方法内部，会有设置存储策略的动作：

```
static HdfsFileStatus startFile(
    FSNamesystem fsn, FSPermissionChecker pc, String src,
    PermissionStatus permissions, String holder, String clientMachine,
    EnumSet<CreateFlag> flag, boolean createParent,
    short replication, long blockSize,
    EncryptionKeyInfo ezInfo, INode.BlocksMapUpdateInfo toRemoveBlocks,
    boolean logRetryEntry)
    throws IOException {
    assert fsn.hasWriteLock();

    boolean create = flag.contains(CreateFlag.CREATE);
    boolean overwrite = flag.contains(CreateFlag.OVERWRITE);
    // 判断 CreateFlag 是否带有 LAZY_PERSIST 标识，来判断是否是内存存储策略
    boolean isLazyPersist = flag.contains(CreateFlag.LAZY_PERSIST);

    ...
    // 在此设置策略
    setNewINodeStoragePolicy(fsd.getBlockManager(), newNode, iip,
        isLazyPersist);
    fsd.getEditLog().logOpenFile(src, newNode, overwrite, logRetryEntry);
    if (NameNode.stateChangeLog.isDebugEnabled()) {
        NameNode.stateChangeLog.debug("DIR* NameSystem.startFile: added " +
            src + " inode " + newNode.getId() + " " + holder);
    }
    return FSDirStatAndListingOp.getFileInfo(fsd, src, false, isRawPath);
}
```

这部分的过程调用见图 1-2。